A Z specification of the boiler*

Pascal BERNARD
bernard@irin.univ-nantes.fr

January 5, 1996

Abstract

A method is presented for specifying a driver from a given behaviour of the system to be controlled and from the perception the driver can have from the system. The advantages and drawbacks of using Z are presented along the development of the specification of the boiler.

The informal specification was provided by J.R. Abrial ([Abr94]). This specification is an adapted version of one provided by LtCol. J.C. Bauer for the Institute for Risk Research (University of Waterloo, Canada).

Introduction

For this specification of the boiler control system, the Z notation and method are chosen. The specification has been performed with two goals in mind:

to understand what the problem really was This has been achieved by reorganizing the requirements and by producing a mathematical models of the boiler behaviour and the driver behaviour.

to obtain the program of the driver More precisely it is a class of programs. Implementation details are not necessarily relevant at that stage and the specification leaves open the choice for different actions of the driver when several of them are equally adequate.

There is no reason why a specification could be suitable for both. But a good understanding of the problem is obviously a prerequisite to the design of the driver. This understanding may be drawn from a rephrasing of the informal specification into a mathematical model of the whole system. It is possible to isolate during this process a part of the specification which correspond to the specification of the controller (or *driver*). This approach is interesting in the sense that:

*This work has been presented at the Dagstuhl Seminar on Methods for Semantics and Specification

- it is possible to check that no information is lost in the specification process: every statement in the informal specification has its counterpart in the formal specification,
- the specification of the whole system is required for checking adequacy. This is an important issue: when several engineers cooperate for some task they must check to some extend the work of each other to make sure that their contribution is correct,
- the specification of the driver *shares* definitions with the specification of the physical system to be under control. As a consequence, if the modelling of the physical system is to be altered, the specification of the driver will be automatically updated,
- the specification is homogeneous, only Z is used.

Z is a notation allowing to build mathematical theories based on set theory [BN92]. It provide operations for composing large specifications from parts of it. The parts can be refered by their names. This permits to have short statements in the construction of the specification whatever is its size. Quite a few case studies have been made using Z. Therefore the notation is a good candidate fot specifying the boiler driver. A method has been developed for specifying softwares using this notation [Spi92]. This method consists in:

- designing a set theoretical modelling of the possible states of the system (this is the invariant),
- specifying, for each operation of the system, the constraint which holds between the states before and after the application of the operation,
- proving that whenever an operation may be applied, there exists a state which will, together with the state of the system when the operation is applied, satisfy the constraint associated to this operation,
- refining the specification in order to get a specification from which a computer program can be produced.

This method is adopted (and adapted) to the specification of the boiler controller. The boiler system is not a "real time" system. In this case there is no problem such as concurrent activities which can be interrupted asynchronously. In that respect, the method is suitable. But the method relies on the possibility of defining a strong invariant property. It will appear that it is not the case for the boiler problem. Moreover, although it provides a framework to prove them, the method does not deal with the statement of liveness properties. In the specifiction of the boiler, the properties over the dynamics of the system revealed to be essential. This is not the case for most of the specification examples using Z so far.

1 Specification of a driver with Z

The specification is given by the behaviour of the system to be controlled and by a constraint over the states it can take when it is controlled. Thus, the specification of the driver is given from outside : its environment. This is why the specification has to describe two systems: the one being controlled and the controlling system. A priori, there is no possible modelling of the states of the controlling system. But we know at least its interface with the controlled system. This modelling is to be completed with the information which is necessary in order to write the (weakest) constraint that entails the required behaviour of the global system. Before starting with the specification of the boiler, a general pattern for the specification of a driver will be presented. This is the adapted method. It consists in answering the following questions:

- What are the operations to be introduced in the description of the systems ?
- What sort of information should be extracted from the modelling of the states of the machines ?
- What are the proof obligations ?

1.1 The operations

The history of the global system is a continous stream of cycles which may be divided into four periods:

- the controlled system evolves on its own, with respect to the commands received from the controlling system,
- measures are taken and are transmitted to the controlling system
- the controlling system computes the commands for the next cycle,
- the commands are transmitted to the controlled system

Here we make the assumption that all the commands are issued *after* the measures are taken. The informal specification of the boiler suggests to simplify matters: transmission time can be neglected, messages (or measures and commands) are received (emitted) simultaneously. It is reasonnable to think that the time for computation can also be neglected. We make the further assumption: time for measuring can be neglected. We are aware that this assumption is debatable: measuring a speed or a flow often takes time. As it does not always take time, as there are other interesting problems to be solved, it is fair to put this hypothesis. This simplifies the cycle which now consists in two periods:

- the first period where the controlled system is left on its own,
- the second period which takes no time where the control takes place.

It should be noted that no command is issued during the first period.

For each of these periods and for each system, we propose to specify an operation which represents the behavior of the systems for the corresponding periods. The "synchronization" of the systems then takes place in the definition of the global system. Each period is associated to an operation which is the conjonction (in the sense of Z) of the operations of the combined systems for the corresponding period.

1.2 The invariants

Three problems are encountered in the definition of the invariant:

- the set of possible states cannot be the same after and before the operations are performed,
- the set of possible states depends on the phase the system is in. There are at least three different phases: a first phase where the driver tries to seize the system to be controlled, the second phase is where the control holds, and the third and terminal phase is when the system has got out of the hands of the driver,
- the invariant constrains the states before and after an operation is performed. It does not constrains the states while the operation is performed.

Our solution consist in:

- there are two sets of states, one included in the other. The first set is the set of states produced by the driver. The second set is the set of states which are reached after the controlled system has been left on its own. As this system tends to diverge, the second set is larger than the first (when the system is back on the road). This may be modelled by taking as invariant the larger set and by stating a precondition for the operations of the first period. The verification that the specification is sound will be made through the proof obligations.
- the phase into which the system is in should be inferred from its state. It means that this information should be available from the modelling of the states.
- for the specification of the boiler, this will not cause any trouble (because of the definition of $M1$ and $M2$). More generally, two sets should be considered: the set of possible states after the transition and the set of states during the transition.

1.3 Proof obligations

The usual proof obligations which verify that the operations are applicable are part of the proofs of the adapted method. It must be proved that the operation of the second period always provides a state where the first operation applies.

Other proof obligations will be added. They concern the livness properties of the system. For instance, it must be proved that when no problem arises, the initialisation process terminates. These proofs may not be possible at the abstract level and can be postponed after the specification has been refined.

2 Formalization of the boiler specification

The description of the system is divided into two parts:

- the static part. This part gives a mathematical model of the system: its properties and the values taken by these properties. This is called the invariant,
- the dynamic part which gives the possible transitions of the system.

2.1 The invariant

First are given the modelling of the parts. Then the parts are assembled and the constraints over the values of their properties are asserted to constitute the invariant.

2.1.1 Specification of the parts

Z allows to specify parts of the whole system individually before putting them together. We start with the specification of the units which are relevant for the driver.

The relevant caracteristics of the boiler are the quantity of water it contains, the quantity of steam exiting and the state of the valve. The type *bool* is introduced for the modelling

$$bool ::= True \mid False$$

as well as two constants (maximal capacity and maximal quantity of steam):

$$C, W : \mathbb{N}$$

To simplify matters, quantities and measures will be taken as naturel numbers. Desired accuracy can be obtained by choosing a very small unit of measure.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{BOILER}} \\
q, v, p : \mathbb{N} \\
valve : bool \\
\hline
q \geq 0 \land q \leq C \\
v \geq 0 \land v \leq W
\end{array}
$$

The relevant caracteristics of water level measurement device are its state (whether it is out of order or not) and the measured level. For the boiler, it was not considered that it could fail (eg. by leaking). The reason is that it it not the responsability of the driver to detect and to sort out such a situation.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{QMETER}} \\
OOQM : bool \\
qmsrd : \mathbb{N}
\end{array}
$$

To be more precise, a limited range could be set for the measure. But it did not occure that this limit would be of any interest as far as it is large enough (range including $0 \ldots C$).

As the pumps are all the same, they may be specified together. Each pump can be out of order. They can be switched on or off. They are pourring a given quantity of water or they are not pourring any water at all.

$$PUMP ::= pump1 \mid pump2 \mid pump3 \mid pump4$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{PUMPS}} \\
OOPump : \mathbb{P}(PUMP) \\
PumpOn : \mathbb{P}(PUMP) \\
PumPour : \mathbb{P}(PUMP) \\
\hline
\forall i : PUMP \bullet PumPour\ i \Rightarrow PumpOn\ i
\end{array}
$$

A pump pourring water is necessarily switched on but it may be on while it does not pour water. The pump control devices are also specified together:

$$
\begin{array}{|l}
\hline
\textit{PUMPMETER} \\
\hline
OOPM : \mathbb{P}(PUMP) \\
PumpActive : \mathbb{P}(PUMP) \\
\hline
\end{array}
$$

The steam measurement device is like the water measurement device. The driver will have to check that it is working properly. Two properties of this device are considered: its state and the measure it gives.

$$
\begin{array}{|l}
\hline
\textit{STEAMETER} \\
\hline
OOSM : bool \\
vmsrd : \mathbb{N} \\
\hline
\end{array}
$$

The operator desk is under the responsability of the physical environment. As for the boiler, we do not consider its possible failure which should be detected and managed outside the driver.

The messsage transmission system can be decomposed into reception and emission. As there is no information on the message passing procedure, we will have an abstract definition of it.

The type INMESS contains the possible messages to the driver.

$$
\begin{array}{l}
INMESS ::= Level\langle\langle \mathbb{N} \rangle\rangle \mid Steam\langle\langle \mathbb{N} \rangle\rangle \mid \\
\quad PumpState\langle\langle PUMP \times bool \rangle\rangle \mid \\
\quad PumpCtrState\langle\langle PUMP \times bool \rangle\rangle \mid \\
\quad Stop\langle\langle \mathbb{N} \rangle\rangle \mid \\
\quad SteamBoilerWaiting \mid PhysicalUnitsReady \mid LevelRepaired \mid SteamRepaired \mid \\
\quad LevelFailureAck \mid SteamFailureAck \mid \\
\quad PumpRepaired\langle\langle PUMP \rangle\rangle \mid \\
\quad PumpCtrRepaired\langle\langle PUMP \rangle\rangle \mid \\
\quad PumpFailureAck\langle\langle PUMP \rangle\rangle \mid \\
\quad PumpCtrFailureAck\langle\langle PUMP \rangle\rangle
\end{array}
$$

Only the Stop message can be duplicated. This is why a count is associated with it. The reception part of the transmission can be modelled by a set of elements of type INMESS. A type OUTMESS is defined in the same way.

$$
\begin{array}{|l}
\hline
\textit{TRANSMISSION} \\
\hline
OOTM : bool \\
inmess : \mathbb{P} INMESS \\
outmess : \mathbb{P} OUTMESS \\
\hline
\end{array}
$$

At this stage, the physical system could be put together. But this would lead to a considerable amount of cases to be considered. Most of them are unecessary if we make the assumption that the properties given in the informal specification are sufficient to be able to detect failures. The invariant depends on the dynamics of the system in order to make detection possible. For instance, we have to set the constraint that if a pump has been asked to start two cycles ago it should pour water. Otherwise there is failure of the pump, the specification telling that this is the only plausible cause (eg it could be thought of a failure in transmitting the order of switching on the pump).

This is why it is necessary to have as a property of the system some information about its history. Moreover, it also depends on the history of the driver (see example above). As this information will have to be available for the implementation of the driver, we first give a modelling of the driver states. Then, the modelling of the whole system will be built by putting together the different parts and by adding constraints among the variables of the different parts.

In order to describe the history, stages of this history are introduced. The invariant will be split later in smaller invariants, each of which corresponding to a stage of the history ([BL95]). This stages do not exactly correspond to the "modes". Initialization is split into three stages: waiting for the message STEAM-BOILER-WAITING, adjusting the water level and waiting for the message PHYSICAL-UNIT-READY. All the stages are not necessarily part of the history of the system: the system may skip to the final stage if a failure of the water level measurement unit is detected (in the text of the informal specification:"failure of the water level detection unit"). Here are the definitions of stages and modes:

$$STAGE ::= init1 \mid init2 \mid init3 \mid control \mid exit$$

$$MODE ::= Initialization \mid Normal \mid Degraded \mid Rescue \mid EmmergencyStop$$

It is necessary to keep track of the failures of units which have been detected. A type is defined which elements name the units which may fail while the system proceeds:

$$UNIT := qmeter \mid pump\langle\langle PUMP\rangle\rangle \mid pumpmeter\langle\langle PUMP\rangle\rangle \mid steameter$$

The driver is modelled in two parts for a reason which will be clear at the definition of the transitions.

$$
\begin{array}{|l}
\hline
\textit{DRIVERDEFS} \\
\hline
detections : \mathbb{P} UNIT \\
driverun : bool \\
mode : MODE \\
\hline
\end{array}
$$

The property *driverun* will be true only at the second period.

$$
\begin{array}{|l}
\hline
\textit{DRIVER} \\
\hline
stage : stage \\
expected, risk, believed : \mathbb{P} BOILER \\
dfailures : \mathbb{P} UNIT \\
valveon : bool \\
log : \mathbb{P} OUTMESS \\
unexpected : \mathbb{P} INMESS \\
\hline
\end{array}
$$

In this schema, *expected* is the set of states in which the current state of the boiler was expected to be (at the previous cycle). *risk* is the set of states the current state of the boiler might be in if some failures occured. *believed* is the set of states in which the driver assumed to be at the previous cycle. *dfailures* is the set of units which were faulty at the previous cycle.

2.1.2 Invariant of the whole system

The squeleton of the modelling of the whole system is defined as:

$$
\begin{array}{l}
PHYSICAL \mathrel{\widehat=} BOILER \land QMETER \land PUMPS \land PUMPMETER \land \\
\quad STEAMETER \\
WS0 \mathrel{\widehat=} PHYSICAL \land TRANSMISSION \land DRIVERDEFS \land DRIVER
\end{array}
$$

The stages and modes are related to each other. Properties that should hold at specific stages or mode are also listed. The last property relates the throughput p with the number of pumps pourring and their nominal capacity P.

$$P : \mathbb{N}$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{WS}} \\
\hline
WS0 \\
\hline
stage = init1 \Rightarrow mode = Initialization \\
stage = init2 \Rightarrow mode = Initialization \\
stage = init3 \Rightarrow mode = Initialization \\
mode = Normal \Rightarrow stage = control \\
mode = Degraded \Rightarrow stage = control \\
mode = Rescue \Rightarrow stage = control \\
mode = EmergencyStop \Leftrightarrow stage = exit \\
(stage = control \wedge dfailures \cup detections = \emptyset) \Rightarrow mode = normal \\
(stage = control \wedge qmeter \in (dfailures \cup detections)) \Rightarrow mode = rescue \\
(stage = control \\
\ \wedge qmeter \notin (dfailures \cup detections) \\
\ \wedge dfailures \cup detections \neq \emptyset) \\
\quad \Rightarrow mode = degraded \\
mode = Initialization \Rightarrow v = 0 \\
stage = control \Rightarrow q \geq M1 \wedge q \leq M2 \\
valve = True \Rightarrow stage = init2 \\
p = \#PumPour * P \\
\hline
\end{array}
$$

2.2 The transitions

The transitions are operating from an initial state which satisfies:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{INIT}} \\
\hline
WS \\
\hline
stage = init1 \\
dfailures = \emptyset \\
detections = \emptyset \\
believed = \{b : BOILER \mid v = 0\} \\
expected = believed \\
risk = believed \\
valve = False \\
valveon = False \\
driverun = True \\
\forall i : PUMP \bullet \neg PumpOn\ i \\
outmess = \emptyset \\
log = \emptyset \\
unexpected = \{PhysicalUnitsReady, LevelRepaired, SteamRepaired\} \cup \\
\quad \{LevelFailureAck, SteamFailureAck\} \cup \\
\quad \{i : PUMP \bullet PumpRepaired\ i\} \cup \\
\quad \{i : PUMP \bullet PumpCtrRepaired\ i\} \cup \\
\quad \{i : PUMP \bullet PumpFailureAck\ i\} \cup \\
\quad \{i : PUMP \bullet PumpCtrFailureAck\ i\} \\
\hline
\end{array}
$$

The driver is started while the boiler is still.

Two transitions are to be defined. We call the first transition BLACKOUT because this is the transition of the system while the physical system is left on its own. The second transition is called FLASH because it does not last any time. The definition of these two transitions are decomposed into two parts: the point of view of the physical system and the point of view of the rest of the system (transmission and driver).

In order to define the BLACKOUT transition from the point of view of the physical system, we first give a model of its evolution.

As there is no information about the pressure in the boiler, it is assumed that the quantity of steam given in litres is in fact the quantity of water which correspond to the steam. Hence, if the flow of steam amounts to v during a period of dt, the quantity of water which escaped from the boiler is $v * dt$. Let v and v' be the quantity of steam exiting the boiler before and after the transition. As the variation of steam quantity is bound by U_1 and U_2, the quantity of steam exiting the boiler during the transaction is within the envelope pictured on figure 1. The quantity of water exiting the boiler during the transaction is in an interval. The maximum value is obtained by

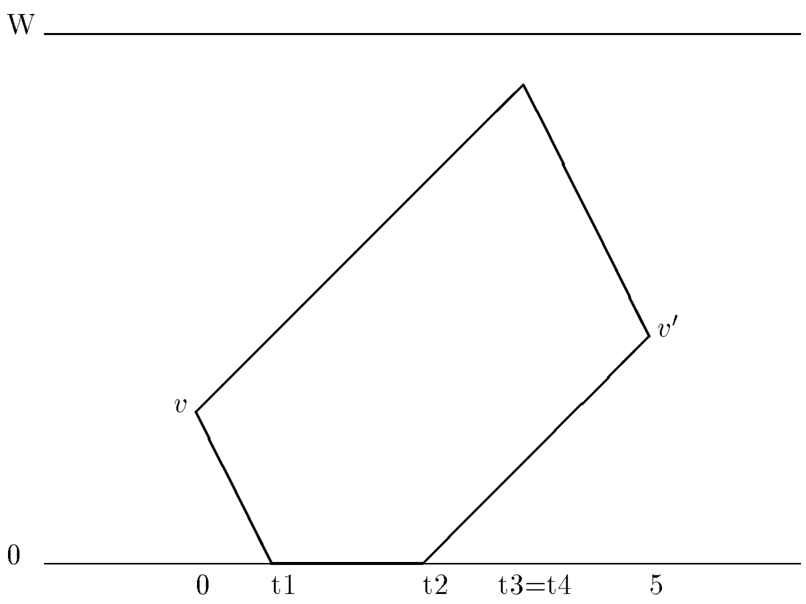

Figure 1: The envelope

taking the upper limit of the envelope for the values of steam quantity, the minimum value by taking the lower limit. The calculation is elementary. Two functions are introduced in the specification for these limits, without presenting the detail of the calculation. The other constants are introduced at the same time. The constant t represents the 5 seconds in the unit of time chosen for sufficient accuracy.

$$
\begin{array}{|l}
t : \mathbb{N} \\
P, W, U_1, U_2, M_1, M_2 : \mathbb{N} \\
upper : \mathbb{N} \times \mathbb{N} \nrightarrow \mathbb{N} \\
lower : \mathbb{N} \times \mathbb{N} \nrightarrow \mathbb{N} \\
\hline
\mathsf{dom}\ lower = \mathsf{dom}\ upper \\
\mathsf{dom}\ upper = \{x, y : \mathbb{N} \mid x \in 0 \ldots W \land y \in 0 \ldots W \\
\quad \land\ y \geq x - t * U_2 \land y \leq x + t * U_1 \bullet (x, y)\}
\end{array}
$$

The domain of the function is the set of pairs (v, v') such that v' can be reached from v. The quantity q' of water after the transaction is obtained by adding the throughput of the pumps and retrieving the steam quantity from the quantity q of water before the transaction. But this throughput may vary. The pumps which are starting or stopping pourring during this period are failing. The pumps which are stopping pourring are switched off. This is necessary to make the detection possible. The definition of BOphysical is divided in three parts. The first part gives the possible evolutions of the boiler itself.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\_\ BOboil\ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
\Delta WS \\
\hline
mode = Initialisation \Rightarrow \\
\quad (q' < q + max(p,p') \wedge q' > q + min(p,p') \\
mode \neq Initialisation \Rightarrow \\
\quad v' \in (\operatorname{dom} upper)(\!|\ \{v\}\ |\!) \\
\quad \wedge q' < q - lower(v,v') + t * max(p,p') \\
\quad \wedge q' > q - upper(v,v') + t * min(p,p') \\
PumPour' = PumpOn \cap PumpOn'
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\_\ PHYSMESS\ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
\Delta WS \\
\hline
(OOTM' = FALSE \Leftrightarrow \\
\quad (\exists_1 n : \mathbb{N} \bullet Level\ n \in inmess') \\
\quad \wedge (\exists_1 n : \mathbb{N} \bullet Steam\ n \in inmess') \\
\quad \wedge (\forall i : PUMP \bullet PumpState(i, True) \in inmess' \Leftrightarrow \\
\qquad \neg PumpState(i, False) \in inmess') \\
\quad \wedge (\forall i : PUMP \bullet (\exists b : bool \bullet PumpCtrState(i, b) \in inmess')) \\
\quad \wedge inmess' \cap unexpected = \emptyset \\
\quad) \\
\wedge (OOQM' = False \Rightarrow Level\ q' \in inmess') \\
\wedge (OOSM' = False \Rightarrow Steam\ v' \in inmess') \\
\wedge (\forall i : PUMP \bullet PumpOn'\ i \Leftrightarrow PumpState(i, True) \in inmess') \\
\wedge (\forall i : PUMP \bullet PumpActive'\ i \Leftrightarrow PumpCtrState(i, True) \in inmess') \\
\wedge (\forall i : PUMP \bullet \neg OOPM'\ i \Rightarrow (PumPour'\ i \Leftrightarrow PumpActive'\ i)) \\
\wedge Valve \in outmess \Rightarrow ((valve = True \Rightarrow valve' = False) \\
\qquad \wedge (valve = False \Rightarrow valve' = True)) \\
\wedge (\forall p : PUMP \mid \neg OOPump'\ p \bullet OpenPump(p) \in outmess \Rightarrow PumpOn'\ p) \\
\wedge (\forall p : PUMP \mid \neg OOPump'\ p \bullet ClosePump(p) \in outmess \Rightarrow \neg PumpOn\ 'p)
\end{array}
$$

In $PHYSMESS$ are stated the constraints relating the properties of the physical devices with the contents of the messages. These constraints show that if we want to know if the messages are telling the truth, we have to know whether the devices are out of order. At this point it is necessary to make more precise what is meant by "out of order". We will say that a device is out of order from the time it fails to fulfil its duty up to when it is known to be repaired. Detections, which are to be operated at every cycle, serve the purpose of knowing the time when physical devices start to fail to fulfil their duty. Because detection takes place at each cycle, the following constraints will be verified:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\_ \; CLEARBREAK \; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
\Delta WS \\
\hline
(OOQM' = True \Leftrightarrow qmeter \in detections' \cup dfailures) \\
\wedge(OOSM' = True \Leftrightarrow steameter \in detections' \cup dfailures) \\
\wedge(\forall i : PUMP \bullet OOPump'\ i \Leftrightarrow pump\ i \in detections' \cup dfailures) \\
\wedge(\forall i : PUMP \bullet OOPM'\ i \Leftrightarrow pumpmeter\ i \in detections' \cup dfailures) \\
\forall i : PUMP \mid \neg OOPump'\ i \\
\quad \bullet PumpOn'\ i \Leftrightarrow \\
\quad\ ClosePump(i) \notin outmess \wedge (OpenPump(i) \in outmess \vee PumpOn\ i) \\
\forall i : PUMP \\
\quad \bullet PumPour'\ i \Leftrightarrow \\
\quad\ (\neg OOPM'\ i \wedge ClosePump(i) \notin outmess \wedge PumpOn\ i) \\
\quad\ \vee(OOPM'\ i \wedge PumpOn\ i \wedge PumpOn'\ i) \\
\hline
\end{array}
$$

$$BOphysical \widehat{=} BOboil \wedge PHYSMESS \wedge CLEARBREAK$$

The second transition, from the point of view of the physical system is:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\_ \; FLphysical \; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
\Delta WS \\
\Xi PHYSICAL \\
\hline
inmess' = inmess \\
\hline
\end{array}
$$

The pumps which were on will proceed or start to pour water, unless they are stopped when they do not fail to respond to the messages.

Let us call BOcomputer the schema constraining the behaviour of the driver. This will be composed with four schemas. The driver is idle during this period. The property *driverun* is set by the transitions to force them to be applied one after the other.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\_ \; BOcomp \; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
\Delta WS \\
\Xi DRIVER \\
\hline
driverun' = True \\
driverun = False \\
\hline
\end{array}
$$

The definition of detections while the driver runs is introduced through three schemas. It is expected that it would be more natural to specify the detection while the driver is not idle. But we felt that the specification was nicer this way. First, the detection of pump failure:

$$
\begin{array}{|l}
\hline
\textit{PUMPFAIL} \\
\Delta WS \\
\hline
(\forall i : PUMP \bullet pump\ i \in detections' \Leftrightarrow \\
\quad (pump\ i \notin dfailures \\
\quad \wedge((OpenPump(i) \in outmess \wedge PumpState(i, False) \in inmess') \\
\quad\quad \vee(ClosePump(i) \in outmess \wedge PumpState(i, True) \in inmess') \\
\quad\quad \vee(ClosePump(i) \notin outmess \\
\quad\quad\quad \wedge OpenPump(i) \notin outmess \\
\quad\quad\quad \wedge(\exists b, bb : bool \\
\quad\quad\quad\quad \mid PumpState(i, b) \in outmess \\
\quad\quad\quad\quad \wedge PumpState(i, bb) \in inmess \\
\quad\quad\quad\quad \bullet b \neq bb))))) \\
\hline
\end{array}
$$

The detection of a pump failure relies on PumpState messages. It means that PumpState messages report on the fact that the pumps are switched on or not (PumpOn). The failure is an unappropriate response to the commands they receive. It is not to provoke an erroneous message. As the observation of this unappropriate response allows to decide whether a pump fails, it can be deduced that the pump does not fail to pour water five seconds after it has been switched on (unless turned off in which case the failure is detected).

Next is the definition of pump control devices. Where the text of the informal specification says "the program knows from elsewhere", the interpretation is "the pump is on and is not out of order". But this could be changed without modifying the rest of the specification.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\_ \; PUMPCTRFAIL \; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
\Delta WS \\
\hline
(\forall i : PUMP \bullet pumpmeter(i) \notin detections' \Leftrightarrow \\
\;(pumpmeter(i) \notin dfailures \\
\;\wedge((ClosePump(i) \in outmess \\
\quad \wedge PumpCtrState(i, True) \in inmess' \\
\quad \wedge PumpState(i, False) \in inmess') \\
\;\; \vee(OpenPump(i) \in log \\
\quad \wedge PumpCtrState(i, False) \in inmess' \\
\quad \wedge PumpState(i, True) \in inmess' \\
\quad \wedge pump(i) \notin dfailures \cup detections') \\
\;\; \vee(PumpCtrState(i, False) \in inmess' \\
\quad \wedge PumpCtrState(i, True) \in inmess \\
\quad \wedge PumpState(i, True) \in inmess') \\
\;\; \vee(PumpCtrState(i, True) \in inmess' \\
\quad \wedge PumpCtrState(i, False) \in inmess \\
\quad \wedge(PumpState(i, False) \in inmess' \\
\qquad \vee(PumpState(i, True) \in inmess' \\
\qquad \; \wedge PumpState(i, False) \in inmess))) \\
\;))) \\
\hline
\end{array}
$$

The physical system described for the specification is an ideal system where the failures are clear enough to be detectable. There is no failure with spontaneous prompt recovery and no false measures which are plausible.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\_ \; METERSFAIL \; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
\Delta WS \\
\hline
steameter \in detections' \Leftrightarrow \\
\;(\forall n : \mathbb{N} \mid Steam\ n \in inmess' \bullet \\
\quad n \notin \{b : BOILER \mid b \in expected \bullet b.v\}) \\
qmeter \in detections' \Leftrightarrow \\
\;(\forall n : \mathbb{N} \mid Level\ n \in inmess' \bullet \\
\quad n \notin \{b : BOILER \mid b \in risk \bullet b.q\}) \\
\hline
\end{array}
$$

$$BOcomputer \widehat{=} BOcomp \wedge PUMPFAIL \wedge PUMPCTRFAIL \wedge METERSFAIL$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad BOcompletion \quad}} \\
\Xi WS \\
\hline
driverun = True \\
\hline
\end{array}
$$

$$BLACKOUT \mathrel{\widehat=} (BOphysical \land BOcomputer) \lor BOcompletion$$

The definition of the behaviour of the driver for the second period is much more complex. It will be constructed combining several schemas. First the easiest case:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad EXCEPTIONS \quad}} \\
WS \\
\hline
OOTM = True \\
\exists i, j : \mathbb{N} \bullet Stop(i) \in inmess \land (i \geq 3 \lor (Stop(j) \in plog.inmess \land i + j \geq 3)) \\
\hline
\end{array}
$$

If this schema is satisfied the answer is:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad EXIT \quad}} \\
\Delta WS \\
\hline
driverun = True \\
state' = exit \\
outmess' = Mode(EmergencyStop) \\
\hline
\end{array}
$$

The rest of the definition is split along the different stages and modes. This will be easier by introducing some functions.

The function *initdiv* returns the set of possible states which are reachable after the BLACKOUT transition from a given state of the boiler and from the number of pumps providing water if no failure occurs, the valve is closed and the mode is initialization.

$$
\begin{array}{|l}
initdiv : BOILER \rightarrow \mathbb{P} BOILER \\
\hline
initdiv = \{b, bb : BOILER \mid b.valve = False \land b.v = 0 \\
\qquad \land bb.q = b.q + b.p * t \\
\qquad \land bb.p = b.p \\
\qquad \land bb.v = 0 \\
\qquad \land bb.valve = False \\
\qquad \bullet (b \mapsto bb)\}
\end{array}
$$

The function *diverge* has the same meaning but for the stage *control*.

$$
\begin{array}{|l}
diverge : BOILER \nrightarrow \mathbb{P} BOILER \\
\hline
diverge = \{b, bb : BOILER \mid b.valve = False \\
\qquad \wedge bb.v \in max(0, b.v - U_2 * t) \ldots min(W, b.v + U_1 * t) \\
\qquad \wedge bb.q < b.q - lower(b.v, bb.v) + b.p * t \\
\qquad \wedge bb.q > b.q - upper(b.v, bb.v) + b.p * t \\
\qquad \wedge bb.p = b.p \\
\qquad \wedge bb.valve = False \\
\qquad \bullet (b \mapsto bb)\}
\end{array}
$$

If the risk of the pumps to fail is considered, the set of reachable states is obtained with *divrisk*.

$$
\begin{array}{|l}
divrisk : BOILER \nrightarrow \mathbb{P} BOILER \\
\hline
divrisk = \{b, bb : BOILER \mid b.valve = False \\
\qquad \wedge bb.v \in max(0, b.v - U_2 * t) \ldots min(W, b.v + U_1 * t) \\
\qquad \wedge bb.q < b.q - lower(b.v, bb.v) + bb.p * t \\
\qquad \wedge bb.q > b.q - upper(b.v, bb.v) + b.p * t \\
\qquad \wedge bb.p \in \{m : \mathbb{N} \mid m * P \leq b.p \bullet m * P\} \\
\qquad \wedge bb.valve = False \\
\qquad \bullet (b \mapsto bb)\}
\end{array}
$$

This allows to define $risk'$ and $expected'$ when we know exactly the current state of the boiler. More general functions are defined in case the driver has not a clear picture of what the current state is. The driver knows that the current state is in a set (*believed*).

$$
\begin{array}{|l}
Diverge : \mathbb{P} BOILER \nrightarrow \mathbb{P} BOILER \\
\hline
\forall p : \mathbb{P} BOILER \bullet Diverge(p) = \bigcup \{b : BOILER \mid b \in p \bullet diverge(b)\}
\end{array}
$$

Divrisk can be defined in the same way.

Most of the setting of *outmess*,*dfailures*,*expected*,*detections* and *log* is common to all the non exceptionnal cases.

$$
\begin{array}{l}
\textit{COMMON} \\
\hline
\Delta WS \\
\hline
driverun = True \land driverun' = False \\
Mode\ mode' \in outmess' \\
state' = exit \Rightarrow EmergencyStop \in outmess' \\
state \neq init2 \Rightarrow \{ProgramReady, Valve\} \cap outmess' = \emptyset \\
\forall p : PUMP \bullet ClosePump(p) \in outmess' \Rightarrow OpenPump(p) \notin outmess' \\
\forall p : PUMP \bullet OpenPump(p) \in outmess' \Rightarrow ClosePump(p) \notin outmess' \\
\forall p : PUMP \bullet PumpFailure(p) \in outmess' \Leftrightarrow pump\ p \in detections \\
\forall p : PUMP \bullet PumpCtrFailure(p) \in outmess' \Leftrightarrow pumpmeter\ p \in detections \\
LevelFailure \in outmess' \Leftrightarrow qmeter \in detections \\
SteamFailure \in outmess' \Leftrightarrow steameter \in detections \\
\forall p : PUMP \bullet PumpRepairedAck(p) \in outmess' \Leftrightarrow PumpRepaired(p) \in inmess \\
LevelRepairedAck \in outmess' \Leftrightarrow LevelRepaired \in inmess \\
SteamRepairedAck \in outmess' \Leftrightarrow SteamRepaired \in inmess \\
detections' = \emptyset \\
dfailures' = detections \cup \\
\quad (dfailures \setminus \{u : UNIT;\ p : PUMP \\
\qquad \mid (u = qmeter \land LevelRepaired \in inmess) \\
\qquad \lor (u = steameter \land SteamRepaired \in inmess) \\
\qquad \lor (u = pump\ p \land PumpRepaired(p) \in inmess) \\
\qquad \lor (u = pumpmeter\ p \land PumpCtrRepaired(p) \in inmess) \\
\qquad \bullet u\}) \\
unexpected' = (unexpected \setminus \\
\quad \{m : INMESS;\ p : PUMP \\
\quad \mid m \in inmess \land \\
\quad (m = LevelRepaired \lor m = LevelFailureAck \\
\quad\ \lor m = SteamRepaired \lor m = SteamFailureAck \\
\quad\ \lor m = PumpRepaired(p) \lor m = PumpFailureAck(p) \\
\quad\ \lor m = PumpCtrRepaired(p) \lor m = PumpCtrFailureAck(p)) \\
\quad \bullet m\} \qquad) \cup \\
\quad \{m, mm : INMESS;\ p : PUMP \\
\quad \mid m \in inmess \land \\
\quad ((LevelFailureAck = m \land mm = LevelRepaired) \\
\quad\ \lor (m = SteamFailureAck \land mm = SteamRepaired) \\
\quad\ \lor (m = PumpFailureAck(p) \land mm = PumpRepaired(p)) \\
\quad\ \lor (m = PumpCtrFailureAck(p) \land mm = PumpCtrRepaired(p))) \\
\qquad \bullet mm\} \cup \\
\quad \{m : INMESS;\ u : UNIT;\ p : PUMP \\
\quad \mid (LevelFailureAck = m \land qmeter \in detections) \\
\quad\ \lor (m = SteamFailureAck \land steameter \in detections) \\
\quad\ \lor (m = PumpFailureAck(p) \land pump\ p \in detections) \\
\quad\ \lor (m = PumpCtrFailureAck(p) \land pumpmeter\ p \in detections) \\
\quad \bullet m\} \\
log' = outmess \\
\hline
\end{array}
$$

In normal mode, no failure is detected. The messages of *inmess* are telling the truth. The informal specification is ambigous:

> As soon as the water level is below N_1 the level can be adjusted by the program by switching the pumps on.

Doest it mean all the pumps ? In any case, the effect is not immediate. If no pump is on in the current state, the water level will still be under N_1 at the next cycle. So, the same question will rise again: how many pumps should be switched on ? Here is proposed a loose specification, only stating that if no pump is on and the level of water is below N_1 then at least one pump should be switched on. Many different policies can be enforced at implementation level: minimizing the risk, trying to get the level over N_1 as quickly as possible ... If failures rarely occur, a reasonable decision would be to switch the pumps such that the average of the maximal and minimal levels of water of “boilers” in $Diverge(believed')$ would be as close as possible to N_2. The definition of the behaviour of the driver in normal mode is:

$$
\begin{array}{l}
\textit{NORMAL} \\
\hline
\Delta WS \\
\hline
mode = normal \Rightarrow \\
\quad believed' = \{b : BOILER;\ q, v : \mathbb{N} \\
\qquad \mid Level(q) \in inmess \wedge b.q = q \\
\qquad \wedge Steam(v) \in inmess \wedge b.v = v \\
\qquad \wedge b.p = p' \\
\qquad \wedge b.valve = False \\
\qquad \bullet b\} \\
\quad \wedge expected' = Diverge(believed') \\
\quad \wedge risk' = Divrisk(believed') \\
\quad \wedge outofbound(risk') \Rightarrow state' = exit \\
\quad \wedge \neg outofbound(risk') \Rightarrow state' = control \\
\quad \wedge((q < N_1 \wedge PumpOn = \emptyset) \Rightarrow \\
\qquad (\exists p : PUMP \bullet OpenPump(p) \in outmess')) \\
\quad \wedge((q > N_2 \wedge PumpOn = PUMP) \Rightarrow \\
\qquad (\exists p : PUMP \bullet ClosePump(p) \in outmess')) \\
\hline
\end{array}
$$

where *outofbound* is defined as expected:

$$
\begin{array}{l}
outofbound : \mathbb{P} BOILER \\
\hline
outofbound = \{b : BOILER \mid b.q < M_1 \vee b.q > M_2\}
\end{array}
$$

$$
\begin{array}{l}
\textit{DEGRADED} \\
\hline
\Delta WS \\
\hline
mode = degraded \Rightarrow \\
\quad believed' = \{b : BOILER;\ q, v : \mathbb{N} \\
\qquad \mid Level(q) \in inmess \land b.q = q \\
\qquad \land (steameter \in dfailure' \Rightarrow b.v \in 0 \ldots W) \\
\qquad \land (steameter \notin dfailure' \Rightarrow Steam(b.v) \in inmess) \\
\qquad \land b.p = p' \\
\qquad \land b.valve = False \\
\qquad \bullet b\} \\
\quad \land expected' = Diverge(believed') \\
\quad \land risk' = Divrisk(believed') \\
\quad \land outofbound(risk') \Rightarrow state' = exit \\
\quad \land \neg outofbound(risk') \Rightarrow state' = control \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textit{RESCUE} \\
\hline
\Delta WS \\
\hline
mode = rescue \Rightarrow \\
\quad believed' = \{b : BOILER \\
\qquad \mid b.q \in \{b : BOILER \mid b \in risk \land Steam(b.v) \in inmess \\
\qquad\quad \land b.p = p \\
\qquad\quad \bullet b.q\} \\
\qquad \land Steam(b.v) \in inmess \\
\qquad \land b.p = p' \\
\qquad \land b.valve = False \\
\qquad \bullet b\} \\
\quad \land expected' = Diverge(believed') \\
\quad \land risk' = Divrisk(believed') \\
\quad \land outofbound(risk') \Rightarrow state' = exit \\
\quad \land \neg outofbound(risk') \Rightarrow state' = control \\
\hline
\end{array}
$$

These are the most interesting cases. For the initialization period, the only problem is to know how the quantity of water varies while the valve is open. It probably depends on the quantity of water in the boiler and the shape of the boiler. With Z, it is possible to keep the issue open by stating that there is a fonction giving the throughout of the valve from the quantity of water in the boiler. Let assume that a schema $INITIALIZATION$ is given for this period. The schema $ALLCASES$ groups the specification parts specific to all stages and modes:

$$ALLCASES \mathrel{\widehat{=}} INITIALIZATION \land NORMAL \land DEGRADED \land RESCUE$$

$$FLcomputer \mathrel{\widehat{=}} (EXCEPTIONS \land EXIT) \lor \\ (\neg EXCEPTIONS \land COMMON \land ALLCASES)$$

$$\begin{array}{|l}
\multicolumn{1}{l}{FLcompletion} \\
\Xi WS \\
\hline
driverun = False \\
\hline
\end{array}$$

$$FLASH \mathrel{\widehat{=}} (FLphysical \land FLcomputer) \lor FLcompletion$$

3 Completion of the specification

The specification of the driver may give rise to implementations which will not put the boiler in danger but which are not really sensible. In order to validate an implementation, it should be possible to prove for instance that the initialization (the period during which the driver sizes the system) terminates. The informal specification gives some properties of the physical system in order to perform the proof:

> "... the program can send continuously the signal PROGRAM_READY until it receives the signal PHYSICAL_UNITS_READY which *must necessarily* be emitted by the physical units."

The requirement of termination of the initialization phase can be done in two ways. One possibility is to refine the previous specification in order to make the proof possible. But this would probably eliminate some models of the specification which satisfy the termination property. The other possibility is to add a proof obligation such as:

$$
\begin{array}{l}
\exists n : \mathbb{N} \\
\bullet (\forall s : \mathsf{seq}\ WS \\
\ \mid s(1) \in INIT \\
\ \wedge (\forall i : \mathbb{N} \mid i \mathbin{\mathsf{mod}} 2 = 0 \\
\quad \bullet (\exists WS;\ WS' \mid BLACKOUT \bullet \theta WS = s(i) \wedge \theta WS' = s(i+1))) \\
\ \wedge (\forall i : \mathbb{N} \mid i \mathbin{\mathsf{mod}} 2 = 1 \\
\quad \bullet (\exists WS;\ WS' \mid FLASH \bullet \theta WS = s(i) \wedge \theta WS' = s(i+1)) \wedge detection = \emptyset) \\
\ \bullet (\forall i : \mathbb{N} \mid i > n \bullet mode \neq Initialization))
\end{array}
$$

We could have a finer proof obligation by allowing some failures, but this shows that we can tackle livness properties with Z, without overspecifying.

4 Implementation

The specification is undeterministic. The states of the system are described with sets that may be very large. The set of detections is defined in a phase where the driver does not work. The program has to be deterministic, can only handle data of reasonable size and must compute when it is its turn. How to fill the gap between the specification and the program ?

4.1 Refinement of the specification

Refinement (see [Spi92, pages 135–141]) consists in

- designing a new specification,
- stating a correspondance between the new specification and the refined specification (roughly speaking, a translation of the states of the new specification in terms of the states of the refined specification and the association of the operations of the new specification with the operations of the refined specification),
- a proof that the models of the new specification are models of the refined specification

Refining the specification allows to design a representation of the states which can be trivialy implemented by data structures provided by programming languages. It should be noted that the representation of sets which are values of *expected* cannot be just given by three intervals (for water quantity, steam and water flows). In the definition of *diverge*, we can see clearly that the interval in which water quantity takes its value depends on the final value of steam flow. As a consequence, it could be detected in some situations that one of the units measuring water quantity or steam flow has failed, but without determining which one has failed. The informal specification suggests that we could take an approximation of the specification presented so that the sets values of *expected*,*believed* and *risk* can be represented by a triplet of intervals. This approximation can be made by changing the definition of *diverge*. Then it is possible to refine the specification taking triplets of pairs of values for representing the sets of possible situations.

The other aim of refinement is to get rid of indeterminacy. Different policies may be taken for this purpose. Anyhow, there is a major point: the system must be kept out of danger as long as possible. This may be in contradiction with the informal specification which says that the system should try to maintain the level of water between N_1 and N_2. A "clever" driver should take decisions such that eventual failures would not leave the boiler in trouble. There is no evidence that the best way of achieving this goal is to start pumps when the level is below N_1 and to stop all the pumps when the level is over N_2. The level at which some pumps should be switched on and the number of pumps to be switched depends on the current flow of steam and water as well as the number of pumps out of order.

4.2 Design of an algorithm

The previous step of refinement results in a specification which is determinist. This means that it was possible to prove that for any situation there was a unique set of messages to be issued by the driver. But this specification may be like "if there is at least one suitable answer, let take the best". But it might take a while to scan all the possible solutions and to compare them. At this stage, either a fast enough construction of the set of messages is found, or we have to rework the specification so that it is not too demanding. In that sense, the given target $[N_1, N_2]$ may lead to simple algorithms.

5 Discussion

Most of this paper is dedicated to a faithful formalisation of the text proposed. The formalisation raised many questions and permited to propose some answers:

- What is the role of N_1 and N_2 ? The design of a simple algorithm.
- How many pumps should be switched on ? This can be computed by evaluating the incidence of the (at most) four possibilities over several cycles (using *Diverge* and *Divrisk*) and by choosing the best figure.

- Is the detection procedure reliable ? We cannot expect a detection procedure 100% safe because it must always rely, to some extend, on the incoming messages. A better question would be: is the detection procedure acceptable or satisfactory ? Meaning: if the driver does not detect a failure, may we expect some severe incident. The answer to this question depends on the responsability of "the environment", the probability of a failure to be undetected. This can be analysed now that the detection mechanism is clearly defined.
- What is the throughput of the valve ? The question is open.
- ...

The major assets of Z for building this specification was the modularity and the possibility of not overspecifying. Modularity is provided by the definition of schemas by combination of others. Z allows a lot of schema combinations. Overspecification is avoided through undeterminism. Once the informal specification is formalised, it is possible to see exactly what the issues are in order to get a program. It was also demonstrated that liveness properties can be asserted in the framework of a Z specification.

Is this specification useful for the design of other drivers ? It is unlikely that parts of the specification presented are possible parts of another driver specification. But it is possible to draw an abstraction of the specification which would guide the development of others. There is a clear structure for the specification: the states divided into four parts, the different stages (which can be split up with modes), the definition of two transitions which are both split up along two views,... This can be used to build a specification from which the presented specification would be a refinement. In fact the presentation of the specification would have been appeared clearer if it had been presented as the refinement of a shorter specification. This means that refinement is not only useful for constructing programs but it is also a powerful way of building specifications. But refinement using Z is generally awkward. Indeed, very few examples of refinement proofs have been presented for Z case studies. The B method is a clear improvement in that respect. But it seems easier to get a first specification with Z. In any case the need for a suitable architecture of the specification is the key to the success.

References

[Abr94] Jean-Raymond Abrial. Steam-boiler control specification problem. Technical report, 1994.

[BL95] Pascal Bernard and Guy Laffitte. The french population census for 1990. In Jonathan P. Bowen and Michael G. Hinchey, editors, *ZUM'95: 9th International Conference of Z Users*, Lecture Notes in Computer Science, pages –. Springer-Verlag, 1995.

[BN92] Stephen Brien and John Nicholls, editors. *Z Base Standard Version 1.0.* Oxford University Computing Laboratory, Programming Research Group, November 1992.

[Spi92] Mike Spivey. *The Z Notation: a Reference Manual.* Series in Computer Science. Prentice Hall, 92.