Formal Specification of Safe Software with Z and Real-Time CSP

Maritta Heisel, Carsten Sühl
Technische Universität Berlin*

Extended Abstract

1 Introduction

We present a method to ensure the safety of systems being controlled at least partially by software components. Safety can be defined as the property of a system to be free from accidents or losses (cf. [Lev95]). It follows that a software component which is considered in isolation cannot be unsafe because it is not directly able to cause a loss event. Safety is a property of a whole system in the context of its environment rather than a property of a separate system component. A method concerned with *software* development for safety-critical systems must aim at *system* safety and can only be evaluated in this respect.

The proposed method uses the specification language Z [Spi92] to express the functional aspects and the safety constraints of the system and real-time CSP [DS95] to model its dynamic behavior.

2 Underlying System Model

We assume that there is a technical process whose control component is at least partially realized by software, see Figure 1 and [Lev95]. Such a software component affects certain process variables (*manipulated variables*) by means of sending commands to actuators. By evaluating the current states of certain process variables which are measured by sensors (*controlled variables*) the control component is able to approximate the current state of the real process in order to verify the realization of the commands sent to the actuators within the process (feedback control).

The behavior of the technical process does not only depend on internal conditions within the process, e.g. the state of the manipulated variables, but it is also influenced by external disturbances. The basic objective of process control is to achieve the process control function in spite of disturbances from the environment. Hence, the following subsystems of a technical process must be modeled:

- *all* parts of the process-control component, i.e. software components, mechanical and electrical components, and interfaces to human operators,
- sensors, determining the projection of the real process state to the internal state of the control component, and

*Franklinstr. 28/29, Sekr. FR 5-6, D-10587 Berlin, Germany. email: {heisel, suehl}@cs.tu-berlin.de, fax: +49-30-314-73488

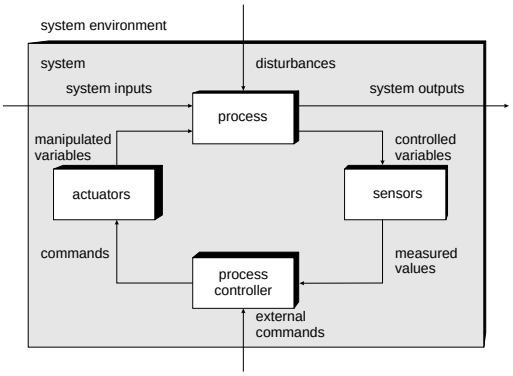

Figure 1: System Model

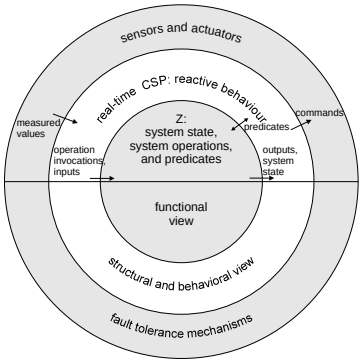

Figure 2: Software Model

- actuators, which determine the realization of commands given by the control component within the real process.

At this point, the essential difference between *correctness* and *safety* becomes clear. The term correctness is defined as the property of a software component to fulfill the relation between inputs and outputs prescribed in the component specification. Thus, incorrect measurements of process variables or measurements of process variables arriving from the sensors at the wrong time as well as the failed realization of given commands by the actuators are not relevant in the context of correctness. In contrast, the notion of safety is defined as the system property to be free from accidents or losses. In this case, the thorough examination of the above situations is a necessary precondition.

3 Stages of Development

Although non-software-based components have to be taken into account in the modeling process,the subject of the development process are merely the software-based components of the control system. The development of these software components is performed in a number of stages that will be applied with repetitions and in an interleaved manner. These stages are: hazard analysis, formal specification, validation of the formal specification, stepwise refinement, and program synthesis. In this paper, we only consider the first three activities, where the main focus is on the development of the formal specification.

3.1 Hazard Analysis

The objective of this stage is to identify hazards, analyze their causes and consequences, design safeguards for their control or elimination, and assess risks.

After identifying the system-level hazards by using a hazard identification approach (which is not part of the proposed method), the system hazards should be traced back to the interface of the intended software components by fault tree analysis. The subsequent qualitative analysis of the developed fault tree yields an informal description of the safety constraints for the software to be developed.

3.2 Formal Specification

The essential task of this stage is to formally define the safety constraints related to software components and to show that the defined safety constraints are logical consequences of the formally specified properties of the software components.

In general, the control component of a technical process refers to a *reactive* system that is event-triggered. It continuously has to react to events within the environment by invoking internal operations and subsequently emitting resulting events to the environment. Consequently, the dynamic behavior of the software component has to be specified. Here, real-time requirements are of major importance. Furthermore, the functional aspects of the software components must be specified, i.e. the functional behavior of the operations and the data structures.

These requirements to the expressive power lead us to use two different specification languages: Z [Spi92] to specify the system state and the system operations and Real-Time CSP [DS95] to define the reactive behavior of the control component including the real-time requirements. To achieve a combination of both formalisms, we propose the software model shown in Figure 2:

1. The innermost component which is expressed in Z specifies the functional aspects, i.e. the structure and the properties of a valid system state as well as the system operations.

2. The CSP process specifies the reactive behavior, i.e. the absorption of values provided by the sensors, the invocation of internal operations, and the transmission of the operation results to the actuators.

3. The outermost component makes it possible to specify the required behavior of sensors and actuators as constituents of the environment of the control component. It is possible to specify fault tolerance mechanisms, e.g. redundant arrangements of sensors and actuators.

For conceptual reasons, it is useful to distinguish between two architectural classes of control components, which might be called time-triggered and event-triggered architectures. In the first case, the control component is connected with simple sensors measuring certain process variables which are permanently available to the controller. In the second case, there are autonomous sensors, which are able to force a reaction within the control component when a particular condition in the environment takes place.

3.3 Validation of the Formal Specification

According to historical experience, a major part of the accidents which were caused by incorrect behavior of software components can be traced back to incomplete or inconsistent requirements specifications. Therefore, we present a list of criteria to check a formal specification developed in compliance to the proposed approach for completeness and consistency. These criteria are heuristic rules for identifying general sources of faults rather than formal completeness criteria.

4 Example

As an example for the specification stage, we consider a simple heating system consisting of a controller, a thermometer, a thermostat, and a piece of bimetal. Its function is to adjust the temperature within a specified interval, delimited by the values *NOMINAL_MIN* and *NOMINAL_MAX* respectively. To accomplish this task the controller determines the temperature by reading the thermometer and gives corresponding commands to the thermostat in equidistant instants of time.

The controller is able to instruct the thermostat to increase, to decrease, or to maintain the current temperature as well as to cut off the heating system[1].

The safety constraint is that the real temperature must not exceed a prescribed maximum MAX; otherwise the environment is in danger. For fault tolerance, there is a supplementary piece of bimetal which alerts the controller in case the temperature exceeds the maximum without the thermometer realizing this situation.

We start with the Z part of the specification. The system can be in the following modes:

$$MODE ::= MAINTAIN \mid INCREASE \mid DECREASE \mid CUTOFF$$

Using this global definition, the system state can be specified as follows:

HeatingSystem

$$temperature : \mathbb{Z}$$
$$mode : MODE$$

$$temperature > MAX \Rightarrow mode = CUTOFF$$
$$temperature < NOMINAL\_MIN \Rightarrow mode = INCREASE$$
$$NOMINAL\_MAX < temperature \leq MAX \Rightarrow mode = DECREASE$$
$$NOMINAL\_MIN \leq temperature \leq NOMINAL\_MAX \Rightarrow mode = MAINTAIN$$

The only internal operation gets the measured temperature as its input and yields as a command the new mode as specified in *HeatingSystem*.

Regulation

$$\Delta HeatingSystem$$
$$measured? : \mathbb{Z}$$
$$command! : MODE$$

$$temperature' = measured?$$
$$command! = mode'$$

Finally, we need a predicate on the global system state that will be used in the CSP process specifying the dynamic behavior of the system.

$$Exceeding \mathrel{\widehat{=}} [\, HeatingSystem \mid mode = CUTOFF \,]$$

This concludes the innermost component of the software model. We continue with the CSP process *ThermostatControl*. Its alphabet, i.e. the set of events whose occurrences are influenced by the process consists of the events *Regulation* (which is meant to be identical to the Z schema with the same name) and *emergency*. The *emergency* event occurs if the redundant sensor detects a violation of the safety constraint.

$$\begin{aligned} ThermostatControl = {} & thermometer\_channel?measured \rightarrow measured\_input!measured \rightarrow \\ & Regulation \rightarrow command\_output?command \rightarrow thermostat\_channel!command \rightarrow \\ & \text{if } Exceeding \text{ then } Stop \text{ else } Wait\ INTERVAL;\ ThermostatControl \text{ fi} \\ \triangle\ & emergency \rightarrow thermostat\_channel!CUTOFF \rightarrow Stop \end{aligned}$$

First, the value of the thermometer is read via the *thermometer_channel* and given to the Z operation via the channel *measured_input*. Then, the *Regulation* operation is invoked. It

[1] In the full paper, a more elaborate example will be presented in more detail.

determines the new mode of the system which is read via the *command_output* channel and given to the actuator via the *thermostat_channel*. If the predicate *Exceeding* is true, the system switches off the heating system and stops. Otherwise, before the considered process definition is called recursively. A *Wait* process delays the execution for some time interval (to be specified in Z). Without this delay, the described sequence of events would take no time, so that the recursive definition would permit an infinite number of events during a finite time interval. The first part of the process can always be interrupted by the *emergency* event which also causes the system to enter a fail-safe state and then to stop.

In addition to specifying behavior by process expressions, real-time CSP allows us to express the requirement that the process environment be ready to accept the *Regulation* event at any time:

$$EnvironmentalAssumption = (\forall\, t : [0, \infty) \bullet Regulation \ \mathsf{open}\ t)$$

5 Combination of Formalisms

Besides presenting a methodology for the specification of software in a safety context, a major contribution of the paper is to establish a connection between the two formal specification languages. We define relations between the elements of the CSP part and the Z part in an informal but systematic manner.

1. There are two possibilities to refer to the system operations specified in Z in the CSP part of the specification: either define one corresponding event to represent the whole execution of the operation, if the duration of execution is considered to be negligible, or define two corresponding events, representing the invocation and termination of the operation, respectively.

2. For each input of a system operation there is a communication channel in the CSP part, onto which an input value derived from sensor measurements is written. Analogously, for each output of a system operation there is a communication channel, from which the output value of the operation is read and then forwarded to the actuators in form of commands.

3. As the dynamic behavior of the control component may depend on the current system state, the definitions of the CSP processes must be able to refer to predicates on the system state. To this end, predicates in the Z part of the specification are defined to which the CSP part can refer.

In addition, we show how to prove properties of the specification that can only be expressed by referring to the Z as well as the CSP part.

The connection between the CSP part and the specification of the intended behavior of the sensors and actuators is as follows. The CSP part is linked with every sensor via a communication channel from which the measured values of the respective sensor are read. Analogously, the CSP part is connected with every actuator via a communication channel onto which the commands to the respective actuator are written.

The specification of a communication channel in terms of CSP processes makes it feasible to model aspects of a distributed communication, for example the delay of transmission or the redundant arrangement of unreliable communication channels.

The idea to use different formal or semi-formal languages to specify a system adequately is not new: Weber [Web96] combines Z and the graphical language Statecharts for purposes similar to ours. Since Statecharts are a semi-formal specification technique, the resulting specification is not completely formal. Using a formal language like CSP leaves us the possibility to work

towards a formal combination of the two languages, thus obtaining completely formal combined specifications.

A combination of Z and CSP has already been used to model security properties [RWW94]. This approach starts with a Z specification which is later transformed into an equivalent CSP definition. That work shows that our approach of combination — even if in a complementary manner, is a reasonable choice.

References

[DS95] Jim Davies and Steve Schneider. Real-time CSP. Available via ftp from ftp.comlab.ox.ac.uk, 1995.

[Lev95] Nancy G. Leveson. *Safeware: System Safety and Computers.* Addison-Wesley, 1995.

[RWW94] A.W. Roscoe, J.C.P. Woodcock and L. Wulf. *Non-interference through Determinism.* Oxford University Computing Laboratory, 1994.

[Spi92] J.M. Spivey. *The Z Notation: A Reference Manual.* Prentice-Hall, 2nd edition, 1992.

[Web96] Matthias Weber. Combining Statecharts and Z for the design of safety-critical systems. In *Proceedings Formal Methods Europe, to appear*, 1996.