A Methodology for co-design based on a healthcare case study

Luigia Petre,
Mauno Rönkkö,
Elena Troubitsyna,
Marina Waldén[*],
Micaela Jansson

Åbo Akademi University, Department of Computer Science,
Lemminkäisenkatu 14, FIN-20520 Turku, Finland

[*]Financed by the Academy of Finland

Turku Centre for Computer Science
TUCS Technical Report No 437
December 2001

ISBN 952-12-0940-2
ISSN 1239-1891

Abstract

Systems upon which a high degree of reliance can be asserted form the class of dependable systems. We present here a development method for a subclass of dependable systems, safety-critical control systems. This method supports simultaneous design of both hardware and software in a system where failures would cause significant losses. The method is based on using mathematically solid techniques that allow us to address safety and reliability issues early in the development cycle. We start with an abstract specification of the system and subsequently add details concerning functionality, safety and reliability in a stepwise manner. From the obtained specification we can deduce which parts of the system are implemented using software and which using hardware. We then decompose the specification into controller, sensors, actuators and plant. We use the formal development method B-action systems for our development process. The method has a good mechanised support via the tools associated with the B Method. We illustrate the development process on a medical laboratory device, i.e. a control oriented case study that is part of a healthcare system.

The work has been done within the EU-project MATISSE - Methodologies and Technologies for Industrial Strength Systems Engineering (IST-1999-11435, *http://www.matisse.qinetiq.com/*).

Keywords: Formal Methods, Control Systems, Hardware/Software Co-design, Action Systems, the B Method

TUCS Research Group

Programming Methodology Group

1. Background

In this report we present a development method for control systems used as part of a healthcare system, e.g., medical laboratory devices. The method supports simultaneous design of both hardware and software in the system, i.e., it supports *hardware/software co-design*. Moreover, the method is based on using mathematically valid techniques that allow us to address safety and reliability issues early in the development cycle. Thus, the method supports development of safety-critical and dependable control systems. Although we present here the method using B-action systems [BW98] as the notational formalism, the approach is general and adaptable also to other, similar, formalisms. Furthermore, although we concentrate here on the development of control systems used as part of healthcare systems, the same ideas apply to embedded systems in general.

Since the kind of control systems that we address in this report comprises both software and hardware components, the presented method supports co-design [PS00]. Thus, we start with an abstract specification of the whole system and add details to that system one by one. Some of these details are functional details, some are safety details, and some reliability details. As we add these details, we may not yet know if they are implemented using software or hardware. This becomes more evident from the specification as it becomes more precise. This process of adding details terminates, when we have a specification that reveals all the necessary components. From such a specification we may then decide which components are implemented using software, and which components are implemented using hardware. The advantage of such a stepwise development method is that we can address the safety and reliability issues without cluttering implementation details.

The reason why we choose to address safety and reliability issues as early as possible in the development process is that it is then inexpensive to try out alternative solutions and modify the overall specification accordingly. Moreover, testing safety of an implemented system, i.e., that the system will not harm humans, environment, and equipment, is not always possible. The same applies to testing reliability, i.e., the system functioning correctly under a given set of operating conditions.

Enhancement of both safety and reliability can be achieved by applying mathematically valid, *formal*, methods. In the past few years regulatory requirements have tightened, whereby there is a need for methods allowing rigorous analysis. Over the years, formal methods have been shown to address these issues early in the development cycle. This is important, because competition in the field constantly requires shortening the marketing window.

In this report we present the development method using one specific formal method, B-action systems [BW98,WS98]. The B Method [Abrial96] and the Atelier B tool provide us with a good mechanised support for refinement-based development. The B Method has already been successfully used in industrial projects. Furthermore, an extension of the method to handle distributed control systems has also been investigated with good results.

2. Control systems as part of a healthcare system

In this report we study the development of control systems used as part of a healthcare system. In particular, we study the use of stepwise hardware/software co-design method for the development of such systems. In the following we present our case study, and then discuss the development of control systems in general.

2.1 Presentation of the Wallac case study

PerkinElmer Life Sciences designs, manufactures, develops, and markets analytical systems for use in drug discovery, mass population screening, and other bioresearch and clinical diagnostics areas. PerkinElmer Life Sciences was formed from the international Wallac Group, and is today a worldwide organization with about one thousand employees.

PerkinElmer Life Sciences (later in the document referred to as Wallac) supplies complete analytical systems. The systems are used to provide researchers and clinicians with reliable determinations of 'difficult to measure' substances found in blood or other biological sample materials. Typically, the substance to be measured is tagged with a radionuclide or a fluorescent or luminescent label. The amount of the substance is then determined from the 'signal', or the amount of radiation or light given off by the label. The manufactured analytical systems include reagents, sample handling and measuring instrumentation, as well as computer software.

The team of the Computer Science Department at Aabo Akademi University (later in the document referred to as Aabo Akademi) is conducting a parallel development of a new product of Wallac's – Fillwell™, a workstation preparing samples [PerkinElmer01]. The workstation is shown in Figure 1. The system belongs to the class of products for drug discovery and bioresearch. The Fillwell microplate liquid handling workstation offers significantly advanced features in the line of the sample preparation systems. The Fillwell base unit consists of a dispense head dispensing liquid into microplates on a processing table. A gantry moves the dispense head with high precision and speed from one plate to another.

Fig. 1: The Fillwell microplate liquid handling workstation.

The Fillwell workstation is the first liquid handling system specifically designed for high density microplates. The system is modular and can therefore be customized into a variety of configurations. The dispense head can have up to 384 tips attached. With the tips the head can perform automated pipetting into plates with 96, 384 or 1536 wells. The head provides a precise dispensing with volumes from 0.5 to 300 μl. The processing table contains up to 6 plate positions. An extension with three plate positions is easily removable. In Fig. 1 this processing table extension has been removed. In order for the dispensing head to be able to reach all the positions on the processing table it is mounted on a gantry that can move in XYZ-directions. The precision of the gantry is very high with an accuracy of 100 μm. The main application of the Fillwell workstation is drug discovery. Within this application area the system can be used for microplate replication, for dilution, transfer and addition of the liquid in the plates, for reformatting of plates with different densities (number of wells), as well as for rapid plate filling to homogenous and cell based systems. The system can function as a standalone workstation or be integrated into a robot.

2.1.1 The setting of the case study

The case study is organised using an industry-as-laboratory approach. This means that the researchers in academia provide the formal methods and B expertise while the experts at the R&D department of the industrial partner bring the domain knowledge to the team.

In order to come up with a good methodology for the development we started with a smaller part of the case study concentrating on modelling the dispense head and its movement up and down as well as its dispensing of liquid. However, the branch of development of medical and pharmaceutical equipment is highly competitive. Because of that, the requirement to preserve confidentiality of the equipment under construction was very strict. As the product has been confidential until recently (mid September) we have developed a mirror case study, called the Robot analyser, in order to have a public case study as an example of the methodology. We have chosen our case study to model all the significant and critical aspects of the Fillwell case study. The mirror case study belongs to the same class of systems.

In the Fillwell case study we have a workstation dispensing to and aspirating from plates. Our mirror case study is a robot analyser that analyses samples on a plate that is placed on a movable operating table. The movement of the operating table corresponds to the movement of the dispense head, while the analysing corresponds to the dispensing. A first extension of the systems would be to take into consideration the XY-movement of the dispense head and, hence, the corresponding rotation of the robot to be able to work with more than one plate.

2.1.2 The requirements of the case study

The Robot analyser consists of two interacting parts, a Robot and an Analyser. We concentrate here on the Analyser. The task of the Robot is to place a plate to be analysed into the Analyser, as well as to remove an analysed plate from it.

The Analyser has an operating table – a horizontal surface on which a plate is placed by the Robot. The Analyser can move its operating table to pick up a plate from the Robot, move the table to its analysing component, analyse the plate, as well as move the table to a position from which the Robot can remove the plate. During the analysis it is checked that there is enough liquid on the plate before the analysis and that the right amount of liquid was used during the analysis. The quality of the analysis should be kept at a constantly high level.

The Robot has two arms, one for placing a plate in the Analyser and the other for removing this plate. These arms are at different heights. Therefore, the table has to be at different positions for placing and removing plates:

Upper end position: A plate is analysed by the analysing component
Middle position: Arm 1 of the Robot places a plate into the Analyser
Lower end position: Arm 2 of the Robot removes a plate from the Analyser

Initially the Analyser is at its middle position and it is empty.

Due to the fact that the operating table and the Robot arms may collide we define the following *safety requirements*:

- When the Robot is ready to load the Analyser, arm 1 may extend only if the table is in its middle position.
- When the Robot is ready to unload the Analyser, arm 2 may extend only if the operating table is in its lowest position.
- The table may move only when the respective Robot arm has retracted.
- The table must not move beyond its upper and lower end position.

During a typical course of events the following sequence of actions happens with a plate. Here, we have not considered the possible failures.

- The Analyser moves the operating table to the middle position (*move(zmid)*).

- The Analyser waits to receive a plate from the first Robot arm after the Robot has picked up a plate and rotated to the Analyser (*receive*).

- The Analyser moves the operating table to the upper end position (*move(zmax)*).

- The Analyser analyses the plate (*analyse*).

- The Analyser moves the operaing table to the lower end position (*move(zmin)*).

- The Analyser waits to deliver an analysed plate to the second Robot arm and the Robot rotates to pass the plate on to another device (*deliver*).

We can note that other courses of events are possible depending on in which order the commands *move, receive, analyse* and *deliver* are given. In order to prohibit the Analyser to analyse a plate and move an infinite number of times, only a certain number of moves are allowed to be performed in a row before some other action need to be taken by the Analyser. Similarly, only a certain number of analyses on one plate are allowed before the plate has to be delivered.

2.2 Modeling control systems

A control system, such as the Fillwell workstation in our case study, consists of four components: a *controller*, *sensors*, *actuators*, and a *plant*. These components are depicted in Figure 2.

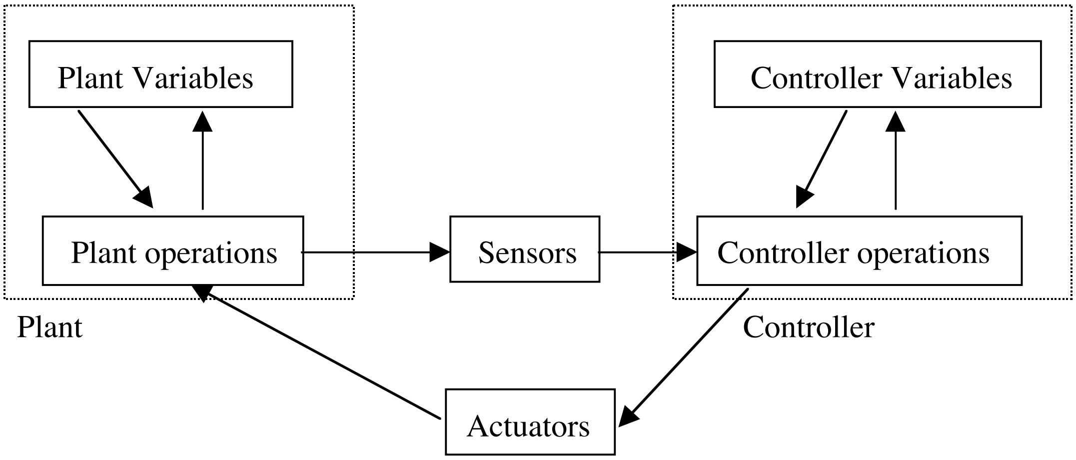

Fig. 2: The structure of the control system specification

The purpose of the controller in a control system is to ensure that the plant operates within the predefined requirements. For simplicity, we consider here only controllers with *responsive dynamics*. Such controllers become active only when they observe a discrepancy, or some other important event, in the plant.

As for the plant in a control system, we consider it here to have *autonomous dynamics*. This means that the plant changes its state by itself and, in a way, indepedently of the controller. Thus, even without the controller, the plant would keep on doing something by itself.

The purpose of the sensors and of the actuators is then to act as information *converters* between the controller and the plant. The sensors convert *measurements* from the plant into *readings* for the controller. Correspondingly, the actuators convert *commands* from the controller into *control signals* to the plant.

The difficulty in the development of a control system is in finding a suitable pair of control logic, sensors, and actuators that allow the controller to maintain the state of the plant within predefined tolerances. Clearly different choices affect the optimality of the overall performance. However, as a general guideline, one tries to minimize manufacturing costs and maximize reliability. This leads to the development of intelligent control logic to allow use of simple and reliable sensors and actuators. From this point of view, the use of a co-design method, i.e., a method that allows postponing of hardware details during the development process, supports these general development goals. As an example of postponing hardware details, we first consider a robot arm movement generally and only later introduce motor specific details, like acceleration and decceleration.

When thinking of control systems appearing as part of healthcare systems, another difficulty is that such systems should exhibit provable safety and realiability properties. Thus, during the development of a control system, one has to be able to incorporate safety and reliability measures into the system in such a way that these measures can be demonstrated to the customer when necessary. Typically this means increasing complexity in the control logic, as it has to cope with malfunctioning sensors and actuators, along with duplication of some physical sensors and, sometimes, even actuators.

3. Control system development

We describe here a *stepwise development method* [BS96] for developing a control system used as a part of a healthcare system. The reason why we prefer a stepwise development method to a verificational approach is that, in a verificational approach, it is difficult to localize the error source should the specification be inconsistent. Also, when using a stepwise development, we do not need to know all the requirements in advance; we can start the development process already with partial requirements. Moreover, a stepwise development method is scalable and can be used for the development of both small scale and large scale systems.

In a stepwise development method, we start from an initial specification and add details of functionality, safety, and reliability to it in a stepwise manner. Thus, as we proceed in the development process we produce more and more detailed specifications of the control system. For each development step we *prove* that the introduced details are consistent with the rest of the specification. Consequently, when we have introduced all the details, we have a provably correct specification of the whole control system that addresses also the safety and the reliability issues. We then proceed by decomposing that specification into separate specifications that model the controller, the sensors, the actuators, and the plant of the control system. A collection of implementable specifications that reflect the general control system architecture shown in Figure 2 is, hence, obtained.

Here, we illustrate the stepwise development method with B-action systems for which we use the tool Atelier B that supports the development process. The prover in Atelier B is used for proving the consistency of the introduced details. Although we limit the discussion here to the use of B-action systems, it is possible to combine it with other modeling approaches to the development process such as UML [UML1.4], as shown in report [PTWBEJ01].

3.1 Formulating the initial specification

The purpose of an initial specification is to describe the goal of the control system in an abstract manner. In other words, an initial specification does not describe the system in detail, rather, it describes the dynamics of the system in a very brief and general manner. An initial specification is by no means complete, still it captures all the possible dynamics in the system without any details.

Typically, an initial specification captures all the components in a control system along with their communication mechanisms. However, because of the missing details, all this information could be represented compactly by using *abstract data types* and only a few variables. A single variable in the initial specification could then represent without details, for instance, all the possible states of a controller. Similarly, a single variable in the initial specification could capture without details, for instance, all the different kinds of communication protocols between the four types of components, the controller, the actuators, the sensors, and the plant.

Because the purpose of the initial specification is to give a general, and rough, view to the entire system, it could also be given in an informal manner. For instance, in report [PTWBEJ01], we have investigated a possibility of giving the initial specification by using UML diagrams. Nevertheless, for the development process to get started, the initial specification has to be stated formally. Here we use B-action systems for this purpose. The initial specification of the Analyser is given in report [PTWBEJ01]. We consider the subroutines *move*, *analyse*, *deliver* and *receive* of the Analyser. Atelier B has been used when proving that the initial specification captures the included properties.

3.2 Introducing details

After we have formulated the initial specification, and verified that it captures the desired abstract properties of the system, we introduce details to it in a stepwise manner. Thus, we introduce only a few details at each step, and prove that these details are consistent with the rest of the specification. In this way, we obtain provably correct intermediate specifications, to which we can continue introducing more details in a similar manner.

In principle, the introduced details can be classified as follows:

1) communication mechanisms between the components,
2) local and global control policies for the components,
3) autonomous plant behaviour of the components, and
4) sensor and actuator dynamics of the individual components.

The order in which these details are introduced is very case specific. Also, it may well happen that some of the details cannot be introduced unless some other details are introduced before. Nevertheless, during the introduction of these details, we consider simultaneous relevant safety and reliability aspects. For instance, when introducing sensor dynamics, we should also consider that the reading from the sensors is not necessarily reliable [PTWBEJ01].

In B-action systems the refinement step consists of the introduction of details in machine refinements giving the safety and reliability requirements in invariants, and showing that the machine refinement refines the previous machine without the newly introduced details. The proof process is supported by Atelier B.

In our case study we add information about the presence of the plate, *blank*, and the position, *zpos*, of the operating table in the first step. An analysis can only be done with a plate present (*blank = TRUE*). Moreover, we keep track of the number of moves, *move_counter*, and analyses, *analyse_counter,* in this step. After a certain number of moves with the same plate the Analyser is only allowed to deliver the plate before it can continue doing something else. If it is repeatedly moving up and down without a plate, it will have to pick up a plate before it can continue moving. Each plate can only be analysed a certain number of times. We also add a waiting feature, *awaited*, to the Analyser. It waits ‘for a while’ for the Robot to be ready to deliver or receive. If the Robot does not become ready during this waiting, the Analyser considers the lack of co-operation as a fault.

In the second refinement step we introduce details on the analysis in form of variables giving the liquid amount, *pl_liq*, of the plate. There need to be enough liquid, constant *anl_liq*, on the plate for the analysis to be able to take place. After the analysis it is checked that the right amount of liquid was used. The result of these refinement steps on the Analyser as a B-action system is the refined Analyser, *Analyser2.ref,* and its included machines [PTWBEJ01] that can be found in Appendix A.

3.3 Determining controller and plant

When all the details are introduced, we proceed by decomposing the component specifications into controller and plant specification pairs. Thus, for each component we obtain both a controller specification and a plant specification. The obtained plant specification describes autonomous behaviour of the component, whereas the controller specification describes algorithms that guide the plant behaviour. Thus, the role of the controller is to react to changes in the plant.

In B-action systems terms, the plant machine contains the transitions to the new states, but before each such transition takes place, a controller machine procedure is called for handling the required decisions. If the controller machine needs not intervene, the procedure body is modeled with a stuttering operation *skip*. Thus, the controller machine procedures are thought of as interrupt procedures.

To obtain such a decomposition in B-action systems, the operations in the machines to be decomposed have to be split, and the variables are partitioned among the plant and the controller machines. As for the splitting of operations, each operation of the form

Oper = SELECT state=act1 ∧ A THEN B; state :=act2 END

is replaced with the operation in the plant

Oper' = SELECT state=act1 ∧ A THEN Act2 END,

where the procedure *Act2* of the controller is

Act2 = PRE state=act1 ∧ A THEN B; state:=act2 END.

Obviously, the effect of the new operation *Oper'* is the same as the effect of the old operation *Oper*. Adding procedures and keeping the old functionality agrees with the superposition refinement step.

As an example, let us consider the operation for delivering a plate. When the operating table with a plate is at the lower end position and the Robot is ready to take the plate *ReadyFor_blank*, the plate is removed (*blank := FALSE*) and the liquid level is set to zero. Also the variable modeling the Analyser waiting for the Robot is reset. The operation *Deliver* in the B-action system to be decomposed,

```
Deliver =   SELECT   astate1=aidle1 & acmd=deliver & plate=TRUE & blank = TRUE & zpos=zmin
            THEN    ReadyFor_blank;  ChangeBlank(FALSE); ResetPlateLiq; awaited:=FALSE;
                    astate1:=adeliver1  END,
```

is replaced with the operation *Deliver* in the plant,

```
Deliver  =  SELECT   astate1=aidle1 & acmd=deliver & plate=TRUE & blank = TRUE & zpos=zmin
            THEN   ReadyFor_blank; Ddeliver  END,
```

where the procedure *Ddeliver* of the controller is

```
Ddeliver =  PRE   astate1=aidle1 & acmd=deliver & blank=TRUE & plate=TRUE  & zpos=zmin
            THEN   ChangeBlank(FALSE) || ResetPlateLiq || awaited:=FALSE || astate1:=adeliver1
            END.
```

We can note that the procedure call on the imported procedure *ReadyFor_blank* in the operation *Deliver* remains in the plant. It can be seen as a procedure call to the controller of the system that exports the procedure. Since the procedures are given in an abstract machine construct, we have multiple assignment instead of sequential composition of assignments in the procedure *Ddeliver*.

The system *Analyser1.ref* in Appendix A is decomposed into two B-action systems: a plant and a controller. The plant that is refining *Analyser2.ref* (see Appendix A.1) is given in *AnalyserPlant.ref* in Appendix B.1. It contains the operations of the system and has, hence, the active role of the system. The controller *AnalyserCtrl.mch,* given in Appendix B.2, contains all the controlling procedures (one for each operation in the *Analyser1.ref*), as well as the global procedures used for signaling triggering events to other components. The local procedures of the refined Analyser in *A_proc2.mch* are local procedures of the controller in machine *Local_Actrl.mch* after the decomposition. The plant contains a reference **INCLUDES** to the controller machine for calling its procedures. When the plant wants to notify the controller of an event, it calls the corresponding control procedure. Every state change in the plant and the controller constitute a single atomic operation.

All the variables that are modified in the controller procedures are declared in the controller. The variables that are only read in these procedures are declared in the plant machine. The plant of the Analyser contains no variables. In the controller, on the other hand, the 'administrative' variables *astate1, acmd*, *awaited, move_counter* and *analyse_counter* are declared. The rest of the variables *zpos, blank* and *pl_liq* are modified in the controller procedures in *Local_Actrl.mch* and are also declared in this machine. The role of state *astate1* is to coordinate the plant and the controller machines. Hence, when the plant machine is about to change the state the corresponding control procedure is called.

The parallel decomposition of a B-action system into several B-action systems that have the same effect as the initial B-action system is only a structuring technique. Therefore, this control system development step is correct. Atelier B has been used for showing that the decomposition preserves the correctness of the original specification.

3.4 Determining sensors and actuators

In the last development step, we determine the sensors and the actuators for the components from the controller and plant specifications. In particular, the variables of the plant and the controller specification usually model more than just properties of these entities; some of these variables are used for communication. From these variables, we find sensors and actuators that are used to synchronize the plant specification with the controller specification.

In B-action systems, the actuators and sensors are global variables of the plant and the controller, and these variables are put into separate machines. The sensor variables are set by the plant and read by the controller, while the actuator variables are set by the controller and read by the plant. Due to this, the plant **INCLUDES** the sensor variables and **SEES** the actuator variables. Dually, the controller **INCLUDES** the actuator variables and **SEES** the sensor variables. An invariant relates the state of the whole control system to the plant and the controller state, as well as to the sensor variables and to the actuator variables.

In the Analyser the position of the operating table, *zpos*, as well as the amount of liquid on the current plate, *pl_liq*, are changed by the controller. Hence, the variable *zpos* and *pl_liq* model actuators. The local variable *blank* of the Analyser as well as the global variable *plate* are communication variables taking care of the synchronisation of the Analyser and the Robot. The variable *blank* models the view of the Analyser concerning the plate while the variable *plate* states the view of the Robot. The calls on the imported procedures, in this case the procedures of the Robot, can be seen as the sensors of the Analyser. The ideas concerning the sensors in this step form part of our ongoing research.

The operations of the plant refine the corresponding operations of the initial specification. The general decomposition schema using B-action systems [Sekerinski98] is shown in Figure 3. Encapsulating the actuator and sensor variables in separate machines allows abstracting from the details of a particular communication mechanism, which depends on the underlying hardware and operating environment. This allows late decisions regarding the hardware implementation.

```
REFINEMENT MachinePlant
REFINES Component_Ref
INCLUDES
  MachineCtrl, MachineSensors
VARIABLES
  plant state
INVARIANT
  variable types ∧
  refinement invariant for plant attributes,
  controller attributes, actuators, sensors
INITIALISATION
  plant state and sensors initialisation
OPERATIONS
  Pl_oper = SELECT Pl_oper_guard
            THEN plant state and
                 sensor change;
                 Ctrl_oper(parameters)
            END;
  …
END
```

```
MACHINE MachineCtrl
INCLUDES
  MachineActuators
SEES
  MachineSensors
VARIABLES
  controller state
INVARIANT
  variable types
INITIALISATION
  controller state and actuators initialisation
OPERATIONS
  Ctrl_oper(parameters) =
        PRE  Pl_oper_guard
        THEN  controller state and
              actuator change
        END;
  …
END
```

```
MACHINE MachineSensors
VARIABLES
  sensors
INVARIANT
  sensors types
INITIALISATION
  sensors :: any value
OPERATIONS
  SetSensor(ss) = sensor := ss;
  …
END
```

```
MACHINE MachineActuators
VARIABLES
  actuators
INVARIANT
  actuators types
INITIALISATION
  actuators :: any value
OPERATIONS
  SetActuator(aa) = actuator := aa;
  …
END
```

Fig. 3: Decomposition schema for control systms within B-action system formalism.

Applying the schema in Figure 3 on the Analyser gives us the final control system as given in Appendix C. This system consists of a plant *AnalyserPlant1.ref* including the controller *AnalyserCtrl1.mch*. The controller in turn includes the actuators in *Actuator_Analyser.mch,* as well as the communication variables in *Local_Analyser.mch* and *GlobalVar_Plate.mch*. The plant includes the imported procedures of the Robot via *R_Proc.mch*.

We can note that since we consider the procedure calls to the procedures of the Robot to be the sensors of the Analyser, the machine *R_proc.mch* corresponds to the machine *MachineSensors* in the schema above. Furthermore, in our case study the plant contains no variables.

With the tool Atelier B it is easily proved that the final control system is a refinement of the previous system without sensors and actuators. In fact this step involves merely rewriting and restructuring. Hence, the final control system is the result of a correct formal development process based on solid refinement techniques. Moreover, at this level the state variable becomes obsolete and can be replaced by up-to-date information received via the sensors.

4. Alternative approaches

The presented development method is a specialization of a general stepwise development approach introduced by Butler, Sekerinski, and Sere in [BSS01]. There are also other specialisations of this development approach adapted for various needs. Sekerinski has studied the use of the B Method for development of general control systems in [SS98]. Plosila has studied adaptation of the stepwise development method to co-design of electric circuits in [Plosila99]. Seceleanu has studied how the stepwise development method can be used in the design of synchronous digital circuits in [Seceleanu01]. Troubitsyna has studied the use of the stepwise development method in the development of dependable systems in [Troubitsyna00]. Lastly, in [RRS00] and [Rönkkö01] Rönkkö, Ravn, and Sere have studied the adaptation of the stepwise development method to control systems where the plant exhibits continuous-time dynamics, i.e., hybrid systems.

The presented stepwise development method with B-action systems uses shared variables for modeling communication; however, that is not the only alternative. Another way of modeling communication is to use functions. Then, the value parameters model the input and the result of the function model the output in that communication. Typically, functions model synchronous communication, whereas shared variables are used for modeling asynchronous communication.

Furthermore, the decomposition into plant and controller, as well as actuators and sensors is sometimes necessary earlier in the refinement process before all the details are introduced. However, there is a benefit of an early decomposition only if the decomposed components are independent of the rest of the system, i.e., the communication interface does not change any more.

5. Conclusion

In this report we have described a stepwise development method for the hardware/software co-design of a control system that operates as part of a healthcare system. The method allows a stepwise integration of safety and reliability details during the development. This, in turn, helps in considering safety and reliability early in the development cycle. Moreover, by supporting co-design, the method allows postponing of the implemenation details till last phases in the development cycle; thus, the method supports late binding of hardware details to the specification.

The presented development method suits well for the development of both small scale and large scale systems; however, the development of a large scale system may require additional management for keeping track of which components have been specified and to what detail. Moreover, with large scale systems, there is a bigger need for backtracking the development, i.e., to change some earlier defined specifications. Such a backtracking cannot be prevented, but the need for it can be reduced by integrating some early structural designs for the development process, for instance, by using UML diagrams.

As already mentioned, the development method as such can also be adapted to other formalisms than just B-action systems. For instance, one could use a theorem prover, such as PVS, to prove that the introduced details are consistent with the rest of the specification. However, in such an approach, one might have to provide and prove most of the refinement mechanisms that are already present and supported by Atelier B.

Acknowledgement

The authors would like to thank Kaisa Sere for her useful comments on the topics treated here.

References

[Abrial96] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[BS96] R.J.R. Back and K. Sere. From modular systems to action systems. *Software - Concepts and Tools 17*, pp. 26-39, 1996.

[BSS01] M. Butler, E. Sekerinski and K. Sere. An Action System Approach to the Steam Boiler Problem. In *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, LNCS 1165, Springer-Verlag, 1996.

[BW98] M. Butler and M. Waldén. Parallel programming with the B Method. Chapter 5 in [SS98], pp 183-195.

[PerkinElmer01]Fillwell™ 2002 – Features Guide. *http://www.abo.fi/~mwalden/Fillwell.pdf*

[PS00] L. Petre and K. Sere. Developing Control Systems Components. In *Proceedings of IFM'2000* - Second International Conference on Integrated Formal Methods, Germany, November 2000. LNCS 1945, pp. 156-175, Springer-Verlag.

[PTWBEJ01] L. Petre, E. Troubitsyna, M. Waldén, P. Boström, N. Engblom and M. Jansson. *A Methodology for integration of formal methods in a healthcare case study*. TUCS Technical Reports, No 436, Turku Centre for Computer Science, Finland.

[Plosila99] J. Plosila. *Self-Timed Circuit Design – The Action Systems Approach*. Ph.D. thesis, University of Turku, Dept. of Applied Physics, Lab. of Electronics and Information Technology, Turku, Finland, 1999.

[Rönkkö01] M. Rönkkö, *Stepwise Development of Hybrid Systems*. Turku Centre for Computer Science, TUCS, Ph.D. thesis No 35, November 2001.

[RRS00] M. Rönkkö, A. P. Ravn, and K. Sere. Hybrid Action Systems. Submitted to *Theoretical Computer Science*, 2000.

[Seceleanu01] T. Seceleanu. *Systematic Design of Synchronous Digital Circuits*. Turku Centre for Computer Science, TUCS, Ph.D. thesis No 32, May 2001.

[Sekerinski98] E. Sekerinski. Production cell. Chapter 6 in [SS98], pp. 197-254.

[SS98] E. Sekerinski and K. Sere (eds.). *Program Development by Refinement - Case Studies Using the B Method*. Springer-Verlag, 1998.

[Troubitsyna00] E. Troubitsyna. *Stepwise Development of Dependable Systems*. Turku Centre for Computer Science, TUCS, Ph.D. thesis No 29, June 2000.

[UML1.4] Unified Modeling Language (UML) 1.4 specification. *http://cgi.omg.org/cgi-bin/doc?formal/01-09-67*

[WS98] M. Waldén and K. Sere. Reasoning About Action Systems Using the B-Method. *Formal Methods in Systems Design 13(5-35),* 1998. Kluwer Academic Publishers.

Appendix A. The refined Analyser as a B-action system

A.1 The refined Analyser Analyser2.ref

The refined Analyser is given as the machine refinement *Analyser2.ref*. The liquid amounts *pl_liq, pl_liq_tmp* are the newly introduced variables in this refinement step. In the operations clause the global procedure *NewCommand* is given together with the subroutines *move*, *receive*, *analyse* and *deliver*. Each subroutine also contains failure and remedy operations.

The variables state the following:

astate1	- the state of the Analyser,
acmd	- the current command given to the Analyser,
awaited	- has the value TRUE if the Analyser has waited for the Robot in order to be able to co-operate with it,
move_counter	- the number of moves with the current plate setting,
analyse_counter	- the number of analyses performed on the current plate,
pl_liq_tmp	- amount of liquid on the plate in the beginning of a current analysis.

```
REFINEMENT
      Analyser2
REFINES
      Analyser1
SEES
      def
INCLUDES
      R_proc, A_proc2, GlobalVar_Plate
VARIABLES
      astate1, acmd, awaited, move_counter, analyse_counter, pl_liq_tmp
INVARIANT
      pl_liq_tmp:NAT &
      (blank=FALSE => pl_liq=0)
INITIALISATION
      astate1:=aidle1 || acmd:=receive || awaited:=FALSE ||
      move_counter,analyse_counter:=0,0 ||  pl_liq_tmp:=0
OPERATIONS
      NewCommand(cc)=
            PRE    cc:ACOMMAND
            THEN   SELECT   astate1=aidle1 & awaited=FALSE  THEN  acmd:=cc  END
            END;
```

```
/*****************************************************************/
          /* move to position 'pos' */
     MoveA(pos)=
          PRE     pos:NAT
          THEN       SELECT    astate1=aidle1 & acmd=move & move_counter<aMaxMoves
                     THEN      Safe;InRange(pos);MoveZ(pos);
                                astate1:=amove1;awaited:=FALSE;
                                move_counter:=move_counter+1
                     WHEN      astate1=aidle1 & acmd=move & move_counter=aMaxMoves
                                & blank=TRUE & pos=zmin
                     THEN      Safe; CheckPosNotOk(zmin); MoveZ(zmin); astate1:=amove1;
                                awaited:=FALSE
                     WHEN      astate1=aidle1 & acmd=move & move_counter=aMaxMoves
                                & blank=FALSE & pos=zmid
                     THEN      Safe;CheckPosNotOk(zmid);MoveZ(zmid);astate1:=amove1;
                                awaited:=FALSE
               END   END;

          /* waiting for the Robot to be Safe */
     MoveWait=
          SELECT  astate1=aidle1 & acmd=move & awaited=FALSE
          THEN  awaited:=TRUE END;

          /* failure - the Robot was not Safe */
     MoveAFail=
          SELECT  astate1=aidle1 & acmd=move & awaited=TRUE
          THEN    NotSafe;astate1:=asuspended1;awaited:=FALSE  END;

          /* protocol failure - the Analyser has moved aMaxMoves times and the next
                               move would not change its position */
     MoveANoPosChangeFail(pos)=
          PRE     pos:NAT
          THEN       SELECT  astate1=aidle1 & acmd=move & move_counter=aMaxMoves
                     THEN    CheckPosOk(pos);astate1:=asuspended1;awaited:=FALSE
                     END  END;

          /* protocol failure - the A. has moved aMaxMoves times. It has a plate
                               that it should deliver, but the given 'pos' isn't zmin */
     MoveAMinFail(pos)=
          PRE     pos:NAT
          THEN       SELECT  astate1=aidle1 & acmd=move & move_counter=aMaxMoves &
                               pos/=zmin & blank=TRUE
                     THEN    astate1:=asuspended1;awaited:=FALSE  END   END;

          /* fix the problem */
     MoveAMinRemedy=
          SELECT     astate1=asuspended1 & acmd=move & move_counter=aMaxMoves &
                     blank=TRUE & zpos/=zmin
          THEN       Safe;MoveZ(zmin);astate1:=aidle1  END;
```

```
        /* protocol failure - the A. has moved aMaxMoves times. It
                         should receive a plate, but the given 'pos' isn't zmid */
    MoveAMidFail(pos)=
        PRE     pos:NAT
        THEN        SELECT  astate1=aidle1 & acmd=move & move_counter=aMaxMoves &
                                pos/=zmid & blank=FALSE
                    THEN    astate1:=asuspended1;awaited:=FALSE  END   END;

        /* fix the problem */
    MoveAMidRemedy=
        SELECT      astate1=asuspended1 & acmd=move & move_counter=aMaxMoves &
                    blank=FALSE & zpos/=zmid
        THEN        Safe;MoveZ(zmid);astate1:=aidle1  END;

        /* the given parameter 'pos' is not in range */
    MoveARangeFail(pos)=
        PRE     pos:NAT
        THEN        SELECT  astate1=aidle1 & acmd=move
                    THEN    NotInRange(pos);astate1:=asuspended1;awaited:=FALSE
                    END   END;

        /* the A. has reached the position 'pos' */
    MoveAReady(pos)=
        PRE     pos:NAT
        THEN        SELECT  astate1=amove1 & acmd=move
                    THEN    CheckPosOk(pos);astate1:=aidle1  END   END;

        /* the A. has not reached the position 'pos' */
    MoveANotReady(pos)=
        PRE     pos:NAT
        THEN        SELECT  astate1=amove1 & acmd=move
                    THEN    CheckPosNotOk(pos);astate1:=amsuspended1  END   END;

        /* fix the problem */
    MoveRemedy(pos)=
        PRE  pos:NAT
        THEN        SELECT  astate1=amsuspended1 & acmd=move
                    THEN    Safe;MoveZ(pos);astate1:=amove1  END   END;

/******************************************************************/
        /* Analyse the plate */
    Analyse=
        SELECT  astate1=aidle1 & acmd=analyse & analyse_counter<aMaxAnalyse
        THEN        PlateInAnalyser; BlankInAnalyser; CheckPosOk(zmax);
                    CheckLiqAmt; pl_liq_tmp := pl_liq; AnalysePlate;
                    astate1:=aanalyse1; analyse_counter:=analyse_counter+1  END;

        /* general failure */
    AnalyseFail=
        SELECT astate1=aidle1 & acmd=analyse THEN astate1:=asuspended1  END;
```

```
    /* the A. has already analysed the plate aMaxAnalyse times */
AnalyseMaxFail=
    SELECT   astate1=aidle1 & acmd=analyse & analyse_counter=aMaxAnalyse
    THEN     astate1:=asuspended1  END;

    /* fix the problem */
AnalyseMaxRemedy=
    SELECT   astate1=asuspended1 & acmd=analyse & analyse_counter=aMaxAnalyse
    THEN   Safe;MoveZ(zmin);astate1:=aidle1  END;

    /* failure - no plate in Analyser */
AnalysePlateFail=
    SELECT   astate1=aidle1 & acmd=analyse
    THEN   NoPlateInAnalyser;astate1:=asuspended1  END;

    /* failure - the A. senses no plate */
AnalyseBlankFail=
    SELECT   astate1=aidle1 & acmd=analyse
    THEN   NoBlankInAnalyser;astate1:=asuspended1  END;

    /* fix the problem */
AnalyseBlankRemedy=
    SELECT   astate1=asuspended1 & acmd=analyse & blank=FALSE
    THEN   ChangeBlank(TRUE); astate1:=aidle1  END;

    /* failure - wrong position for analysing */
AnalysePosFail=
    SELECT  astate1=aidle1 & acmd=analyse
    THEN    CheckPosNotOk(zmax); astate1:=asuspended1  END;

    /* fix the problem */
AnalysePosRemedy=
    SELECT    astate1=asuspended1 & acmd=analyse & zpos/=zmax
    THEN      Safe; MoveZ(zmax); astate1:=aidle1  END;

AnalyseLiqFail=
    SELECT    astate1=aidle1 & acmd=analyse
    THEN      CheckLiqAmtNotOk; astate1:=asuspended1  END;

    /* the analysis is done */
AnalyseReady=
    SELECT  astate1=aanalyse1 & acmd=analyse
    THEN    AnalyseOk(pl_liq_tmp); astate1:=aidle1  END;

    /* failure - the analysis has not been done */
AnalyseNotReady=
    SELECT  astate1=aanalyse1 & acmd=analyse
    THEN    astate1:=aasuspended1  END;
```

```
    AnalyseNotOkReady=
        SELECT    astate1=aanalyse1 & acmd=analyse
        THEN      AnalyseNotOk(pl_liq_tmp); astate1:=aasuspended1  END;

        /* fix the problem */
    AnalyseRemedy=
        SELECT  astate1=aasuspended1 & acmd=analyse
        THEN    PlateInAnalyser; BlankInAnalyser;
                    CheckPosOk(zmax); astate1:=aanalyse1  END;

/***************************************************************/
        /* deliver a plate to the Robot */
    Deliver=
        SELECT  astate1=aidle1 & acmd=deliver
        THEN    PlateInAnalyser; BlankInAnalyser; CheckPosOk(zmin); ReadyFor_blank;
                  ChangeBlank(FALSE); ResetPlateLiq;
                  astate1:=adeliver1; awaited:=FALSE  END;

        /* general failure */
    DeliverFail=
        SELECT  astate1=aidle1 & acmd=deliver
        THEN    astate1:=asuspended1; awaited:=FALSE  END;

        /* waiting for the Robot to be ready */
    DeliverWait=
        SELECT  astate1=aidle1 & acmd=deliver & awaited=FALSE
        THEN    awaited:=TRUE  END;

        /* failure - no plate to deliver */
    DeliverPlateFail=
        SELECT  astate1=aidle1 & acmd=deliver
        THEN    NoPlateInAnalyser;astate1:=asuspended1;awaited:=FALSE  END;

        /* failure - A. senses no plate to deliver */
    DeliverBlankFail=
        SELECT  astate1=aidle1 & acmd=deliver
        THEN    NoBlankInAnalyser;astate1:=asuspended1;awaited:=FALSE  END;

        /* fix the problem */
    DeliverBlankRemedy=
        SELECT    astate1=asuspended1 & acmd=deliver & blank=FALSE
        THEN      ChangeBlank(TRUE);astate1:=aidle1  END;

        /* failure - operating table at wrong position */
    DeliverPosFail=
        SELECT  astate1=aidle1 & acmd=deliver
        THEN    CheckPosNotOk(zmin);astate1:=asuspended1;awaited:=FALSE  END;
```

```
        /* fix the problem */
    DeliverPosRemedy=
        SELECT   astate1=asuspended1 & acmd=deliver & zpos/=zmin
        THEN   Safe;MoveZ(zmin);astate1:=aidle1  END;

        /* failure - the Robot is not ready */
    DeliverFailAtRobot=
        SELECT  astate1=aidle1 & acmd=deliver & awaited=TRUE
        THEN    NotReadyFor_blank;astate1:=asuspended1;awaited:=FALSE  END;

        /* plate delivered ok */
    DeliverReady=
        SELECT  astate1=adeliver1 & acmd=deliver
        THEN    NoPlateInAnalyser;NoBlankInAnalyser;
                    astate1:=aidle1;move_counter:=0;analyse_counter:=0  END;

        /* general 'plate delivery not ok' */
    DeliverNotReady=
        SELECT  astate1=adeliver1 & acmd=deliver
        THEN    astate1:=adsuspended1  END;

        /* failure - there is still a plate in the A. */
    DeliverNotReadyPlate=
        SELECT  astate1=adeliver1 & acmd=deliver
        THEN    PlateInAnalyser;astate1:=adsuspended1  END;

        /* failure - the A. still senses a plate */
    DeliverNotReadyBlank=
        SELECT  astate1=adeliver1 & acmd=deliver
        THEN    BlankInAnalyser;astate1:=adsuspended1  END;

        /* fix the problem */
    DeliverRemedy=
        SELECT  astate1=adsuspended1 & acmd=deliver & pl_liq = 0
        THEN   NoPlateInAnalyser; Safe;
                    ChangeBlank(FALSE); MoveZ(zmin); astate1:=adeliver1   END;

/****************************************************************/
        /* receive a plate from the Robot */
    Receive=
        SELECT  astate1=aidle1 & acmd=receive
        THEN   NoPlateInAnalyser; NoBlankInAnalyser; CheckPosOk(zmid);
                    Available_blank; ChangeBlank(TRUE); PlateLiquidRead;
                    astate1:=areceive1; awaited:=FALSE   END;

        /* general failure */
    ReceiveFail=
        SELECT  astate1=aidle1 & acmd=receive
        THEN    astate1:=asuspended1;awaited:=FALSE   END;
```

```
        /* waiting for the Robot to be ready */
ReceiveWait=
        SELECT  astate1=aidle1 & acmd=receive & awaited=FALSE
        THEN    awaited:=TRUE   END;

        /* failure - there is already a plate in the A. */
ReceivePlateFail=
        SELECT  astate1=aidle1 & acmd=receive
        THEN    PlateInAnalyser;astate1:=asuspended1;awaited:=FALSE   END;

        /* failure - the A. already senses a plate */
ReceiveBlankFail=
        SELECT  astate1=aidle1 & acmd=receive
        THEN    BlankInAnalyser;astate1:=asuspended1;awaited:=FALSE   END;

        /* fix the problem */
ReceiveBlankRemedy=
        SELECT    astate1=asuspended1 & acmd=receive & blank=TRUE & pl_liq = 0
        THEN      ChangeBlank(FALSE);astate1:=aidle1   END;

        /* failure - operating table at wrong position */
ReceivePosFail=
        SELECT  astate1=aidle1 & acmd=receive
        THEN    CheckPosNotOk(zmid);astate1:=asuspended1;awaited:=FALSE   END;

        /* fix the problem */
ReceivePosRemedy=
        SELECT    astate1=asuspended1 & acmd=receive & zpos/=zmid
        THEN      Safe;MoveZ(zmid);astate1:=aidle1   END;

        /* failure - there is no plate available from the Robot */
ReceiveFailAtRobot=
        SELECT  astate1=aidle1 & acmd=receive & awaited=TRUE
        THEN    NoAvailable_blank;astate1:=asuspended1;awaited:=FALSE   END;

        /* plate received ok */
ReceiveReady=
        SELECT  astate1=areceive1 & acmd=receive
        THEN    PlateInAnalyser; BlankInAnalyser; astate1:=aidle1; move_counter:=0   END;

        /* plate not received properly */
ReceiveNotReady=
        SELECT  astate1=areceive1 & acmd=receive   THEN    astate1:=arsuspended1    END;

        /* failure - the A. has not got a plate */
ReceiveNotReadyPlate=
        SELECT  astate1=areceive1 & acmd=receive
        THEN    NoPlateInAnalyser;astate1:=arsuspended1   END;
```

```
        /* failure - the A. does not sense a blank */
    ReceiveNotReadyBlank=
        SELECT  astate1=areceive1 & acmd=receive
        THEN    NoBlankInAnalyser;astate1:=arsuspended1   END;

        /* fix the problem */
    ReceiveRemedy=
        SELECT  astate1=arsuspended1 & acmd=receive
        THEN     PlateInAnalyser; Safe;
                      ChangeBlank(TRUE); MoveZ(zmid); astate1:=areceive1   END;

/*******************************************************************************/
        /* general 'not ready' failure */
    AServiceNotReady=    SELECT  astate1=aidle1 THEN astate1:=asuspended1 END;

        /* fix the problem */
    ARemedy=             SELECT    astate1=asuspended1 THEN astate1:=aidle1 END;

        /* there is no remedy for the problem */
    AFailure=
        SELECT  astate1:{asuspended1,amsuspended1,aasuspended1,arsuspended1,adsuspended1}
        THEN    astate1:=aabort1   END
END
```

A.2 The sets of the Analyser def.mch

In *def.mch* we define the sets of states and commands used in the Analyser. The three possible positions of the operating table, as well as the maximum number of moves and analyses are given as constants.

```
MACHINE
    def
SETS
    ASTATE=  {aidle, amove, aanalyse, areceive, adeliver, asuspended, aabort};
    ASTATE1=  {aidle1, amove1, aanalyse1, areceive1, adeliver1, asuspended1, amsuspended1,
                  aasuspended1, arsuspended1, adsuspended1, aabort1};
    ACOMMAND= {move, analyse, deliver, receive};
CONSTANTS
    zmax, zmin, zmid, aMaxMoves, aMaxAnalyse
PROPERTIES
    zmax:NAT & zmin:NAT & zmid:NAT &
    zmax>zmin & zmax>zmid & zmin<zmid &
    aMaxMoves:NAT & aMaxAnalyse:NAT
END
```

A.3 The global variables *GlobalVar_Plate.mch*

The global variable plate is given in a separete machine *GlobalVar_Plate.mch*.

```
MACHINE
        GlobalVar_Plate
VARIABLES
        plate
INVARIANT
        plate:BOOL
INITIALISATION
        plate:=FALSE
OPERATIONS
        ChangePlate(bb)= PRE   bb:BOOL THEN  plate:=bb  END;

        PlateInAnalyser= SELECT  plate=TRUE  THEN  skip  END;

        NoPlateInAnalyser= SELECT  plate=FALSE  THEN  skip  END
END
```

A.4 The imported procedures *R_proc.mch*

The imported procedures from the Robot are given in *R_proc.mch*. These procedures are only given as *skip,* since the Analyser is not aware of their implementation.

```
MACHINE
        R_proc                  /* Global procedures */
OPERATIONS
                /* the Robot is Safe */
        Safe =      BEGIN       skip   END;

                /* th R. is not Safe */
        NotSafe =  BEGIN       skip   END;

                /* the R. has a plate available */
        Available_blank =       BEGIN       skip   END;

                /* the R. does not have a plate available */
        NoAvailable_blank =   BEGIN       skip   END;

                /* the R. is ready to unload the Analyser */
        ReadyFor_blank =       BEGIN       skip   END;

                /* the R. is not ready to unload the Analyser */
        NotReadyFor_blank =  BEGIN       skip   END
END
```

A.5 The procedures of the Analyser A_proc2.mch

The exported procedures that will be called by the Robot as well as the local procedures of the Analyser are given in machine *A_proc2.mch*.

The variables state the following:
zpos - the position of the operating table,
blank - has the value TRUE if the Analyser contains a plate, otherwise FALSE,
pl_liq - the amount of liquid in the current plate.

```
MACHINE
        A_proc2
SEES
        def,  Liq_def
VARIABLES
        zpos, blank, pl_liq
INVARIANT
        zpos:NAT  &  zpos:zmin..zmax  &  blank:BOOL & pl_liq:NAT
INITIALISATION
        zpos:=zmid  ||  blank:=FALSE || pl_liq:=0
OPERATIONS
                        /* Global procedures */

                /* the Analyser is ready to deliver a plate */
        ReadyToDeliver=         SELECT    zpos=zmin & blank=TRUE  THEN  skip  END;

                /* the A. is not ready to deliver a plate */
        NotReadyToDeliver=   SELECT    zpos/=zmin or blank=FALSE  THEN  skip  END;

                /* the A. is ready to receive a plate */
        ReadyToReceive=       SELECT    zpos=zmid & blank=FALSE THEN  skip  END;

                /* the A. is not ready to receive a plate */
        NotReadyToReceive= SELECT    zpos/=zmid or blank=TRUE THEN  skip  END;

                /* the A. senses a plate */
        BlankInAnalyser=        SELECT    blank=TRUE THEN skip END;

                /* the A. does not sense a plate */
        NoBlankInAnalyser=    SELECT    blank=FALSE THEN skip END;

/***************************************************************/
                        /* Local procedures */

                /* change the sensor 'blank' */
        ChangeBlank(bb)=       PRE  bb: BOOL  THEN  blank:=bb  END;
```

```
        /* move the operating table to 'pos' */
    MoveZ(pos) =     PRE  pos:NAT & pos:zmin..zmax   THEN zpos:=pos  END;

        /* check if 'pos' is in range */
    InRange(pos) =
        PRE  pos:NAT   THEN   SELECT  pos<=zmax  & pos>=zmin  THEN  skip  END  END;

        /* check if 'pos' is not in range */
    NotInRange(pos)=
        PRE  pos:NAT   THEN SELECT    pos<zmin or pos>zmax THEN skip END   END;

        /* check if the operating table is at 'pos' */
    CheckPosOk(pos)=
        PRE  pos:NAT  THEN  SELECT  pos=zpos  THEN  skip  END   END;

        /* check if the operating table is not at 'pos' */
    CheckPosNotOk(pos)=
        PRE  pos:NAT  THEN   SELECT  pos/=zpos  THEN  skip  END   END;

        /* read the liquid level of the plate */
    PlateLiquidRead=       BEGIN       pl_liq::(0..pmax)  END;

        /* reset the liquid level to zero */
    ResetPlateLiq=         BEGIN       pl_liq:=0     END;

        /*  check if there is enough liquid for analysing, anl_liq */
    CheckLiqAmt=           SELECT anl_liq<=pl_liq   THEN  skip  END;

        /*  check if there is not enough liquid for analysing, anl_liq */
    CheckLiqAmtNotOk=   SELECT  pl_liq<anl_liq  THEN  skip  END;

        /*  take an amount of liquid for analysing, anl_liq, from the plate */
    AnalysePlate=          PRE  pl_liq-anl_liq:0..pmax  THEN  pl_liq:=pl_liq-anl_liq  END;

        /* check if the right amount of liquid was used for the analysis */
    AnalyseOk(aa)=
        PRE  aa:NAT  THEN   SELECT  pl_liq+anl_liq=aa  THEN  skip  END   END;

        /* check if an incorrect amount of liquid was used for the analysis */
    AnalyseNotOk(aa)=
        PRE  aa:NAT  THEN   SELECT  pl_liq+anl_liq/=aa  THEN  skip  END   END
END
```

A.6 The constants of the Analyser *Liq_def.mch*

In *Liq_def.mch* we define the constants concerning the liquid used in the Analyser. The constant *anl_liq* states the amount of liquid needed for each analysis and *pmax* is the maximum capacity of each plate.

```
MACHINE
        Liq_def
CONSTANTS
        anl_liq, pmax
PROPERTIES
        anl_liq:NAT  &  pmax:NAT  &  anl_liq:0..pmax
END
```

Appendix B. Plant and controller

When the system is decomposed into plant and controller we get the following machines.

B.1 The plant of the Analyser AnalyserPlant1.ref

The plant is given in *AnalyserPlant.ref*.

```
REFINEMENT
        AnalyserPlant
REFINES
        Analyser2
SEES
        def, Liq_def
INCLUDES
        AnalyserCtrl, R_proc
OPERATIONS
        NewCommand(cc)=
                PRE     cc:ACOMMAND
                THEN        SELECT    astate1=aidle1 & awaited=FALSE
                            THEN       New_Command(cc)  END   END;

        MoveA(pos)=
                PRE     pos:NAT
                THEN        SELECT    astate1=aidle1 & acmd=move & move_counter<aMaxMoves &
                                       pos:zmin..zmax
                            THEN       Safe; Move(pos)

                            WHEN       astate1=aidle1 & acmd=move & move_counter=aMaxMoves &
                                       pos=zmin & zpos/=zmin
                            THEN       Safe; BlankInAnalyser; MoveMin(zmin)

                            WHEN       astate1=aidle1 & acmd=move & move_counter=aMaxMoves &
                                       pos=zmid & zpos/=zmid
                            THEN       Safe; NoBlankInAnalyser; MoveMid(zmid)
                            END    END;

        MoveWait=
                SELECT astate1=aidle1 & acmd=move & awaited=FALSE THEN MWait END;

        MoveAFail=
                SELECT    astate1=aidle1 & acmd=move & awaited=TRUE
                THEN      NotSafe; MoveFail  END;
```

```
MoveANoPosChangeFail(pos)=
    PRE   pos:NAT
    THEN      SELECT    astate1=aidle1 & acmd=move & move_counter=aMaxMoves &
                        zpos=pos
              THEN      NoPosChangeFail(pos)  END   END;

MoveAMinFail(pos)=
    PRE   pos:NAT
    THEN      SELECT    astate1=aidle1 & acmd=move & move_counter=aMaxMoves &
                        pos/=zmin
              THEN      BlankInAnalyser; MinFail(pos)  END   END;

MoveAMinRemedy=
    SELECT    astate1=asuspended1 & acmd=move & move_counter=aMaxMoves &  zpos/=zmin
    THEN      Safe; BlankInAnalyser; MinRemedy  END;

MoveAMidFail(pos)=
    PRE   pos:NAT
    THEN      SELECT    astate1=aidle1 & acmd=move & move_counter=aMaxMoves &
                        pos/=zmid
              THEN      NoBlankInAnalyser; MidFail(pos)  END   END;

MoveAMidRemedy=
    SELECT    astate1=asuspended1 & acmd=move & move_counter=aMaxMoves &  zpos/=zmid
    THEN      Safe; NoBlankInAnalyser; MidRemedy  END;

MoveARangeFail(pos)=
    PRE   pos:NAT
    THEN      SELECT  astate1=aidle1 & acmd=move & pos/:zmin..zmax
              THEN      RangeFail(pos)  END    END;

MoveAReady(pos)=
    PRE   pos:NAT
    THEN      SELECT  astate1=amove1 & acmd=move & zpos=pos
              THEN      MoveReady(pos)  END   END;

MoveANotReady(pos)=
    PRE   pos:NAT
    THEN      SELECT  astate1=amove1 & acmd=move & zpos/=pos
              THEN      MoveNotReady(pos)  END   END;

MoveRemedy(pos)=
    PRE  pos:NAT
    THEN      SELECT  astate1=amsuspended1 & acmd=move
              THEN    Safe; MRemedy(pos)   END   END;

/****************************************************************************/
Analyse=
    SELECT     astate1=aidle1 & acmd=analyse & analyse_counter<aMaxAnalyse &
              plate=TRUE & zpos=zmax & pl_liq>=anl_liq
```

```
    THEN      BlankInAnalyser;Aanalyse  END;

AnalyseFail=
    SELECT astate1=aidle1 & acmd=analyse   THEN  AFail  END;

AnalyseMaxFail=
    SELECT    astate1=aidle1 & acmd=analyse & analyse_counter=aMaxAnalyse
    THEN      MaxFail  END;

AnalyseMaxRemedy=
    SELECT    astate1=asuspended1 & acmd=analyse & analyse_counter=aMaxAnalyse
    THEN      Safe; MaxRemedy END;

AnalysePlateFail=
    SELECT     astate1=aidle1 & acmd=analyse & plate=FALSE  THEN  APlateFail  END;

AnalyseBlankFail=
    SELECT     astate1=aidle1 & acmd=analyse   THEN   NoBlankInAnalyser; ABlankFail END;

AnalyseBlankRemedy=
    SELECT     astate1=asuspended1 & acmd=analyse
    THEN      NoBlankInAnalyser; ABlankRemedy END;

AnalysePosFail=
    SELECT  astate1=aidle1 & acmd=analyse & zpos/=zmax  THEN  APosFail END;

AnalysePosRemedy=
    SELECT    astate1=asuspended1 & acmd=analyse & zpos/=zmax
    THEN      Safe; APosRemedy  END;

AnalyseLiqFail=
    SELECT    astate1=aidle1 & acmd=analyse & pl_liq<anl_liq   THEN   ALiqFail  END;

AnalyseReady=
    SELECT     astate1=aanalyse1 & acmd=analyse & pl_liq+anl_liq=pl_liq_tmp
    THEN      AReady  END;

AnalyseNotReady=
    SELECT    astate1=aanalyse1 & acmd=analyse  THEN  ANotReady  END;

AnalyseNotOkReady=
    SELECT    astate1=aanalyse1 & acmd=analyse & pl_liq+anl_liq/=pl_liq_tmp
    THEN      ANotOkReady  END;

AnalyseRemedy=
    SELECT    astate1=aasuspended1 & acmd=analyse & plate=TRUE & zpos=zmax
    THEN      BlankInAnalyser; AaRemedy END;
```

```
/****************************************************************************/
    Deliver=
        SELECT    astate1=aidle1 & acmd=deliver & plate=TRUE & zpos=zmin
        THEN      ReadyFor_blank; BlankInAnalyser; Ddeliver  END;

    DeliverFail=
        SELECT    astate1=aidle1 & acmd=deliver  THEN  DFail  END;

    DeliverWait=
        SELECT astate1=aidle1 & acmd=deliver & awaited=FALSE THEN  DWait  END;

    DeliverPlateFail=
        SELECT    astate1=aidle1 & acmd=deliver & plate=FALSE  THEN  DPlateFail1  END;

    DeliverBlankFail=
        SELECT    astate1=aidle1 & acmd=deliver  THEN   NoBlankInAnalyser; DBlankFail1  END;

    DeliverBlankRemedy=
        SELECT    astate1=asuspended1 & acmd=deliver
        THEN      NoBlankInAnalyser; DBlankRemedy  END;

    DeliverPosFail=
        SELECT    astate1=aidle1 & acmd=deliver & zpos/=zmin  THEN  DposFail  END;

    DeliverPosRemedy=
        SELECT    astate1=asuspended1 & acmd=deliver & zpos/=zmin
        THEN      Safe; DPosRemedy  END;

    DeliverFailAtRobot=
        SELECT    astate1=aidle1 & acmd=deliver & awaited=TRUE
        THEN      NotReadyFor_blank; DRobotFail  END;

    DeliverReady=
        SELECT    astate1=adeliver1 & acmd=deliver & plate=FALSE
        THEN      NoBlankInAnalyser; Dready  END;

    DeliverNotReady=
        SELECT    astate1=adeliver1 & acmd=deliver  THEN  DNotReady  END;

    DeliverNotReadyPlate=
        SELECT    astate1=adeliver1 & acmd=deliver & plate=TRUE  THEN  DPlateFail2  END;

    DeliverNotReadyBlank=
        SELECT    astate1=adeliver1 & acmd=deliver   THEN   BlankInAnalyser; DBlankFail2  END;

    DeliverRemedy=
        SELECT    astate1=adsuspended1 & acmd=deliver & plate=FALSE & pl_liq=0
        THEN      Safe; Dremedy  END;
```

```
/*************************************************************************/
   Receive=
      SELECT    astate1=aidle1 & acmd=receive & plate=FALSE & zpos=zmid
      THEN      Available_blank; NoBlankInAnalyser; RReceive  END;

   ReceiveFail=
      SELECT    astate1=aidle1 & acmd=receive  THEN  RFail  END;

   ReceiveWait=
      SELECT astate1=aidle1 & acmd=receive & awaited=FALSE THEN RWait  END;

   ReceivePlateFail=
      SELECT    astate1=aidle1 & acmd=receive & plate=TRUE  THEN  RPlateFail1  END;

   ReceiveBlankFail=
      SELECT    astate1=aidle1 & acmd=receive  THEN  BlankInAnalyser; RBlankFail1  END;

   ReceiveBlankRemedy=
      SELECT    astate1=asuspended1 & acmd=receive & pl_liq=0
      THEN      BlankInAnalyser;  RBlankRemedy  END;

   ReceivePosFail=
      SELECT    astate1=aidle1 & acmd=receive & zpos/=zmid  THEN  RPosFail  END;

   ReceivePosRemedy=
      SELECT    astate1=asuspended1 & acmd=receive & zpos/=zmid
      THEN      Safe; RPosRemedy  END;

   ReceiveFailAtRobot=
      SELECT    astate1=aidle1 & acmd=receive & awaited=TRUE
      THEN      NoAvailable_blank; RRobotFail  END;

   ReceiveReady=
      SELECT    astate1=areceive1 & acmd=receive & plate=TRUE
      THEN      BlankInAnalyser; RReady  END;

   ReceiveNotReady=
      SELECT    astate1=areceive1 & acmd=receive  THEN  RNotReady  END;

   ReceiveNotReadyPlate=
      SELECT    astate1=areceive1 & acmd=receive & plate=FALSE   THEN   RPlateFail2  END;

   ReceiveNotReadyBlank=
      SELECT   astate1=areceive1 & acmd=receive  THEN  NoBlankInAnalyser; RBlankFail2  END;

   ReceiveRemedy=
      SELECT    astate1=arsuspended1 & acmd=receive & plate=TRUE
      THEN      Safe; Rremedy  END;
```

```
/*****************************************************************************/
	AServiceNotReady=	SELECT astate1=aidle1 THEN ASNotReady  END;

	ARemedy=		SELECT astate1=asuspended1 THEN AARemedy  END;

	AFailure=
		SELECT  astate1:{{asuspended1,amsuspended1, aasuspended1, arsuspended1,
					adsuspended1}
		THEN    Failure END
END
```

B.2 The controller of the Analyser AnalyserCtrl.mch

The controller is given in *AnalyserCtrl.mch*. The global procedures of the Analyser are included in the controller.

```
MACHINE
	AnalyserCtrl
SEES
	def, Liq_def
INCLUDES
	GlobalVar_Plate, Local_Actrl
VARIABLES
	astate1, acmd, awaited, analyse_counter, move_counter, pl_liq_tmp
INVARIANT
	astate1:ASTATE1 &
	acmd:ACOMMAND &
	awaited:BOOL &
	move_counter:NAT &
	move_counter:0..aMaxMoves &
	analyse_counter:NAT &
	analyse_counter:0..aMaxAnalyse &
	pl_liq_tmp:NAT &

	(astate1=aanalyse1 => blank=TRUE& zpos=zmax) &
	(astate1=adeliver1 => blank=FALSE & zpos=zmin) &
	(astate1=areceive1 => blank=TRUE& zpos=zmid) &

	(astate1=aasuspended1 => acmd=analyse) &
	(astate1=amsuspended1 => acmd=move) &
	(astate1=adsuspended1 => acmd=deliver) &
	(astate1=arsuspended1 => acmd=receive) &

	(blank=FALSE => pl_liq=0)
INITIALISATION
	astate1:=aidle1 || acmd:=receive || awaited:=FALSE ||
	move_counter:=0 || analyse_counter:=0 ||  pl_liq_tmp:=0
```

```
OPERATIONS
                /* Global procedures */

        /* the Analyser is ready to deliver a plate */
    ReadyToDeliver=       SELECT    zpos=zmin & blank=TRUE  THEN  skip   END;

        /* the A. is not ready to deliver a plate */
    NotReadyToDeliver=   SELECT    zpos/=zmin or blank=FALSE  THEN  skip  END;

        /* the A. is ready to receive a plate */
    ReadyToReceive=      SELECT    zpos=zmid & blank=FALSE  THEN  skip  END;

        /* the A. is not ready to receive a plate */
    NotReadyToReceive= SELECT    zpos/=zmid or blank=TRUE  THEN  skip  END;

        /* the A. senses a plate */
    BlankInAnalyser=       SELECT    blank=TRUE THEN skip END;

        /* the A. does not sense a plate */
    NoBlankInAnalyser=   SELECT  blank=FALSE THEN skip END;

/****************************************************************/
                /* Local procedures */

    New_Command(cc)=
        PRE  cc:ACOMMAND & astate1=aidle1 & awaited=FALSE  THEN  acmd:=cc  END;

    Move(pos)=
        PRE  pos:NAT & astate1=aidle1 & acmd=move & move_counter<aMaxMoves &
                pos:zmin..zmax
        THEN  MoveZ(pos) || astate1:=amove1 || awaited:=FALSE ||
                    move_counter:=move_counter+1   END;

    MoveMin(pos)=
        PRE  pos:NAT & astate1=aidle1 & acmd=move & move_counter=aMaxMoves &
                blank=TRUE & pos=zmin & zpos/=zmin
        THEN   MoveZ(zmin) || astate1:=amove1 || awaited:=FALSE   END;

    MoveMid(pos)=
        PRE  pos:NAT & astate1=aidle1 & acmd=move & move_counter=aMaxMoves &
                blank=FALSE & pos=zmid & zpos/=zmid
        THEN   MoveZ(zmid) || astate1:=amove1 || awaited:=FALSE   END;

    MWait=
        PRE astate1=aidle1 & acmd=move & awaited=FALSE  THEN  awaited:=TRUE  END;


    MoveFail=
        PRE  astate1=aidle1 & acmd=move & awaited=TRUE
        THEN        astate1:=asuspended1 || awaited:=FALSE  END;
```

```
NoPosChangeFail(pos)=
        PRE  pos:NAT & astate1=aidle1 & acmd=move & move_counter=aMaxMoves &  zpos = pos
        THEN   astate1:=asuspended1 || awaited:=FALSE  END;

MinFail(pos)=
        PRE  pos:NAT & astate1=aidle1 & acmd=move & move_counter=aMaxMoves &
                pos/=zmin & blank=TRUE
        THEN   astate1:=asuspended1 || awaited:=FALSE  END;

MinRemedy=
        PRE  astate1=asuspended1 & acmd=move & move_counter=aMaxMoves &
                blank=TRUE & zpos/=zmin
        THEN  MoveZ(zmin) || astate1:=aidle1  END;

MidFail(pos)=
        PRE  pos:NAT & astate1=aidle1 & acmd=move & move_counter=aMaxMoves &
                pos/=zmid & blank=FALSE
        THEN  astate1:=asuspended1 || awaited:=FALSE  END;

MidRemedy=
        PRE  astate1=asuspended1 & acmd=move & move_counter=aMaxMoves &
                blank=FALSE & zpos/=zmid
        THEN   MoveZ(zmid) || astate1:=aidle1  END;

RangeFail(pos)=
        PRE  pos:NAT & astate1=aidle1 & acmd=move & pos/:zmin..zmax
        THEN   astate1:=asuspended1 || awaited:=FALSE  END;

MoveReady(pos)=
        PRE  pos:NAT & astate1=amove1 & acmd=move & zpos=pos   THEN   astate1:=aidle1  END;

MoveNotReady(pos)=
        PRE  pos:NAT & astate1=amove1 & acmd=move & zpos/=pos
        THEN   astate1:=amsuspended1  END;

MRemedy(pos)=
        PRE  pos:NAT & astate1=amsuspended1 & acmd=move
        THEN   MoveZ(pos) || astate1:=amove1  END;

/************************************************************************/
Aanalyse=
        PRE  astate1=aidle1 & acmd=analyse & analyse_counter<aMaxAnalyse &
                blank=TRUE & plate=TRUE & zpos=zmax & pl_liq>=anl_liq
        THEN   pl_liq_tmp := pl_liq || AnalysePlate || astate1:=aanalyse1 ||
                        analyse_counter:=analyse_counter+1  END;

AFail=
        PRE astate1=aidle1 & acmd=analyse THEN astate1:=asuspended1  END;
```

```
    MaxFail=
        PRE  astate1=aidle1 & acmd=analyse & analyse_counter=aMaxAnalyse
        THEN  astate1:=asuspended1  END;

    MaxRemedy=
        PRE  astate1=asuspended1 & acmd=analyse & analyse_counter=aMaxAnalyse
        THEN   MoveZ(zmin) || astate1:=aidle1  END;

    APlateFail=
        PRE  astate1=aidle1 & acmd=analyse & plate=FALSE   THEN   astate1:=asuspended1  END;

    ABlankFail=
        PRE  astate1=aidle1 & acmd=analyse & blank=FALSE   THEN   astate1:=asuspended1  END;

    ABlankRemedy=
        PRE  astate1=asuspended1 & acmd=analyse & blank=FALSE
        THEN   ChangeBlank(TRUE) || astate1:=aidle1  END;

    APosFail=
        PRE  astate1=aidle1 & acmd=analyse & zpos/=zmax   THEN   astate1:=asuspended1  END;

    APosRemedy=
        PRE  astate1=asuspended1 & acmd=analyse & zpos/=zmax
        THEN   MoveZ(zmax) || astate1:=aidle1  END;

    ALiqFail=
        PRE  astate1=aidle1 & acmd=analyse & pl_liq<anl_liq   THEN   astate1:=asuspended1  END;

    AReady=
        PRE  astate1=aanalyse1 & acmd=analyse & pl_liq+anl_liq=pl_liq_tmp
        THEN   astate1:=aidle1  END;

    ANotReady=
        PRE  astate1=aanalyse1 & acmd=analyse  THEN  astate1:=aasuspended1 END;


    ANotOkReady=
        PRE  astate1=aanalyse1 & acmd=analyse & pl_liq+anl_liq/=pl_liq_tmp
        THEN   astate1:=asuspended1  END;

    AaRemedy=
        PRE  astate1=aasuspended1 & acmd=analyse & blank=TRUE & plate=TRUE & zpos=zmax
        THEN   astate1:=aanalyse1  END;

/*****************************************************************************/
    Ddeliver=
        PRE   astate1=aidle1 & acmd=deliver & blank=TRUE & plate=TRUE  & zpos=zmin
        THEN   ChangeBlankAndResetPlateLiq(FALSE)  ||
                   astate1:=adeliver1 || awaited:=FALSE   END;
```

```
DFail=
    PRE  astate1=aidle1 & acmd=deliver  THEN  astate1:=asuspended1 || awaited:=FALSE  END;

DWait=
    PRE  astate1=aidle1 & acmd=deliver & awaited=FALSE   THEN   awaited:=TRUE  END;

DPlateFail1=
    PRE  astate1=aidle1 & acmd=deliver & plate=FALSE
    THEN   astate1:=asuspended1 || awaited:=FALSE  END;

DBlankFail1=
    PRE  astate1=aidle1 & acmd=deliver & blank=FALSE
    THEN   astate1:=asuspended1 || awaited:=FALSE  END;

DBlankRemedy=
    PRE  astate1=asuspended1 & acmd=deliver & blank=FALSE
    THEN   ChangeBlank(TRUE) || astate1:=aidle1  END;

DPosFail=
    PRE  astate1=aidle1 & acmd=deliver & zpos/=zmin
    THEN   astate1:=asuspended1 || awaited:=FALSE  END;

DPosRemedy=
    PRE  astate1=asuspended1 & acmd=deliver & zpos/=zmin
    THEN   MoveZ(zmin) || astate1:=aidle1  END;

DRobotFail=
    PRE  astate1=aidle1 & acmd=deliver & awaited=TRUE
    THEN   astate1:=asuspended1 || awaited:=FALSE  END;

DReady=
    PRE  astate1=adeliver1 & acmd=deliver & blank=FALSE & plate=FALSE
    THEN   astate1:=aidle1 ||  move_counter:=0 || analyse_counter:=0  END;

DNotReady=
    PRE  astate1=adeliver1 & acmd=deliver  THEN  astate1:=adsuspended1  END;

DPlateFail2=
    PRE  astate1=adeliver1 & acmd=deliver & plate=TRUE  THEN  astate1:=adsuspended1  END;

DBlankFail2=
    PRE  astate1=adeliver1 & acmd=deliver & blank=TRUE  THEN  astate1:=adsuspended1  END;

DRemedy=
    PRE  astate1=adsuspended1 & acmd=deliver & plate=FALSE & pl_liq=0
    THEN   ChangeBlankAndMoveZ(FALSE,zmin) ||  astate1:=adeliver1  END;
```

```
/*****************************************************************************/
    RReceive=
        PRE  astate1=aidle1 & acmd=receive & blank=FALSE &  plate=FALSE & zpos=zmid
        THEN   ChangeBlankAndPlateLiquidRead(TRUE)  ||
                        astate1:=areceive1 || awaited:=FALSE  END;

    RFail=
        PRE  astate1=aidle1 & acmd=receive  THEN  astate1:=asuspended1 || awaited:=FALSE  END;

    RWait=
        PRE  astate1=aidle1 & acmd=receive & awaited=FALSE    THEN   awaited:=TRUE  END;

    RPlateFail1=
        PRE  astate1=aidle1 & acmd=receive & plate=TRUE
        THEN   astate1:=asuspended1 || awaited:=FALSE  END;

    RBlankFail1=
        PRE  astate1=aidle1 & acmd=receive & blank=TRUE
        THEN   astate1:=asuspended1 || awaited:=FALSE  END;

    RBlankRemedy=
        PRE  astate1=asuspended1 & acmd=receive & blank=TRUE & pl_liq=0
        THEN   ChangeBlank(FALSE) || astate1:=aidle1  END;

    RPosFail=
        PRE  astate1=aidle1 & acmd=receive & zpos/=zmid
        THEN   astate1:=asuspended1 || awaited:=FALSE  END;

    RPosRemedy=
        PRE  astate1=asuspended1 & acmd=receive & zpos/=zmid
        THEN   MoveZ(zmid) || astate1:=aidle1  END;

    RRobotFail=
        PRE  astate1=aidle1 & acmd=receive & awaited=TRUE
        THEN   astate1:=asuspended1 || awaited:=FALSE  END;

    RReady=
        PRE  astate1=areceive1 & acmd=receive & blank=TRUE & plate=TRUE
        THEN   astate1:=aidle1 || move_counter:=0  END;

    RNotReady=
        PRE  astate1=areceive1 & acmd=receive  THEN  astate1:=arsuspended1  END;

    RPlateFail2=
        PRE  astate1=areceive1 & acmd=receive & plate=FALSE
        THEN   astate1:=arsuspended1  END;

    RBlankFail2=
        PRE  astate1=areceive1 & acmd=receive & blank=FALSE
        THEN   astate1:=arsuspended1  END;
```

```
    RRemedy=
          PRE  astate1=arsuspended1 & acmd=receive & plate=TRUE
          THEN   ChangeBlankAndMoveZ(TRUE,zmid) ll  astate1:=areceive1  END;

/*****************************************************************************/
    ASNotReady=    PRE  astate1=aidle1 THEN astate1:=asuspended1 END;

    AARemedy=      PRE  astate1=asuspended1 THEN astate1:=aidle1 END;

    Failure=       PRE  astate1:{asuspended1, amsuspended1, aasuspended1,
                                  arsuspended1, adsuspended1}
                   THEN  astate1:=aabort1  END
END
```

B.3 The variables of the Analyser *Local_Actrl.mch*

The global variable *plate* is unchanged and given in machine *GlobalVar_Plate.mch* as before (see Appendix A.3). The local variables are together with the local procedures given in the machine *Local_Actrl.mch*.

```
MACHINE
    Local_Actrl
SEES
    def, Liq_def
VARIABLES
    zpos, blank, pl_liq
INVARIANT
    zpos:NAT  &  zpos:zmin..zmax  &  blank:BOOL & pl_liq:NAT
INITIALISATION
    zpos:=zmid ll  blank:=FALSE ll pl_liq:=0
OPERATIONS
    ChangeBlank(bb)=      PRE  bb:BOOL THEN blank:=bb END;

    MoveZ(pos)=
          PRE  pos:NAT  &  pos:zmin..zmax  THEN  zpos:=pos   END;

    InRange(pos)=
          PRE  pos:NAT  THEN   SELECT  pos<=zmax & pos>=zmin  THEN  skip  END  END;

    NotInRange(pos)=
          PRE  pos:NAT  THEN   SELECT   pos<zmin or pos>zmax  THEN  skip  END  END;

    CheckPosOk(pos)=
          PRE  pos:NAT  THEN   SELECT  pos=zpos  THEN  skip  END   END;

    CheckPosNotOk(pos)=
          PRE  pos:NAT   THEN   SELECT  pos/=zpos  THEN  skip  END   END;
```

```
    ChangeBlankAndMoveZ(bb,pos)=
        PRE  bb:BOOL & pos:NAT & pos:zmin..zmax  THEN  blank:=bb || zpos:=pos END;

        /* read the liquid level of the plate */
    PlateLiquidRead=       BEGIN       pl_liq::(0..pmax)  END;

        /* reset the liquid level to zero */
    ResetPlateLiq=         BEGIN       pl_liq:=0     END;

        /*  check if there is enough liquid for analysing, anl_liq */
    CheckLiqAmt=           SELECT anl_liq<=pl_liq   THEN  skip  END;

        /*  check if there is not enough liquid for analysing, anl_liq */
    CheckLiqAmtNotOk=   SELECT  pl_liq<anl_liq  THEN  skip  END;

        /*  take an amount of liquid for analysing, anl_liq, from the plate */
    AnalysePlate=          PRE  pl_liq-anl_liq:0..pmax  THEN  pl_liq:=pl_liq-anl_liq  END;

        /* check if the right amount of liquid was used for the analysis */
    AnalyseOk(aa)=         PRE  aa:NAT  THEN   SELECT  pl_liq+anl_liq=aa  THEN  skip  END   END;

        /* check if an incorrect amount of liquid was used for the analysis */
    AnalyseNotOk(aa)=
        PRE  aa:NAT  THEN   SELECT  pl_liq+anl_liq/=aa  THEN  skip  END   END;

    ChangeBlankAndPlateLiquidRead(bb)=
        PRE  bb:BOOL THEN  blank:=bb || pl_liq :: (0..pmax)  END;

    ChangeBlankAndResetPlateLiq(bb)=
        PRE  bb:BOOL THEN  blank:=bb || pl_liq:=0  END
END
```

Appendix C. Actuators and sensors

C.1 The final plant of the Analyser AnalyserPlant1.ref

Finally, we introduce actuators and sensors. This step consists of four machines a plant, a controller, the actuators and the sensors. The plant is given in *AnalyserPlant1.ref.* The **OPERATIONS**-clause of the plant does not change in this step. Therefore, we exclude it here and refer to *AnalyserPlant.ref* in Appendix B.1.

```
REFINEMENT
    AnalyserPlant1
REFINES
    AnalyserPlant
SEES
    def, Liq_def
INCLUDES
    AnalyserCtrl1, R_proc
OPERATIONS
        . . .
END
```

C.2 The final controller of the Analyser AnalyserCtrl1.mch

The controller is given in *AnalyserCtrl1.mch*. There are very minor changes to the controller in this refinement step. Below we only give those parts of the refined controller that differ from *AnalyserCtrl.mch* in Appendix B.2. Since the variables *zpos* and *blank* were located in the same machine *Local_Actrl.mch,* they needed to be changed in the same procedure in *AnalyserCtrl.mch* due to B restrictions. After decomposing actuators and sensors, the variable *zpos* is an actuator located in *Actuator_Analyser.mch*, while the variable *blank* is a communication variable in *Local_Analyser.mch*. Hence, they can be changed in separate procedures in *DRemedy* and *RRemedy* as shown below.

```
MACHINE
    AnalyserCtrl1
SEES
    def
INCLUDES
        Actuator_Analyser, Local_Analyser, GlobalVar_Plate
. . .
```

```
OPERATIONS
    . . .
    Ddeliver=
        PRE  astate1=aidle1 & acmd=deliver & blank=TRUE & plate=TRUE  & zpos=zmin
        THEN   ChangeBlank(FALSE) || ResetPlateLiq  ||  astate1:=adeliver1 || awaited:=FALSE
        END;

    DRemedy=
        PRE  astate1=adsuspended1 & acmd=deliver & plate=FALSE
        THEN   ChangeBlank(FALSE) || MoveZ(zmin) ||  astate1:=adeliver1  END;

/*****************************************************************************/
    RReceive=
        PRE  astate1=aidle1 & acmd=receive & blank=FALSE &  plate=FALSE & zpos=zmid
        THEN   ChangeBlank(TRUE)  ||  PlateLiquidRead  ||  astate1:=areceive1 || awaited:=FALSE
        END;

    RRemedy=
        PRE  astate1=arsuspended1 & acmd=receive & plate=TRUE
        THEN   ChangeBlank(TRUE) || MoveZ(zmid) ||  astate1:=areceive1  END;
    . . .
END
```

C.3 The actuators of the Analyser Actuator_Analyser.mch

We consider the calls on the global imported procedures from the Robot in *R_proc.mch* to be the sensors. The actuators *zpos* and *pl_liq* are given in machine *Actuator_Analyser.mch*.

```
MACHINE
    Actuator_Analyser
SEES
    def, Liq_def
VARIABLES
    zpos, pl_liq
INVARIANT
    zpos:zmin..zmax & pl_liq:NAT
INITIALISATION
    zpos:=zmid || pl_liq:=0
OPERATIONS
    MoveZ(pos)=             PRE  pos:NAT  &  pos:zmin..zmax  THEN  zpos:=pos   END;

    CheckPosOk(pos)=        PRE  pos:NAT  THEN   SELECT  pos=zpos  THEN  skip  END   END;

    CheckPosNotOk(pos)=     PRE pos:NAT  THEN   SELECT  pos/=zpos  THEN  skip  END  END;
```

```
    InRange(pos)=
        PRE  pos:NAT  THEN   SELECT  pos<=zmax & pos>=zmin THEN  skip  END   END;

    NotInRange(pos)=
        PRE  pos:NAT  THEN   SELECT  pos<zmin or pos>zmax  THEN  skip  END   END;

    PlateLiquidRead=       BEGIN      pl_liq::(0..pmax)  END;

    ResetPlateLiq=         BEGIN      pl_liq:=0    END;

    CheckLiqAmt=           SELECT anl_liq<=pl_liq   THEN  skip  END;

    CheckLiqAmtNotOk=      SELECT  pl_liq<anl_liq  THEN  skip  END;

    AnalysePlate=          PRE  pl_liq-anl_liq:0..pmax  THEN  pl_liq:=pl_liq-anl_liq  END;

    AnalyseOk(aa)=         PRE  aa:NAT   THEN   SELECT  pl_liq+anl_liq=aa  THEN  skip  END   END;

    AnalyseNotOk(aa)=      PRE  aa:NAT   THEN   SELECT  pl_liq+anl_liq/=aa  THEN  skip  END   END
END
```

C.4 The communication variables of the Analyser Local_Analyser.mch

The communication variable *plate* is given in machine *GlobalVar_Plate.mch* as before (see Appendix A.3), while the communication variable *blank* is given in machine *Local_Analyser.mch*.

```
MACHINE
    Local_Analyser
VARIABLES
    blank
INVARIANT
    blank:BOOL
INITIALISATION
    blank:=FALSE
OPERATIONS
    ChangeBlank(bb)=     PRE  bb:BOOL THEN blank:=bb END
END
```

Turku Centre for Computer Science
Lemminkäisenkatu 14
FIN-20520 Turku
Finland

http://www.tucs.abo.fi/

University of Turku
• Department of Mathematical Sciences

Åbo Akademi University
• Department of Computer Science
• Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
• Institute of Information Systems Science