A Case Study in Verification of UML Statecharts: the PROFIsafe Protocol

R. Malik
Siemens Corporate Technology, CT SE 1
81730 München, Germany
robi.malik@nexgo.de

R. Mühlfeld
Siemens Corporate Technology, CT SE 5
Postfach 3220, 91050 Erlangen, Germany
reinhard.muehlfeld@siemens.com

12th February 2003

Abstract

We discuss our experience obtained during the PROFIsafe verification and test case generation project at Siemens Corporate Technology. In this project, a formal analysis of the PROFIsafe protocol for failsafe communication has been carried out, using the VALID Toolset. A formal model based on finite state machines has been obtained from the UML specification of the protocol. This model has been analysed with formal verification techniques, and several important properties have been proven. Based on the verified model, a set of test cases for the automatic execution of conformance tests has been derived. The paper points out important aspects and problems occurring during the modelling and verification of real industrial applications.

1 Introduction

In this paper, we present our experience in verification of UML statecharts, which we obtained when verifying the industrial field bus protocol PROFIsafe [3]. The protocol has been analysed in a project at Siemens Corporate Technology, using the VALID Toolset as a model checking environment. A preliminary report on this project has been presented in [5].

The behaviour of the PROFIsafe protocol is specified by means of UML statecharts. Its verification therefore leads to the general problem of verifying UML statecharts, which has been addressed by several other researchers, e.g. [4, 7]. In our project, the UML statecharts are restricted so as to have the unambiguous model semantics which is required to perform formal analysis.

In Section 2 we explain the relevant aspects of the PROFIsafe specification and the statecharts used. Afterwards, in Section 3, we discuss the abstraction steps needed in order to obtain a formal model of the protocol. Then, in Section 4 and 5 we explain the verification and the test case generation for PROFIsafe.

2 The PROFIsafe Protocol

The PROFIsafe protocol [3] is used for *failsafe communication* between two agents using an insecure communication medium. The aim of failsafe communication is to ensure that two communication partners always enter a defined safe state in the event of any kind of communication failure. This is important in many technical systems, where a high level of safety is required.

Although originally defined as an extension of the PROFIBUS field bus protocol [2], the PROFIsafe protocol can be used to establish failsafe communication based on any underlying communication medium. Independently of the communication medium, it has to ensure a maximum error rate of one unnoticed fault in 10^9 hours [3].

The protocol defines communication between two distinguished communication partners, called *host* and *slave*, who exchange messages via the underlying communication medium, called *grey channel* (Figure 1). The host usually runs on a controlling computer collecting data from and sending commands to multiple slaves. The slave usually runs on a technical device controlled by the host.

No assumptions are made about the grey channel. It may produce all kinds of communication failures, such as delaying, modification, duplication, or loss of messages. However, in most practical applications, such errors are assumed to occur only sparsely. The objective of PROFIsafe is to detect these sparse failures, should they occur, and to switch host and slave to their defined safe states before the fault can cause any harm.

The PROFIsafe profile [3] defines a new layer of safe communication, by specifying two new components called *F-host* and *F-slave*[1]. These components use the grey channel as an underlying communication medium, but provide a means of failsafe communication to their users, represented by the *application processes* of the host and slave (Figure 1).

The basic idea of the protocol consists of sending and monitoring live signs in the form of consecutive numbers in combination with timers. To this end, the PROFIsafe profile defines the following additional information to be put into all messages sent via the insecure grey channel.

Consecutive number. Each message is equipped with a consecutive number, which is used by the recipient for monitoring the life of the sender and the communication link. Both communication partners continuously check whether the other partner manages to update the consecutive number, before a defined *watchdog time* has elapsed.

Eight bits are reserved for the consecutive number. The value 0 is reserved for the first protocol cycle. Afterwards, the consecutive number runs in cyclic mode from $1 \ldots 255$, wrapping over back to 1 at the end.

CRC2 checksum. Each message is also equipped with an additional CRC checksum. This checksum is used to detect fake messages and corrupted messages, which have slipped undetected through the grey channel.

Status byte. Each acknowledgement sent from the F-slave to the F-host contains an additional status byte, with individual bits reserved for the different possible faults. In this way, the F-slave informs the F-host that it has detected a certain error, e.g. a CRC fault.

The PROFIsafe profile provides a formal specification in UML of the behaviour of the PROFIsafe F-host and F-slave. As an example, Figure 2 shows the statechart defining the behaviour of the PROFIsafe F-host.

The statecharts used here are very simple, exploiting only few features of the rich statecharts language provided by UML [8]. For example, there is no state hierarchy, and there are no entry, exit, or internal actions associated to states. In this way, the model remains very clear and avoids the ambiguities related to advanced features of UML statecharts [7]. Also, the developers of this specification can be very sure that it will be interpreted in the same way by different readers and UML tools.

The description of the grey channel given in the PROFIsafe profile allows for the possibility of communication faults. This can be captured by a simple extension of the general message passing scheme in UML. Activity states, in which a communication partner is waiting for new messages to arrive, are highlighted in the PROFIsafe state diagrams (Figure 2), in order to make the synchronisation constraints explicit.

3 Creating a Formal Model

In order to verify the model, a rigorous formal description needs to be extracted from the statecharts. This step involves adding the formal details which are missing in the semi-formal guards and actions labelling the transitions. Based on the

[1] The 'F' in F-host and F-slave, e.g., is used throughout the PROFIsafe profile to identify the 'failsafe' components introduced.

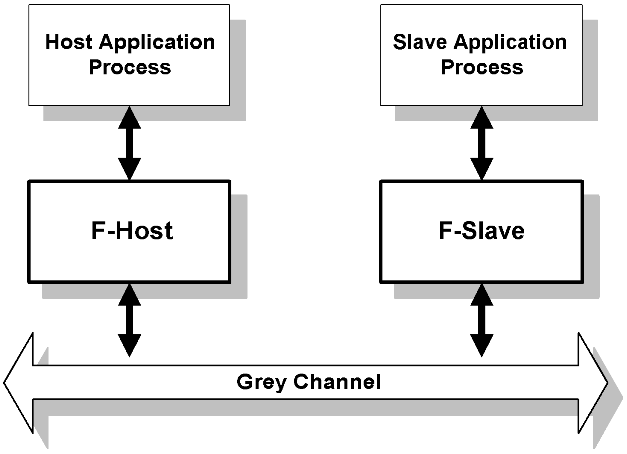

Figure 1: PROFIsafe architecture.

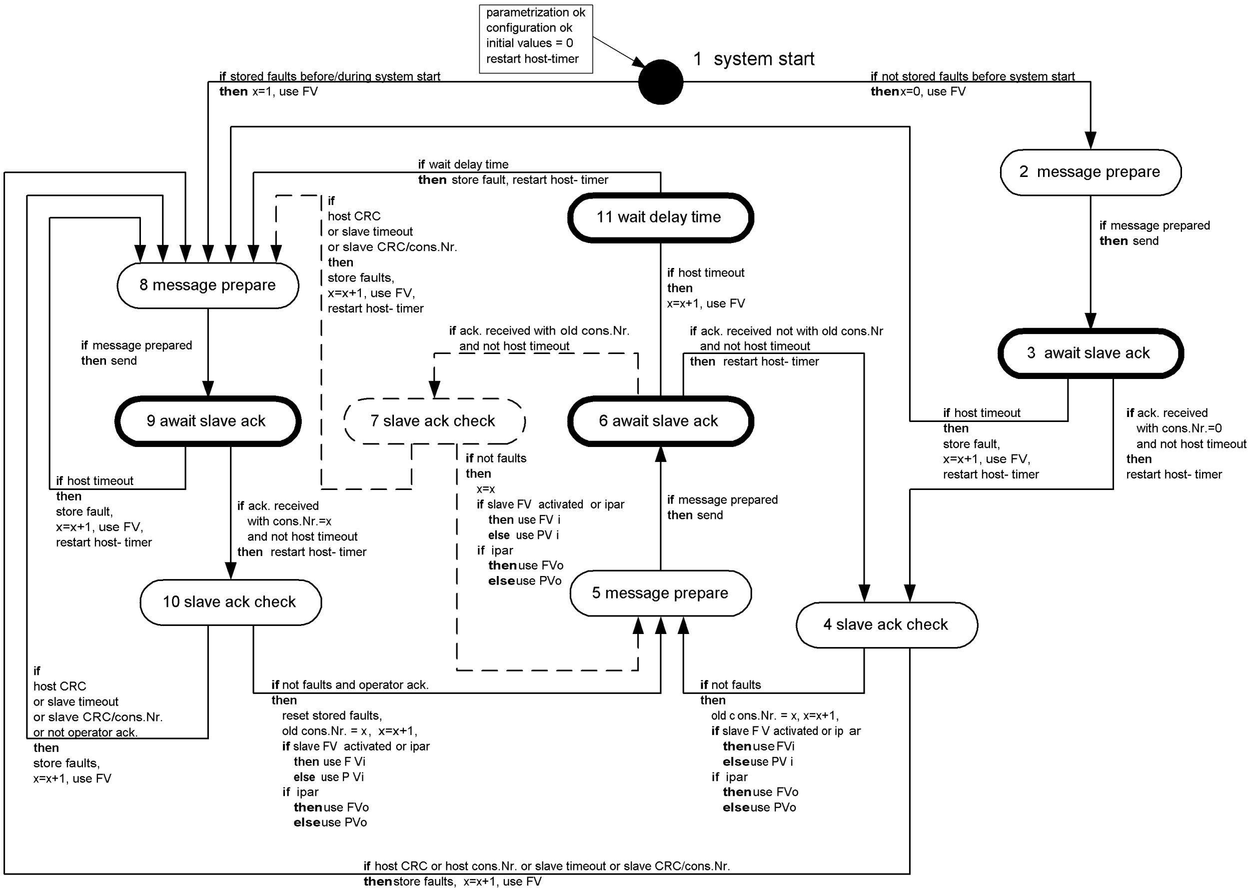

Figure 2: Statechart defining behaviour of PROFIsafe F-host.

detailed statecharts, a formal description using finite state machines is computed.

But before creating this formal description, we need to abstract from some of the data used by the protocol, and we need to introduce a simple description of the behaviour of the grey channel. These tasks are not only needed in order to cope with the complexity of the model, but also to obtain a meaningful verification model at all.

We use the following data abstractions.

- The verification model abstracts away from all the user data. It deals only with the logic behaviour of the two communication partners; it cannot be checked whether any data values are transferred correctly.

- The CRC2 checksum, which in the real model is an integer of 16 or 32 bits, is not modelled explicitly. Instead, it is replaced by a single bit, containing only the information whether a message contains a correct or an incorrect checksum. This completely suffices to analyse the logic behaviour of the protocol, but of course makes it impossible to analyse the correctness of the CRC computation algorithm.

- The verification model assumes a maximum sequence number of 4, thus considering only a small part of the original range from 0 to 255. This simplification can be justified by the observation that even the simplified model includes any malicious behaviour of the grey channel that may occur. Moreover, it is harder to ensure the requirements of failsafe communication if fewer sequence numbers are available; therefore the correctness of such properties for the simplified model also implies these properties to be correct for the original model.

In order to perform formal verification, it also is essential to formalise the behaviour of the environment in which the system to be verified runs, i.e. the grey channel. The behaviour of the grey channel is described verbally in the PROFIsafe profile. For verification purposes, we need to model the possibility of communication faults.

We use a grey channel of limited buffering capacity which can delete, modify, and duplicate messages, and produce fake messages. But if a message is modified or a fake message is produced, we assume the CRC2 checksum of the delivered message to be incorrect. Without such additional assumptions, it is not possible to prove any useful properties of a communication protocol.

Most verification tasks of industrial applications require similar kinds of abstraction and additional modelling. Unfortunately, carrying out such abstractions is a difficult task which cannot be performed automatically. The authors believe that this is one of the major obstacles preventing formal verification from becoming commonly used in industry.

4 Verification

We use the model checker of the VALID Toolset to verify the PROFIsafe protocol. The VALID Toolset, developed at Siemens Corporate Technology, supports the modelling, verification and code generation of finite state machines as described in [6, 9]. Accordingly, the formalised and abstracted statecharts specifying the PROFIsafe protocol are translated into finite state machines.

The translation involves creating a finite state machine for each statechart, and for each variable used by the protocol. For example, there is a finite state machine for the variable representing the sequence number used by each communication partner, with one state for each possible value of this variable. All these state machines are synchronised appropriately, so as to capture the behaviour modelled by the UML statecharts.

Based on the translated model of the PROFIsafe protocol, we have performed several kinds of formal analysis. Firstly, we have performed some standard checks in order to find out whether the logic specified by the statecharts is consistent. We have proven that the protocol is free from deadlocks and livelocks, that the F-host and F-slave can always react to any messages or timeouts which they may receive, and that the PROFIsafe profile defines a deterministic behaviour for the F-host and F-slave.

In a second round of verification, we have defined and checked application-specific properties of the PROFIsafe protocol. Most importantly, we have examined whether the requirements of failsafe communication are guaranteed by the protocol. We have checked whether the occurrence of a fault causes both communication partners to switch to their failsafe states within the required delay time. This analysis produced a few counter-examples, which pointed to some problems in an early version of PROFIsafe, and which were used to improve the next version of the profile [3].

5 Automated Testing

Based on the verified model, we have computed test suites, which can be used to execute conformance tests in order to ascertain that a protocol implementation has the same behaviour as the verified model. These test cases will be made available to vendors of PROFIBUS applications, who will be able to obtain certificates of conformance for their implementations more easily.

VALID can automatically compute a set of test cases satisfying certain coverage criteria for a given model. In this way, we obtain a test suite covering a well-defined set of situations to be tested, and which therefore is sufficient to perform a thorough test. Since the test case generation is based on the verified model, any implementation passing these test cases will most probably also satisfy the verified properties.

The task of test case generation has been carried out for the PROFIsafe F-slave model, and the results are currently used to implement an automated testing environment for F-slave implementations. In a second step, we plan to do the same for the PROFIsafe F-host.

6 Conclusions

We have discussed the verification and test case generation for an industrial field bus protocol. The protocol is specified using a restricted version of UML statecharts, which is free from ambiguities. After applying some abstractions, the statecharts are translated into finite state machines and analysed using the VALID Toolset.

Based on the restricted statecharts language, it is possible to obtain expressive verification results. Furthermore, it is possible to generate test suites with guaranteed coverage, which will be used for automated conformance tests of PROFIsafe implementations.

Acknowledgements

The authors would like to thank Bertil Brandin for his support and helpful comments in preparing this abstract.

References

[1] B. A. Brandin, R. Malik, P. Dietrich, "Incremental system verification and synthesis of minimally restrictive behaviours." American Control Conference (ACC 2000), Chicago, 2000.

[2] EN 50170, European Standard for Profibus-DP and FMS.

[3] Profibus Nutzerorganisation e. V., "PROFIsafe—profile for safety technology." Version 1.12, 2002.

[4] D. Latella, I. Majzik, M. Massink, "Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model checker." Formal Aspects Comp. Sci., **11**, 637–664, 1999.

[5] R. Mühlfeld, R. Malik, "Anwenderbericht: Verifikation von UML-Statecharts am Beispiel des PROFIsafe-Profils." Rational-Anwenderkonferenz (RAK 2002), 2002.

[6] P. J. Ramadge, W. M. Wonham, "The control of discrete-event systems." IEEE Proceedings, **77** (1), 81–98, 1989.

[7] T. Schäfer, A. Knapp, S. Merz, "Model checking UML state machines and collaborations." In: S. D. Stoller, W. Visser, "Electronic Notes in Theoretical Computer Science", **55** (3), 1–13, Elsevier, 2001.

[8] Object Management Group, "Unified modelling language specification, Version 1.3." OMG, 2001; available at `http://www.omg.org`.

[9] W. M. Wonham, "Notes on control of discrete event systems." Systems Control Group, Department of Electrical and Computer Engineering, University of Toronto, 1999; available at `http://www.control.utoronto.ca/` under "Research".