Modeling Safety-Critical Systems with Z and Petri Nets

Monika Heiner[1] and Maritta Heisel[2]

[1] Brandenburgische Technische Universität Cottbus, Institut für Informatik, D-03013 Cottbus, email: mh@informatik.tu-cottbus.de

[2] Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik, Institut für Verteilte Systeme, D-39016 Magdeburg, Germany, email: heisel@cs.uni-magdeburg.de

Abstract. We show how to combine the specification notation Z with Petri nets for modeling safety-critical systems. The combination preserves the strengths of the two formalisms, while ameliorating their drawbacks. We illustrate our approach by modeling a part of a production cell and validating that model with respect to safety-related properties.

1 Introduction

Petri nets [Sta90] are a well-established formalism for modeling the behavior of concurrent systems. They have a formal semantics and can be animated by tools. Moreover, there exist sophisticated analysis techniques to demonstrate properties of Petri net models. Animation and validation are particularly important for safety-critical systems, making Petri nets a suitable formalism to use in that area.

A drawback of Petri nets is that they tend to become very large for systems of realistic size. Taking into account not only behavioral but also data-related aspects of the modeled system further increases the size of the Petri net. Data-oriented aspects concern the internal data that the system must maintain to adequately react to environmental or internal conditions. Safety-critical systems (like other computerized systems, too) usually need such an internal data state.

In contrast to Petri nets, the specification notation Z [Spi92b] was designed to specify data and the evolution of data. It does not provide any means to explicitly specify behavior. In Z, we can only specify sets of operations, but we cannot express that we want to occur the operations in a certain order. As compared to other formal specification languages, Z is fairly well-accepted and well equipped with tools, such as type checkers [Spi92a,BGHH98] and theorem provers [KSW96,Saa97].

To adequately specify safety-critical systems, both aspects, behavioral as well as data-oriented ones, must be taken into account. Therefore, we investigate a combination of Z and Petri nets. We use Z to specify the data-oriented aspects of the system, and Petri nets to specify its behavioral aspects. Combining the two languages, we achieve a separation of concerns, which results in better comprehensible, smaller and thus better analyzable system models. Hence, the combination keeps the advantages of both specification formalisms, while ameliorating their drawbacks.

In Section 2, we describe the way in which systems are modeled, using the two formalisms. Section 3 is devoted to a case study that illustrates the approach and shows how combined specifications can be validated. We conclude by comparing our combination of Z and Petri nets with other combinations of data-based and behavioral specification formalisms and by pointing out directions for future research (Section 4).

M. Felici, K. Kanoun, A. Pasquini (Eds.): SAFECOMP'99, LNCS 1698, pp. 361-374, 1999

2 Modeling Principles

A goal of our combination of Z and Petri nets is to obtain nets that are much smaller and better analyzable than when using Petri nets alone to model a safety-critical system. When using the combination, data aspects need not be encoded in the nets, but can be specified in Z.

We define the system state and operations that specify how that state can evolve in Z. In Z, it cannot directly be expressed in which order the Z operations should "happen" (indeed, there is not even a notion of executing an operation in Z). To express behavior, we use Petri nets, where Z operations correspond to transitions of the Petri net. Moreover, we assume that Z operations can only be "executed" if their precondition is fulfilled. The precondition of an operation states that there exists an after-state and values for the output variables such that the schema predicate is fulfilled. In this paper, we always give the preconditions explicitly. The following figure illustrates this approach:

The model of a safety-critical system is then made up of both specifications, i.e., the *conjunction* of the constraints imposed by the two specifications must be fulfilled. To obtain a useful system model by combining specifications in different formalisms, we must show that the two specifications do not contradict each other. To ensure compatibility of the two specifications, we have identified several proof obligations. Checking these compatibility conditions may reveal errors in the model and thus contributes to the quality of the specification.

- The initial marking of the Petri net must be consistent with the initiality conditions of the Z specification.
- The conditions associated with incoming places of transitions correspond to preconditions of Z operations, the conditions associated with outgoing places of transitions correspond to postconditions established by Z operations. Hence, for chains we have the obligation to show that the postcondition of an operation implies the precondition of its successor in the chain.
- If the Petri net admits concurrent execution of operations op_1 and op_2 that work on common state components, we must show that
  - the operations do not exclude each other, i.e., $\neg\ (\mathsf{pre}\ op_1 \land \mathsf{pre}\ op_2 \Leftrightarrow false)$
  - for all states where $\mathsf{pre}\ op_1 \land \mathsf{pre}\ op_2$ holds, both orders are possible and lead to the same final state. This allows us to use an interleaving semantics of concurrency for Z operations.

Usually, a transition is enabled if the precondition of its corresponding Z operation holds. But to keep the Petri net sizes small by concentrating on the essential control

flow, the precondition of a transition may be weaker than the precondition of its Z operation. To highlight this fact, the letter "Z" appears in the transition symbol (see Figure 2). Z-labeled transitions, if considered only on the Petri net level, exhibit more behavior than allowed by the Z specification. This mechanism may be used to resolve dynamic conflicts in the combined Z - Petri net specification. If two transitions are in a dynamic conflict, whose corresponding Z operations have incompatible preconditions, then only the transition can actually take place whose corresponding Z operation holds, if any. The fact that the Petri net considered in isolation can engage in more behavior than permitted by the Z specification is not a problem when we analyze the Petri net for safety-related properties. There, we show that unsafe system states cannot be entered. Therefore, if the net with the more liberal behavior is safe, than the more restricted behavior is also safe.

To obtain self-contained and analyzable models, we not only consider the control software, but also model parts of the environment, for example sensors (see Section 3).

3 Case Study: Production Cell

To illustrate our approach, we model a part of a production cell [LL95]. The production cell, an existing industrial facility, consists of six physical components: two conveyor belts, a rotable robot equipped with two extendable arms, an elevating rotary table, a press, and a traveling crane (which has been added to make the cell self-contained). The machines are organized in a (closed) pipeline, see Figure 1. Their common goal is to transport and process metal plates. Altogether, 14 sensors and 34 actuators can be used to control the cell.

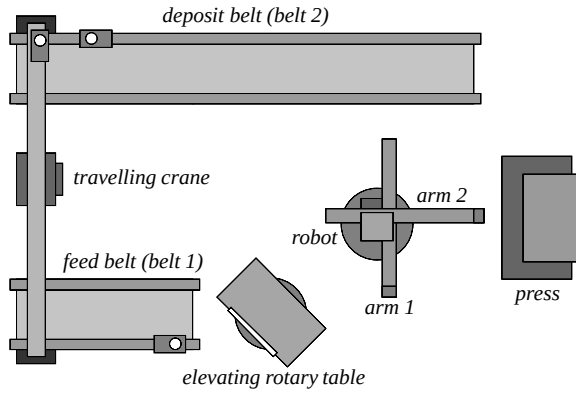

Fig. 1. Production cell

As a first step to develop the control software, we develop a formal and executable system specification. To save space, we restrict ourselves to the first two components, the feed belt and the elevating rotary table, which are quite different technical devices.

While the feed belt may just be switched on and off, the motion of the table is controlled by position engines. In addition, the feed belt has two sensors that indicate whether a plate is at its front or end, respectively.

The original task description considers only the case where at most one plate is on the feed belt at a time. In our model, the maximal number of plates allowed to be present on the feed belt (henceforth called feed belt capacity) is a parameter of the specification. In this way, our model is more general than required by the original task description.

In Section 3.1, we model the two subsystems in Z and specify how their state can change via operations. The order in which the state-changing operations can occur is specified by a Petri net given in Section 3.2. In Section 3.3, the Petri net is analyzed using tools. The coherence of the two specifications is demonstrated in Section 3.4, and some application-dependent safety-related properties of the model are analyzed in Section 3.5.

3.1 Z Part of the Specification

The specification models the two subsystems feed belt and table separately[1]. It exhibits the situations where the two subsystems must communicate or cooperate to achieve a desired state transition. For validation purposes, the whole Z specification was type-checked.[2]

States of the Subsystems For the feed belt, we need to know whether it is switched on (i.e., whether it moves) or not, and how many plates it carries. We partition the feed belt into three zones: the front, where new plates are dropped, and where a sensor signals the presence of a plate; the end, where the plates are passed on to the elevating rotary table, which is signaled by another sensor; and the zone in between the range of the two sensors.

$$
\begin{array}{|l}
\hline
feed\_belt \\
\hline
at\_front, at\_end : 0 \,..\, 1 \\
in\_between, number\_of\_plates : 0 \,..\, maxplates \\
fb\_mvt : OnOff \\
\hline
number\_of\_plates = \\
\qquad at\_front + in\_between + at\_end \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
Init\_feed\_belt \\
\hline
feed\_belt' \\
\hline
number\_of\_plates' = 0 \\
fb\_mvt' = off \\
\hline
\end{array}
$$

Here, $maxplates$ is the feed belt capacity, and the type $OnOff$ is defined as an enumeration type $OnOff ::= on \mid off$. In its initial state, the feed belt is switched off, and there are no plates on it. The decoration "′" of variable names means that they describe the state *after* an operation is completed. Plain variables describe the state in which an operation is started.

[1] A first version of this specification was based on the partial specification given in [LS96].

[2] Readers not familiar with Z are referred to [Spi92b].

For the elevating rotary table (just called "table" in the following), we need to know its position (modeled by a basic type $Table\_Position$), whether it moves or not, and whether there is a plate on it or not. The type $YesNo$ is defined as $YesNo ::= yes \mid no$. Whether or not the table is ready to receive a plate is expressed by the derived state component $can\_receive$. The two extreme positions of the table are called $load\_position$ and $unload\_position$.

table

$t\_loaded : YesNo$
$t\_position : Table\_Position$
$t\_mvt : OnOff$
$can\_receive : YesNo$

$can\_receive = yes \Leftrightarrow$
$\quad t\_position = load\_position \wedge$
$\quad t\_loaded = no \wedge t\_mvt = off$

Init_table

$table'$

$t\_loaded' = no$
$t\_position' = unload\_position$
$t\_mvt' = off$

Table control operations We assume a total ordering relation $<$ on the type $Table\_Position$, where $load\_position$ is the smallest and $unload\_position$ is the largest position. The function $next$ increases the position by one unit, the function $prev$ decreases it. "$\Delta table$" means that the state of the table may change.

start_load_to_unload

$\Delta table$

$t\_loaded = yes$
$t\_position = load\_position$
$t\_mvt = off$

$t\_loaded' = yes$
$t\_position' = load\_position$
$t\_mvt' = on$

move_load_to_unload

$\Delta table$

$t\_loaded = yes$
$t\_position < unload\_position$
$t\_mvt = on$

$t\_loaded' = yes$
$t\_position' = next(t\_position)$
$t\_mvt' = on$

The operation $start\_load\_to\_unload$ starts the motor in the $load\_position$, the operation $move\_load\_to\_unload$ increases the position of the table by one unit. When the table has reached the $unload\_position$, the motor must be switched off, as specified by the operation $stop\_at\_unload$. Because we do not model how the plate is passed on from the table to the robot, the operation $unload\_table$ just resets the state component t_loaded, i.e., $unload\_table$ acts as a consumer.

stop_at_unload

$\Delta table$

$t\_loaded = yes$
$t\_position = unload\_position$
$t\_mvt = on$

$t\_loaded' = yes$
$t\_position' = unload\_position$
$t\_mvt' = off$

unload_table

$\Delta table$

$t\_loaded = yes$
$t\_position = unload\_position$
$t\_mvt = off$

$t\_loaded' = no$
$t\_position' = unload\_position$
$t\_mvt' = off$

The operations $start\_unload\_to\_load$, $move\_unload\_to\_load$, and $stop\_at\_load$ are defined analogously.

Operations related to the feed belt environment These operations correspond to phenomena that cannot be influenced by the control software of the feed belt but are reported by sensors. Plates are dropped on the feed belt by a producer, which causes the sensor at the front of the feed belt to respond (operation $load\_fb$). When the feed belt is moving, the sensor will eventually report that there is no longer a plate at the front of the feed belt, as specified by the operation $leave\_front$. Furthermore, the sensor situated at the end of the feed belt will eventually report that a plate has reached the point where it can be passed on to the table (operation $detect$).

load_fb

$\Delta feed\_belt$

$number\_of\_plates < maxplates$
$at\_front = 0$

$at\_front' = 1$
$in\_between' = in\_between$
$at\_end' = at\_end$
$fb\_mvt' = fb\_mvt$

leave_front

$\Delta feed\_belt$

$fb\_mvt = on$
$at\_front = 1$

$in\_between' = in\_between + 1$
$at\_front' = 0$
$at\_end' = at\_end$
$fb\_mvt' = fb\_mvt$

detect

$\Delta feed\_belt$

$fb\_mvt = on$
$at\_end = 0$
$in\_between > 0$

$at\_end' = 1$
$in\_between' = in\_between - 1$
$at\_front' = at\_front$
$fb\_mvt' = on$

Feed belt control operations The feed belt controller must decide when to switch on or off the feed belt. To take these decisions in such a way that the safety of the system is guaranteed, it must communicate with the table, i.e., the feed belt control operations import either $\Xi table$ (if the state of the table is only queried but not changed) or $\Delta table$.

The operation $switch\_on$ specifies that the feed belt may only be switched on if the table is ready to receive a plate or if there is no plate at the end of the feed belt. The feed belt must be switched off if a plate has arrived at the end of the feed belt but the table is not ready to receive it (operation $switch\_off$). Otherwise, the plate is passed on from the feed belt to the table, as specified by the operation fb_to_table.

Note that the Z specification cannot entirely describe the behavior of the production cell. It only restricts possible behavior via preconditions of operations.

switch_on

$\Delta feed\_belt$
$\Xi table$

$fb\_mvt = off$
$number\_of\_plates > 0$
$can\_receive = yes \lor at\_end = 0$

$fb\_mvt' = on$
$in\_between' = in\_between$
$at\_front' = at\_front$
$at\_end' = at\_end$

switch_off

$\Delta feed\_belt$
$\Xi table$

$fb\_mvt = on$
$at\_end = 1$
$can\_receive = no$

$fb\_mvt' = off$
$in\_between' = in\_between$
$at\_front' = at\_front$
$at\_end' = at\_end$

fb_to_table

$\Delta feed\_belt$
$\Delta table$

$fb\_mvt = on$
$at\_end = 1$
$can\_receive = yes$

$at\_end' = 0$
$in\_between' = in\_between$
$at\_front' = at\_front$
$fb\_mvt' = fb\_mvt$

$t\_loaded' = yes$
$t\_position' = t\_position$
$t\_mvt' = t\_mvt$

3.2 Petri Net Part of the Specification

Figure 2 shows the Petri net that specifies the control flow, i.e., the order in which the various Z operations can be executed. To enhance the readability of the Petri net, we introduce the following two layout conventions. (1) To avoid edge crossing we use *logical nodes* (the gray ones) to serve as connectors between identically named nodes. (2) Each independent process is drawn separately, and synchronization happens by logical nodes (there is place as well as transition synchronization).

Moreover, to avoid unnecessary restrictions of the concurrency degree, we use test arcs (black dots instead of the arrow head) to model side conditions of transitions (i.e., places that are incoming as well as outgoing for a given transition). Under the interleaving semantics, test arcs can be simulated by two opposite arcs between the transition and its side condition.

Following these rules, the Petri net exhibits a strong separation of controller and environment model into different parts. The controller part generally consists of a finite and static set of communicating processes, one for each physical component. The environment part is composed of small reusable net components: the producer/consumer processes of the work flow, and the devices of the controlled plant (as far as they are necessary on the net level).

Each physical device is basically characterized by its finite set of discrete states (e.g., the sensor at_front may recognize a plate or not, the feed belt may be switched on or off), whereby a discrete state may represent an equivalence class of a possibly

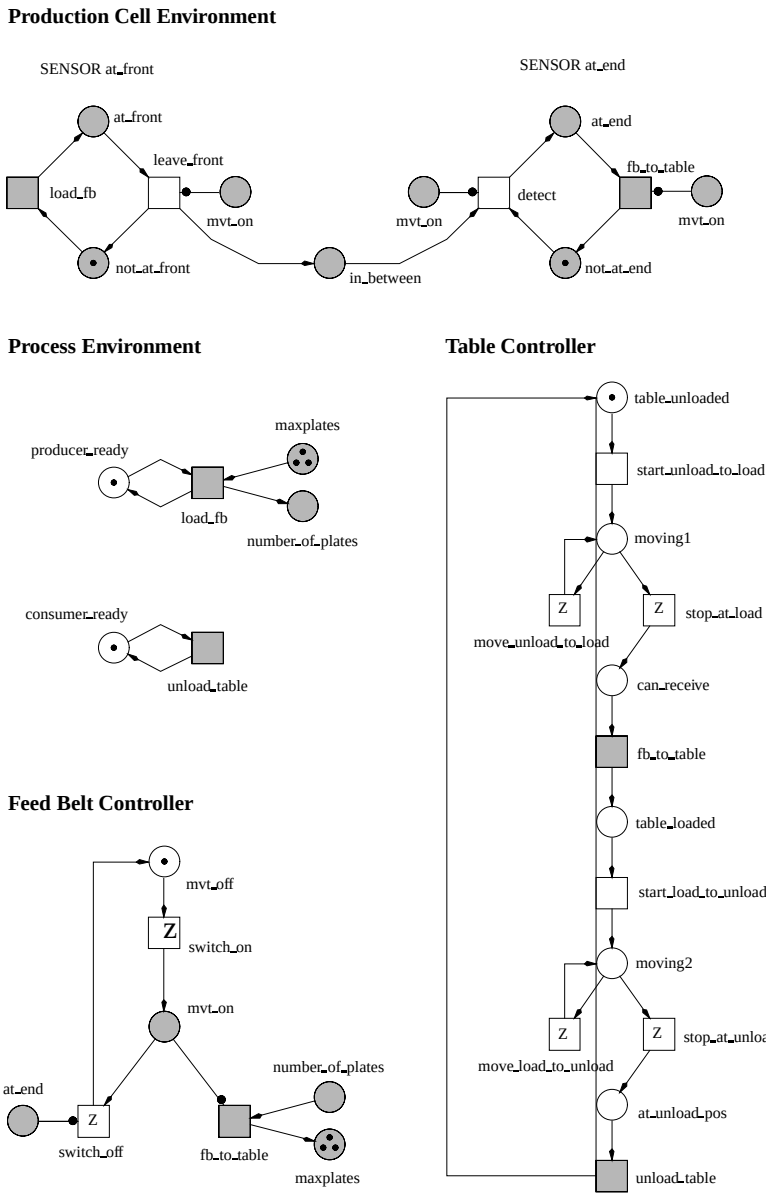

Fig. 2. Petri net for production cell

infinite set of states (e.g. $in\_between$ summarizes all feed belt states where some plate is located between the two sensors). Obviously, each device must be in one and only one state at any time. In terms of Petri net theory, the states of a device form a place invariant establishing a consistency condition of the system model, see Section 3.3.

To model the assumption of intelligent environment behavior (the producer places a new plate on the belt only if there is room for it), we introduce a place for the state variable $number\_of\_plates$, and a co-place $maxplates$ for its maximal value. In this way, we get a generic system specification - the given feed belt capacity is adapted by the parameter $maxplates$.

Note that there are six Z-labeled transitions, among them all conflicting transitions of the table controller. The transition fb_to_table, engaged in a conflict within the feed belt controller, is not a Z-labeled one, because the transition is enabled if and only if the precondition of its Z operation holds. In contrast, the operations $switch\_off$ and $switch\_on$ have more detailed preconditions than perceivable in the Petri net structure. Therefore, they are labeled with Z.

For comparison: In [HDS99], a hierarchical Petri net model of the complete production cell has been published, comprising altogether about 200 places and 200 transitions structured into 65 pages. The feed belt–table subsystem needs 46 places and 34 transitions. The Petri net of Figure 2 consists of only 17 places and 13 transitions, which makes a reduction factor of about 2.5.

3.3 Analysis of the Petri Net

Following our approach, the wide variety of available Petri net analysis techniques and tools becomes applicable for computer-aided model analysis. We briefly summarize the results of analyzing the Petri net of Figure 2, using our current Petri net tool box. The following tools have been applied: PED – a hierarchical Petri net EDitor [Tie97] for design, PEDVisor [Men97] for token flow animation, the Integrated Net Analysis tool INA [SR97] for analysis of the consistency conditions, and PEP [BG96] for analysis of the concurrency degree.

After being satisfied with the behavior exhibited by the animation of the Petri net, we perform general analyses. As a general consistency condition for our model, we show that the underlying Petri net is well-formed (which combines boundedness and liveness).

Boundedness. A place invariant x is a set of places, for which the token conservation equation

$$\sum_{p \in P} x(p) \cdot m_0(p) = \sum_{p \in P} x(p) \cdot m(p)$$

holds for all reachable markings m. Our net is covered by the following 8 semi-positive place invariants (where $x(p)$ always equals 1 for the mentioned places, and 0 otherwise):

$inv1 : (at\_front, not\_at\_front)$
$inv2 : (at\_end, not\_at\_end)$
$inv3 : (mvt\_on, mvt\_off)$

$inv4 : (number\_of\_plates, maxplates)$
$inv5 : (at\_front, at\_end, maxplates, in\_between)$
$inv6 : (table\_unloaded, moving1, can\_receive, table\_loaded, moving2, at\_unload\_pos)$
$inv7 : (consumer\_ready)$
$inv8 : (producer\_ready)$

Therefore it is bounded. More concretely: the token sum of the place invariants 1–3, 6–8 equals 1. Therefore, the corresponding places are 1-bounded. The token sum of the place invariants 4 and 5, on the other hand, equals $maxplates$. Therefore, the corresponding places can hold at most $maxplates$ tokens. Hence, we are able to conclude the k-boundedness of the net (with $k = maxplates$). Moreover, by combining the place invariants 4 and 5 we are able to conclude that for all reachable markings holds: $at\_front + in\_between + at\_end = number\_of\_plates$, which reflects an obvious consistency condition of the feed belt model, see Sect. 3.1, invariant of the schema $feed\_belt$.

Liveness. The net structure is extended simple, and the structural property called deadlock-trap property [Sta90] holds. Therefore, we can conclude without construction of the total system space that the pure net is live (which includes deadlock freedom). This is a necessary condition for the liveness of the Z-Petri net.

Concurrency degree. Figure 3 shows the basic concurrent behavior (after the initialization phase) of a production cell with a feed belt capacity of one. It has been derived from the so-called finite prefix of branching processes (produced by PEP), to highlight (and check) the essential behavior of the designed production cell. The derived net demonstrates the behavior of the Petri net shown in Figure 2 under the partial order semantics by showing two concurrent cycles of atomic actions (transitions, Z operations), synchronized by one common operation (fb_to_table). In other words, the action sequence $load\_fb - leave\_front - detect$ to equip the feed belt occurs concurrently with the table's motions from $load\_position$ to $unload\_position$ and back.

The three conflicts (non-deterministic behavior) in the concurrent behavior description are resolved by the Z operations' preconditions, because these exclude each other. Taking those into account, the feed belt is only switched off if the table does not work fast enough. Consequently, the feed belt will still be running while it is empty (because the producer is too slow). Therefore, the net is not reversible: loading the feed belt, and moving the table to the load position while the feed belt is switched off may happen only once at the beginning. This kind of behavior (which may be deemed to be undesirable) hardly becomes obvious by merely inspecting the specification.

If $maxplates = 1$, there only exists concurrency between operations on different subsystems that are independent of each other, i.e., work on disjoint state components. For $maxplates > 1$, however, we can identify the following concurrent operations[3]:

[3] Up to now, the theory of (a finite prefix of) branching processes is restricted to 1-bounded Petri nets. Therefore, the identification of the additional concurrent operations was not tool-supported. However, the generalization to bounded Petri nets is an emerging research area in Petri net theory.

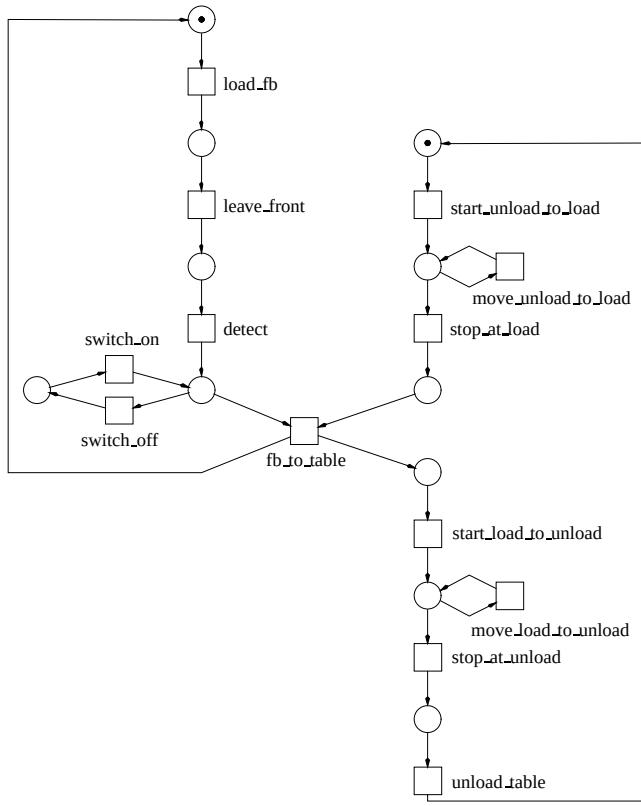

Fig. 3. Concurrent behavior of Petri net of Fig. 2 for $maxplates = 1$

- The operation $load\_fb$ may happen concurrently with $detect$, $switch\_on$, $switch\_off$, and fb_to_table.
- The operation $leave\_front$ may happen concurrently with $detect$, and fb_to_table.

3.4 Checking Coherence of the Two Specifications

As described in Section 2, we must show the compatibility of the two specifications.

It is easily verified that the initial marking of the Petri net is consistent with the Z specification. For example, there is a token at the place not_at_front, which is necessary because the initial condition of the feed belt requires $number\_of\_plates = 0$.

For the chains in the Petri net, it is also easy to see that the postconditions of all operations in a chain are compatible with the preconditions of their successors. As an example, consider the operations $switch\_off$ and $switch\_on$. The operation $switch\_off$ establishes the condition $fb\_mvt = off$, as required by $switch\_on$. Because $switch\_off$ establishes the condition $at\_end = 1$, the second precondition of $switch\_on$, viz. $number\_of\_plates > 0$ is also fulfilled. It follows that after a $switch\_off$ operation,

a $switch\_on$ operation is possible as soon as $can\_receive$ holds, i.e., as soon as the table is ready.

Finally, we must demonstrate that the order in which concurrent operations are invoked is irrelevant. Considering for example the operations $load\_fb$ and $detect$, we first identify the set of states where both preconditions hold. For these states, we have $0 < number\_of\_plates < maxplates$, $at\_front = at\_end = 0$, $fb\_mvt = on$, and $in\_between > 0$. Executing the operations $load\_fb$ and $detect$ yields a state where $number\_of\_plates$ is increased by one, $in\_between$ is decreased by one, $at\_front = at\_end = 1$, and $fb\_mvt = on$, independently of the order in which the operations are invoked. For the other concurrent operations, the reasoning proceeds in the same way.

3.5 Application-Dependent Validation of the Model

Lewerentz and Lindner [LL95] enumerate several safety requirements for the production cell control software. The requirements concerning the feed belt and the table are:

1. The blanks have sufficient distance so that they can be distinguished.
2. The table does not move beyond its extreme points.
3. Blanks are not dropped off the feed belt when the table is not ready. The feed belt is stopped before this can happen.

Requirement 1 is reflected in our specification by the precondition of the operation $load\_fb$: the state variable at_front must be zero, which means that the sensor reports that no plate is present at the front of the feed belt. This condition suffices to fulfill requirement 1. Once the operation $load\_fb$ is implemented, it must be demonstrated that the implementation indeed faithfully reflects the Z specification.

Requirement 2 is taken care of in the postconditions of the operations $stop\_at\_load$ and $stop\_at\_unload$. When the table has reached one of the extreme positions, the position engines are switched off, and the operations $move\_unload\_to\_load$ or $move\_load\_to\_unload$ respectively, are no longer possible. Again, every implemented system being correct with respect to the Z specification fullfills safety requirement 2.

To show that requirement 3 is fulfilled, we must show that if $at\_end = 1 \wedge fb\_mvt = on \wedge can\_receive = no$ then $fb\_mvt = off$ must hold within a certain time bound that is small enough to prevent the plate from being dropped in an unsafe area. If $maxplates = 1$, we can show that $fb\_mvt = off$ holds in the next state after $at\_end = 1 \wedge fb\_mvt = on \wedge can\_receive = no$ holds, because the only operation whose precondition is fulfilled is $switch\_off$. If $maxplates > 1$, however, we can only guarantee under the interleaving semantics that $fb\_mvt = off$ holds after at most $2(maxplates - 1)$ operations other than $switch\_off$ have been executed (these operations are $load\_fb$ and $leave\_front$). An exact proof that the feed belt is switched off fast enough requires a quantitative analysis using time-dependent Petri nets or a partial order semantics. Using transitions whose firing is restricted by time intervals, we would be able to formulate and check the time conditions under which $switch\_off$ is always faster than $load\_fb$ and $leave\_front$, respectively.

4 Conclusions

Nowadays, it is well recognized that combining data-oriented and behavioral formalisms is an adequate approach to specify embedded systems[4], and in particular safety-critical systems. Z has been combined with a number of other formalisms. We contrast our approach with two such combinations that have been specifically designed for specifying safety-critical embedded systems.

The combination of Z and real-time CSP defined by Heisel and Sühl [HS96] leads to very abstract and concise specifications, but tool support for validating specifications is limited and it is in general impossible to animate such specifications.

The language $\mu\mathcal{SZ}$ developed in the German ESPRESS project [BDG+96] is a combination of the Statemate languages [HLN+90] (namely statecharts and activity charts) and Z. Its advantage is that many engineers are familiar with finite state machines and hence may find $\mu\mathcal{SZ}$ specifications easily accessible.

The combination of Z and Petri nets, however, is superior to both afore-mentioned combinations as far as the means for animation and analysis are concerned. Animation tools provide an executable model of the system that allows customers to get an impression of how the system will behave. Furthermore, a variety of analysis tools (which are available free of charge) provide richer validation facilities than they are available for other formalisms. Checking consistency of the two parts of the specification further enhances confidence in the model.

The reader may see some similarities between our approach and CPN [Jen92] (a quasi-standard of coloured Petri nets), which combines Petri nets with inscriptions written in (a version of) the functional programming language ML. Because our primary objective is formal system specification, however, we prefer to use a pure specification language instead of a programming language in combination with Petri nets.

Our case study has shown that, because of the combination with Z, the Petri net model becomes quite concise and well comprehensible. The reduction of the number of nodes is considerable. Hence, our combination of Z and Petri nets is a promising approach to model safety-critical systems and validate these models.

To make our combined language acceptable to a wider audience, it is necessary to provide methodological support for its application. In the future, we intend to develop methods for

- Setting up combined specifications.
  Here, we need to develop heuristics for the order in which the two parts of a specification are developed, how to separate the software controller from its environment, etc.
- Validating combined specifications.
  A relation to classical safety analysis techniques would be desirable.
- Deriving implementations from combined specifications.

[4] This year, an international workshop "Integrated Formal Methods 1999–A Workshop on Combining State-Based and Behavioural Formalisms" on this specific topic takes place.

The combined language, an underlying methodology, and related tool support is likely to lead to a powerful approach for tackling the problem of system and especially software safety.

References

[BDG+96] R. Büssow, H. Dörr, R. Geisler, W. Grieskamp, and M. Klar. μSZ – ein Ansatz zur systematischen Verbindung von Z und Statecharts. Technical Report TR 96-32, Technische Universität Berlin, 1996.

[BG96] E. Best and B. Grahlmann. PEP–more than a Petri net tool. In *Proceedings TACAS'96*, LNCS 1055, pages 397–401. Springer-Verlag, 1996.

[BGHH98] R. Büssow, W. Grieskamp, W. Heicking, and S. Herrmann. An open environment for the integration of heterogeneous modelling techniques and tools. In *Current Trends in Applied Formal Methods*. Springer-Verlag, 1998. to appear.

[HDS99] M. Heiner, P. Deussen, and J. Spranger. A case study in developing control software of manufacturing systems with hierarchical Petri nets. *Int. Journal of Advanced Manufacturing Technology*, 15:139–152, 1999.

[HLN+90] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. rakhtenbrot. Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16 No. 4, April 1990.

[HS96] M. Heisel and C. Sühl. Formal specification of safety-critical software with Z and real-time CSP. In E. Schoitsch, editor, *Proceedings 15th International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, pages 31–45. Springer-Verlag London, 1996.

[Jen92] K. Jensen. *Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use, Vol. 1*. Springer-Verlag, 1992.

[KSW96] Kolyang, T. Santen, and B. Wolff. A structure preserving encoding of Z in Isabelle/HOL. In J. von Wright, J. Grundy, and J. Harrison, editors, *Theorem Proving in Higher-Order Logics*, LNCS 1125, pages 283–298. Springer-Verlag, 1996.

[LL95] C. Lewerentz and T. Lindner, editors. *Formal Development of Reactive Systems*. LNCS 891. Springer-Verlag, 1995.

[LS96] N. Lévy and J. Souquières. A "Coming and Going" Approach to Specification Construction: a Scenario. In W. Schäfer, J. Kramer, and A. Wolf, editors, *Proc. 8th Int. Workshop on Software Specification and Design*, pages 115–118. IEEE Computer Society Press, 1996.

[Men97] T. Menzel. Entwurf und Prototypimplementierung eines Petri-Netz-Framework. Technical report, BTU Cottbus, Institut für Informatik, 1997.

[Saa97] M. Saaltink. The Z/EVES system. In J. Bowen, M. Hinchey, and D. Till, editors, *ZUM'97: The Z Formal Specification Notation*, LNCS 1212, pages 72–88. Springer-Verlag, 1997.

[Spi92a] J. M. Spivey. The fuzz manual. Computing Science Consultancy, Oxford, 1992.

[Spi92b] J. M. Spivey. *The Z Notation – A Reference Manual*. Prentice Hall, 2nd edition, 1992.

[SR97] P. H. Starke and S. Roch. INA–Integrated Net Analyser version 1.7. Technical report, Humboldt-Universität Berlin, 1997.

[Sta90] P. H. Starke. *Analyse von Petri-Netz-Modellen*. Teubner, 1990.

[Tie97] R. Tiedemann. PED – Hierarchischer Petri-Netz-Editor. Technical report, BTU Cottbus, Institut für Informatik, 1997.