Universitè de Lorraine

Erasmus Mundus

Dependable Software Systems

Hemodialysis in Event-B

Author:
Anders Olav Candasamy

Supervisor:
Dominique Méry

April 25, 2016

1 Definitions

Machine ==¿ Hemodialysis machine

2 Variables

This section will detail various techniques that have been used to extract the required Event-B variables from the requirements.

2.1 Variable categories

In the case of the hemodialysis machine, there are a large amount of variables. This can be confirmed by reading the case study by in [x]. The variables required can be grouped into different categories. Categorising these variables can greatly help in improving our knowledge of the system. The concrete definition of the categories are left informal. We want our

Controlled are variables that the system has full control over

Monitored are variables that can only be observed and not directly modified.

User Input are monitored variables inserted by a nurse. Strictly $UserInput \subseteq Monitored$

Timed are monitored variables that have a dependency on time. Strictly $Timed \subseteq Monitored$

The two major categories of interest are the *controlled* and *monitored* variables. The User Input category is created to separate the internally monitored variables from the variables set by a nurse. This is done because the list of variables added by a nurse is such a large percentage of total variables. We have also decided to separate the monitored variables that include an element of time. As Event-B does not directly support time, we will extract them to their own category. These timed variables will be added at the very last machine refinement.

An example of a *controlled* variable is our blood pump. The software is in full control of the state of the pump. An example of a monitored variable is the *blood flow direction*. Although we directly control the pump, we do not control the blood flow direction. A *controlled* variable modifies the *environment* that again modifies the *monitored* variables. In other words, the system is unable to directly control the blood flow. An argument could be made that the if the pump is on, it is guaranteed that the blood flows rotation is positive. This is however making assumptions about the system that we do not know. In any case, our paper will define *controlled variables* as the variables the system has full and direct control over.

The idea of splitting variables into Controlled and Monitored came from reading [x]. Although Parnas presented this as a minor point in his paper, it is an interesting approach with regards to Event-B. This list is not an exhaustive list as other machines may require additional categories.

3 Machine Refinements

A defining characteristic of Event-B is its refinement of machines. At the start of the modelling process, we have our most abstract representation of the system. This abstract machine is then refined into a new machine that contains more details. The choice of the details is upto the modeller. Refinements usually fall within two categories; either a feature extension or a data refinement. Feature extensions usually implies that a machine has additional events or variables introduced. A data refinement is, as its name implies, refining the data of a machine. It is used when we have defined some data very abstractly and what to refine it to be more concrete. For example, a traffic light might initially only be either on or off. We can refine it to have red, green and yellow lights. We can link these variables together by adding an invariant that shows their equivalence. This is known a gluing invariant as it is gluing/connecting invariants.

$$abstractLight = ON \Leftrightarrow \{GREEN\} = getLights(concreteLight)$$

$$abstractLight = OFF \Leftrightarrow \{RED\} = getLights(concreteLight)$$

In this example, when the *abstractLight* is turned on the *concreteLight* must be GREEN. While OFF must be RED.

The choice of both abstraction level and incremental refinements is one that should be taken with great care.

4 Event-B Models

– Notes on Event-B machine –

4.1 HD2

Ignore all values, focus only on the very abstract level. Don't have physical entities, like pumps that can be on or off. HDM1 - Patient blood never goes below 1.

Fails at refinement 2. Can not clean the blood before giving it is pumped back to the patient.

HD3

HDM10 If bloodpump in on, the system increases its amount of blood currently inside the system. Blood inside the system can be cleaned if the ultrafiltration pump is turned on and there is blood in the system. Cleaned blood is removed from the system. IN real life it is returned to the patient.

HDM100 For any reason, we may turn off the Blood pump

HDM1000 Can only turn on blood pump when the alarm is off. Alarm can be turned on for any reason

HDM10000

Need a function that takes a SystemState and return its substates. Cleaning, connecting etc. Added SystemPhaseCtx for the 3 system phases and new csystemPhase variable. Also added an event to change a the variable to different phases. Added events that contains monitored variables that cause the blood pump to stop and activate an alarm. For now, the alarm is turned on and the BP can only be turned off when the alarm is on.

SystemPhaseCtx1 Added the SUBPHASES of the HD treatment.

$$SUBPHASE \in SystemPhases \longrightarrow SubPhases$$