Abstract State Machines and System Theoretic Process Analysis for Safety-Critical Systems

Farah Al-Shareefi, Alexei Lisitsa, and Clare Dixon

Department of Computer Science, University of Liverpool
Liverpool, L69 3BX, UK
{F.M.A.Al-Shareefi,lisitsa,cldixon}@liverpool.ac.uk

Abstract. The Abstract State Machine (ASM) method is a formal specification and modeling technique that allows us to specify computational systems at the required abstraction level and facilitates formal analysis and verification. System Theoretic Process Analysis (STPA) is a semi-formal hazard analysis method that aims to identify safety requirements emerging from the analysis of potential interactions among components and inadequate control in the system's design. In this paper, we combine these two techniques to develop a methodology capturing both the formal representation of ASM with the ability to generate safety properties from the STPA hazard analysis. This has the advantages of verifying the STPA requirements in a formal way, and giving insights for the improvement of the ASM specification, depending on these requirements. We illustrate our methodology by applying it to an insulin pump control system case study, showing what safety issues it highlights.

Keywords: Abstract State Machines, System Theoretic Process Analysis, Temporal Logic, Validation and Verification

1 Introduction

Due to the increasing adoption of software in safety critical systems, together with their potential failure, it has become imperative to develop safe and efficient systems before deployment. We address this issue by combining a particular formal method with the results of a particular safety analysis technique.

Within the existing variety of formal methods, the Abstract State Machine (ASM) method can seamlessly direct the development process of computational systems, from capturing the requirements to practical implementation [11]. Several modeling and analysis tools have been developed for ASMs. In this paper, we have chosen the set of interoperable tools integrated in a meta modelling framework called ASMETA [9], which includes automatic tools for the editing [13], validation [12], and verification [6] of ASM models. These tools help the modeler to develop an appropriate model for the functional requirements.

System Theoretic Process Analysis (STPA) is a safety analysis technique from the safety engineering domain [19]. It was developed to analyse complex modern systems that involve interactions between their software, hardware, human and environment components. This technique uses a non-linear accident causation model for the whole system, where the failure of interaction between the system's components or component failures can lead to unsafe states. It has been demonstrated that this technique is able to identify a wide range of hazard causes and safety requirements in a semi-formal manner [24]. Recently, the STPA technique has been used, in [1,3], as an integrated tool with verification activity by supplying it with the formulated safety requirements. However, the formalization process of safety requirements is both cumbersome and does not accurately capture some of the temporal aspects of the requirements.

In this paper, we present a methodology for developing correct and safe critical systems, that is based on the ASM method, the STPA technique, and temporal logic. It starts with modelling the system in the AsmetaL (ASMETA Language) to obtain an accurate mathematical representation. Then, using the AsmetaV validation tool, the model validation process is applied to ensure that it meets the functional requirements. Next the STPA technique is utilized to elicit safety requirements. These requirements are formalized into Linear Temporal Logic (LTL) that can be verified against an AsmetaL model using the AsmetaSMV verification tool.

The methodology is illustrated through the Insulin Pump Control System (IPCS) [23]. This system is chosen because its design provides a plausible level of complexity for research, and its safety aspects are still under scrutiny.

The main contributions of this paper are: a systematic methodology for developing safety critical systems through combining ASM with STPA, with the target of developing safe specifications, and adequate and concise temporal formalizations of the STPA requirements.

The rest of the paper is organized as follows: Section 2 is an overview of the tools and techniques of our methodology. Section 3 presents our case study. In Section 4, we describe our methodology. The application of this methodology is explained in Section 5. In Section 6, we evaluate our methodology. Section 7 discusses the related work. Section 8, finally, concludes the paper.

2 Background

2.1 Abstract State Machines

Abstract State Machines (ASMs), were originally introduced by Gurevich [15] as a versatile and extended way of representing Finite State Machines, where unstructured control states are replaced by multi-sorted first order structure states. Using ASMs, the modeler can specify the system from a high-level of abstraction, called a ground model, to the required detailed one [11]. ASM is based on abstract states, to model the system's structure, and on transition rules, to model the system's dynamic behavior. An ASM state is denoted by a pair (location, location value). The location is represented by an n-ary function name and its list of first-order terms, while the location value is a value assigned to that location. The ASM locations or functions can be *static*, which never change during any run of the machine, or *dynamic*, which may be changed by the environment or by machine updates. The dynamic functions are also differentiated between *controlled* (read and write by the machine), and *monitored* (read by the machine and write by the environment).

Changing an ASM state is performed by a control logic rule that has the following format:"if *Condition* then *Update*".This rule is invoked at the current state to produce the subsequent state. In addition to if-then, there is a set of rule constructors, such as par (parallel execution of the grouped rules), choose (non deterministic selection) and switch case (extension of the control logic rule).

A set of tools have been developed around ASMs to support the ASM method, and to help the developer in performing different analysis activities within the same development platform. These tools are included in a meta modelling framework called ASMETA (ASM mETAmodelling)[1] [9]. The ASMETA tools that have been utilized in this paper are as follows: (1) The ASMETA Simulator (AsmetaS) tool [13], which executes ASM models that are written in ASMETA Language (AsmetaL). (2) The ASMETA Validator (AsmetaV) tool [12] which validates AsmetaL specifications by scenarios written in ASMETA validation language, named Avalla. Avalla expresses the execution for a scenario in an algorithmic manner via a set of constructs: set (determine the values for monitored functions), check (inspect the machine state), step (perform one transition into another state), and step until (perform several transitions). The AmetaV tool captures any violation of Avalla scenario by producing only success or fail validation verdicts. (3) The ASMETA SMV (AsmetaSMV) model checker tool [6] which is for formal verification of ASMs. The inputs for this tool are the AsmetaL model and temporal properties, which can be written in either LTL or CTL. The AsmetaSMV translates these inputs into the NuSMV model checker. In this paper, we will use only LTL properties. The propositional and future–time connectives that are available for writing LTL properties in AsmetaL are: ! (not), iff ('if and only if', $\leftrightarrow$), and ($\wedge$), or ($\vee$), implies ($\rightarrow$), x ('next state', $\bigcirc$), g ('globally', $\Box$), and u(p, q) ('until', p U q).

2.2 System Theoretic Process Analysis

System Theoretic Process Analysis (STPA) is a hazard analysis technique that was proposed by Leveson [17] to address safety as a control problem. It considers the system's component interaction and dynamic behavior, rather than considering component failure only.

Typically, in this technique, the system is deemed as a safety control loop which consists of a controller, actuator, sensor and controlled process. The controller includes a model of the process it is controlling, in order to identify the requisite control action to be issued. The actuator executes this action on the controlled process, and the sensor returns the current status data about the controlled process to the controller. The analysis through STPA focuses on identifying the context and timing conditions that affect the action to make it a hazardous action.

[1] http://asmeta.sourceforge.net/

Implementation of this technique is outlined in five steps [18], as follows: (1) Identify system analysis fundamentals by determining expected accidents, and the potential hazards that can lead to these accidents. (2) Identify unsafe control actions under different timing conditions (provided at any time, provided too early/too late, not provided), as well as determining the controller process model (a set of environmental and system variables) which can contribute to providing the control action. (3) Ask an expert to determine which control action, with which combination of values taken by the environmental and system variables, and under which timing conditions, is a hazardous action. (4) Translate the hazardous control actions into safety requirements. (5) Determine the potential causes of each unsafe control action depending on the expert. In our work, we will focus on the first four steps.

3 Case Study: The Insulin Pump Control System

The Insulin Pump Control System (IPCS) is a therapeutic system used to improve diabetes treatment. The problem with traditional treatments is the possibility of taking an insulin overdose or insufficient dose due to focusing only on the current glucose value and ignoring the last insulin injection time. It has been chosen as a case study for software analysis of safety-critical systems in [23]. The IPCS works in three different modes: automatic, manual, and switching off. In the automatic mode, the software controller can implement one of the following two activities at a time: running (performed every 10 minutes) or testing (performed every 30 seconds). In addition, the software controller resets the cumulative dose to 0 every 24 hours. The running activity starts with sensing the current glucose value, then it analyses this value by comparing it with two saved values (10 and 20 minutes prior) to calculate the required dose. Before delivering the dose, it does a safety check, considering the maximum daily dose and maximum single dose. During the delivery of the dose, the controller sends pulses equivalent to each unit of the delivered dose. Within the running activity, the warning alarm must be run when the received glucose value is less than the minimum safe limit, the available insulin is less than or equal to four maximum single doses, or delivering the dose will exceed the maximum daily dose. The testing activity involves detecting any hardware unit failure (sensor, battery, needle, insulin reservoir) to suspend the IPCS work and to run the failure alarm. In the manual mode, the system will deliver the dose manually, hence the software controller will not perform safety checking, but it will update the quantity of the available insulin and the cumulative dose. The complete requirements are documented and specified in the Z language in [22], and part of the specification is provided in [23].

4 The Proposed Methodology

Our proposed methodology is based on: AsmetaL, AsmetaV, AsmetaSMV, STPA, and temporal logic. Using this methodology, we aim to guide the modeller to improve the ASM model, depending on the detection of any violations to the functional and safety requirements via the validation and verification tools, and to provide the verification tool with the STPA requirements in a formal way.

Figure 1 shows an overview of our methodology which includes the following steps: (1) Modelling the system using the AsmetaL to capture the system's requirements. (2) Validating that the AsmetaL model satisfies the functional requirements, which relate to the user needs about the system through using the AsmetaV tool. This tool allows construct particular scenarios describing the interactions between the system and its environment. The AsmetaV tool reads a scenario written by the user in Avalla, and invokes the AsmetaS tool to simulate this scenario and checks if the AsmetaL model satisfies this scenario or not. In the event that any of these scenarios are not satisfied, the AsmetaL model must be modified. (3) Eliciting safety requirements for the system via STPA. (4) Formalizing the elicited STPA safety requirements into LTL specifications using the formula in Section 5.4 that captures the four STPA timing requirements of control actions. (5) Verifying that the AsmetaL model satisfies the formulated STPA safety requirements. If any of these requirements are not satisfied, then a counter example will guide the modeller to improve the AsmetaL model.

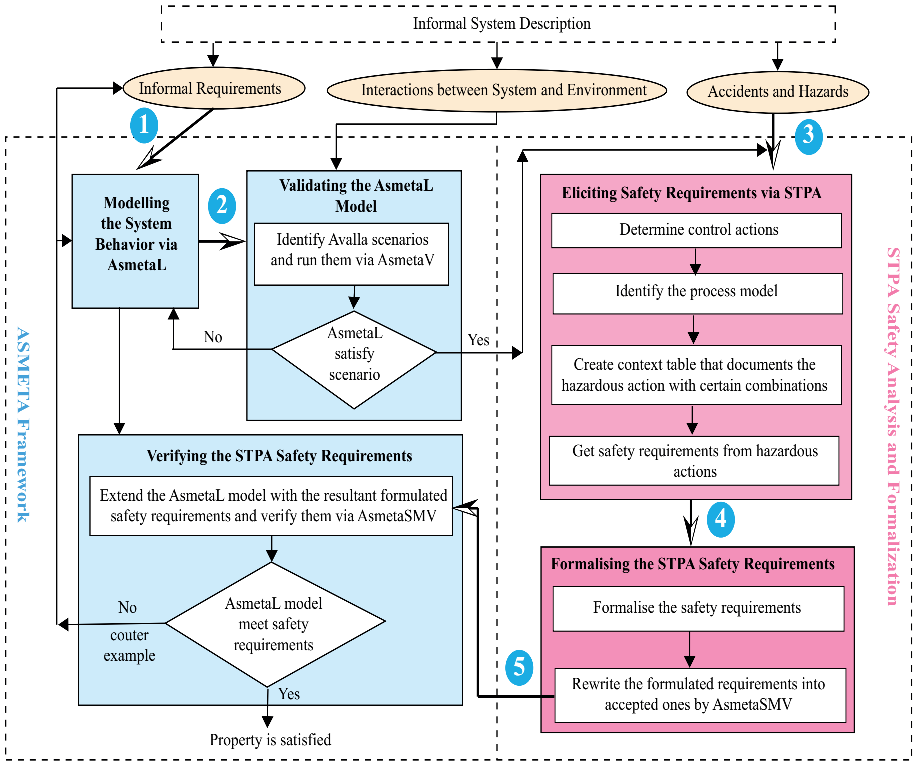

Informal System Description
Informal Requirements
Interactions between System and Environment
Accidents and Hazards
1
2
3
4
5
Modelling the System Behavior via AsmetaL
Validating the AsmetaL Model
Identify Avalla scenarios and run them via AsmetaV
AsmetaL satisfy scenario
No
Yes
Eliciting Safety Requirements via STPA
Determine control actions
Identify the process model
Create context table that documents the hazardous action with certain combinations
Get safety requirements from hazardous actions
Formalising the STPA Safety Requirements
Formalise the safety requirements
Rewrite the formulated requirements into accepted ones by AsmetaSMV
Verifying the STPA Safety Requirements
Extend the AsmetaL model with the resultant formulated safety requirements and verify them via AsmetaSMV
AsmetaL model meet safety requirements
No
couter example
Yes
Property is satisfied
ASMETA Framework
STPA Safety Analysis and Formalization

Fig. 1: The Proposed Methodology

5 Methodology Applied to IPCS Case Study

In this section, we apply our methodology to the IPCS case study.

5.1 Modeling the System Behavior via AsmetaL

In this stage, we present an abstract model or what is called a ground model, for the IPCS, written in AsmetaL[2]. The ground model is shown in Code 1. Through this model, we want to show how AsmetaL specifies the issues that have not been addressed by [22], such as switching between the system operation mode (automatic, manual, or switching off) at any state by the user, and timing details of the software controller activities: setting the cumulative dose to 0, running, and testing, which occurring at different times: 24 hours, 10 minutes, 30 seconds, respectively. We do not discuss the following in detail: the switching off and manual operating modes, and the running activity stages, which include sensing the glucose value, analysing it, calculating the insulin dose, checking the calculated dose and delivering it, since they are explained in [22, 23].

In the AsmetaL model, we first define several data types, followed by a number of functions, as discussed below. Finally a number of rules are defined to show how the IPCS works. We declared the btn monitored function that represents the operation mode of the system, which can be ON (automatic mode), MANUAL (manual mode), or OFF (switching off mode). The transition from any state into another one in the model is derived by the btn function in the r_main rule. We also declared the function cS whose value represents the state of the software controller, which can be SNS (sensing the current glucose value), ANLSCAL (analysing the current glucose value and calculating the dose), SFTCK (safety checking for the computed dose before delivering it), DLVR (delivering the dose), or TST (testing). The value of cS function, when the system works automatically (btn=ON), determines what the software controller can perform during the running activity or can exclusively perform the testing activity. Running activity, which is modelled by the r_run rule, starts when the controller state is sns. If it is so, then the controller gets the value from the sensor via the valS function, and the state of the controller is set to ANLSCAL. After calculating the dose, the value of cS function becomes SFTCK. Following safety checking for the computed dose, the cS value changes into DLVR. After delivering the dose, the running activity is finished and the cS function turns into TST to perform the testing activity.

Performing the testing activity will be through inspecting the values for the iA, ndl, rsv, fl functions in the r_test rule. The iA function is the available insulin value, which must not be less than the maximum single dose (mSD=4). The rsv function records whether the reservoir has presented (PRS) or not (NOT), and the same role for the ndl function of the needle. The fl function shows if there is a failure in any hardware unit, such as the sensor, pump, needle, or battery. Showing hardware units failure in one rather than several monitored functions

[2] All the rules for the refined model are available online at http://cgi.csc.liv.ac.uk/~hsfalsha/Insulin_Pump_Control_System.html

helps to keep the model size small. If any of the monitored functions indicate a failure, then the system must be put in a suspension state by updating the value of the controlled function spn to true, and at the same time a command must be given to the alarm through the alarmCommand function.

```
asm insulinpump
signature:
  domain Seconds subsetof Integer
  domain Dose subsetof Integer
  domain ThirtyC subsetof Integer
  domain TenC subsetof Integer
  domain InsulinRange subsetof Integer
  enum domain ControllerState={SNS,
       ANLSCAL, SFTCK, DLVR, TST}
  enum domain Button={ON, OFF, MANUAL}
  enum domain Present={NOT, PRS}
  monitored pas:Seconds->Boolean
  monitored valS:Dose
  monitored mD:Dose
  monitored fl:Boolean
  monitored ndl:Present
  monitored rsv:Present
  monitored btn:Button
  controlled cR:Dose
  controlled cS:ControllerState
  controlled spn:Boolean
  controlled alarmCommand:Boolean
  controlled manualD:Dose
  controlled iA:InsulinRange
  controlled sC30:ThirtyC
  controlled mC10:TenC
  domain Seconds={30}
  domain ThirtyC={1..20}
  domain TenC={1..144}
  domain Dose={0..35}
  domain InsulinRnge={0..100}
  function mSD=4
  function mDD=25
rule r_manualdelivering=
 if updateiACommand=false then
  if iA<=100 then //This must be modified
    if exist $md in Dose with ($md=mD and
       ($md>=1 and $md<=5)) then
       par
         manualD:=mD
         updateiACommand:=true
       endpar
    endif
  endif
 else
    par
      iA:=iA-manualD
      updateiACommand:=false
    endpar
 endif
rule r_safetycheck=
par
 cS:=DLVR
 if comD=0 then
   dD:= 0
 else
   if (comD+cD)>mDD then
     dD:=mDD-cD
   else
    if (comD+cD)<mDD then//No equality
        if (comD<=mSD)then
           dD:=comD
        else
          dD:=mSD
        endif
     endif
     .....
```

```
rule r_test=
   if (fl=true) or (rsv=NOT) or (ndl=NOT)
        or (iA<mSD) then
     par
       spn:=true
       alarmCommand:=true
     endpar
   endif
rule r_sense=
 if exist $x in Dose with $x=valS then
   par
     cR:=valS
     cS:=ANLSCAL
   endpar
 endif
rule r_run=
  switch cS
   case SNS: r_sense[]
   case ANLSCAL: cS:=SFTCK
   case SFTCK: r_safetycheck[]
   case DLVR: cS:=TST
  endswitch
rule r_autoperating=
 if spn=false then
  par
   if sC30=1 and cS!=TST then
     r_run[]
   endif
   if sC30>=1 and sC30<=19
    and cS=TST then
     if pas(30)=true then
       par
         r_test[]
         sC30:=sC30+1
       endpar
     endif
   endif
   if sC30=20 then
    if pas(30)=true then
      par
        cS:=SNS
        sC30:=1
        if mC10=144 then
          par
            mC10:=1
            cD:=0
          endpar
         else
           mC10:=mC10+1
         endif
       ...
  endpar
rule r_ceasing= .....
main rule r_Main =
 switch btn
  case ON:r_autoperating[]
  case MANUAL:r_manualdelivering[]
  case OFF:r_ceasing[]
 endswitch
default init s0:
 function sC30=1
 function mC10=1
 function cS=SNS
 function spn=false
 function iA=100
 function cD=0
 function updateiACommand=false
```

Code 1: The AsmetaL ground model for IPCS

Executing the testing and running activities are also restricted by time (30 seconds and 10 minutes). This is carried out by guards in the r_autoperating rule. As there is no tool to deal with time within the ASMETA framework, we treat time in an abstract manner. To achieve this, we use the controlled function sC30 to represent the number of 30 second cycles in 10 minutes. The maximum value for this function is 20. As the controller performs the running activity every 10 minutes and the testing activity every 30 seconds (but running and testing can not take place at the same time), one of these 20 cycles is for running and the other cycles are for testing. During the running activity, the controller sets the cumulative dose cD to 0 every 24 hours. We use the controlled function mC10 to represent the number of 10 minute cycles in 24 hours (its maximum value is 144). When this function reaches 144 and the sC30 function reaches 20, then the controller will set the cumulative dose to 0. Furthermore, we deal with increasing these functions in an abstract manner via the boolean monitored function pas(30). This means, when the pas(30)=true, some function should be increased, and at the same time, some activity should be performed. For example, if sC30=20 and 30 seconds has passed since the last update of sC30 to 20, then the running activity must be started by changing cS into SNS, sC30 becomes 1, and at the same time mC10 is checked. If it has reached 144, it is set to 1, otherwise it is increased to the next value. If 30 seconds has passed since the last update of sC30 to a value within 1-19, then the testing activity must be performed and sC30 is increased to the next value.

5.2 Validating the AsmetaL Model

This stage attempts to validate the AsmetaL model by running particular Avalla scenarios, and obtaining a fail/success outcome. The scenario describes the identifiable interactions between the system and its environment to represent informal functional requirements. In the IPCS, the interactions are represented by the current glucose value and the delivered dose. We identify 14 scenarios that correspond to the delivered dose quantity requirements. From these scenarios, we only discuss the scenario that has a fail verdict (see Code 2). Code 2 is the scenario that is written in Avalla as input to the AsmetaV tool. This scenario corresponds to the following requirements: if the cumulative dose does not exceed the maximum daily dose, and the computed dose itself is less than or equal to the maximum single dose, then the delivered dose is equal to the computed dose.

```
//setting the initial 250 states
set btn:=ON;
set valS:=22;
step
check cR=22 and cS=ANLSCAL;
step until cS=TST;
check cD=22;
set btn:=ON;
set pas(30):=true;
set fl:=false;
set rsv:=PRS;
set ndl:=PRS;
step until sC30=20;
```

```
check spn=false;
set btn:=ON;
step
check cS=SNS;
set btn:=ON;
set valS:=34;
step
check cR=34 and cS=ANLSCAL;
step
check comD=3 and comD+cD<=25
and comD<=mSD;
step
check dD=comD;
```

Code 2: The scenario that has a fail verdict

The scenario in Code 2 can be described as follows: the system is operating in automatic mode, the current glucose value is 34, the previous glucose value from 10 minutes earlier is 22, the cumulative dose is equal to 22, there is no suspension situation, the computed dose is 3 units, and the requirement that must be checked is: the delivered dose should be equal to the computed dose.

The simulation of the scenario in Code 2 is illustrated in Figure 2. In this figure, we use the following abbreviations: vdct (verdict), succ (succeed). The comD and dD functions represent the computed dose and the delivered dose, respectively. The simulation shows that we obtain the succ verdict for: the first received glucose value (22), the cumulative dose, no suspension, the second received glucose value (34), the sum of the computed dose and the cumulative dose equals the maximum daily dose (25), and the computed dose is less than the maximum single dose (4), while a fail verdict is obtained at state 283, due to missing the equality operator in the safety condition on the computed dose before delivering it (see r_safetycheck rule in Code 1). This condition checks whether the summation of the computed dose plus the cumulative dose is greater or less than the maximum daily dose, but it does not checks the equality situation ((comD(3)+cD(22))=mDD(25)). Therefore, the delivered dose is not calculated and we obtained a fail verdict. Thus, we have shown that ignoring the equality testing in the [22] specification may lead to a serious issue in the IPCS.

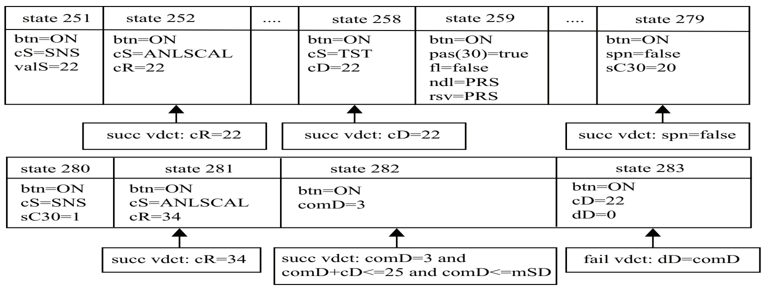


Fig. 2: Simulation of the scenario shown in Code 2

5.3 Eliciting Safety Requirements via STPA

Next we employ the STPA technique for eliciting the safety requirements of the IPCS, and it consists of the following steps:

- Indicating the main expected accidents, e.g. damage to the patient's eyes or kidneys if the required insulin dose is not taken.
- Identifying the possible hazards that can lead to the previous accidents, such as the user's unawareness of warning or failure conditions.
- Determining the actions issued by the controller that can lead to hazards in the previous step, such as: run the alarm, update the available insulin, deliver the dose.

– Identifying the process model for the controller. We define this as a set of monitored and controlled functions of the AsmetaL model. Each member of this set consists of a function name and its values, e.g., the process model that affects the run warning alarm action is: $\{\mathsf{btn}$=(ON, OFF, MANUAL), spn=(true, false), cR=($\geqslant$sMin, $<$sMin), cS=(SNS, ANLSCAL, SFTCK, DLVR), nP=(0, 1, 2, 3, 4), iA=($>4\times$mSD, $\leqslant 4\times$mSD), sCC=($>$mDD, $\leqslant$mDD)$\}$. Where the meaning of cR is the current reading of glucose, sMin is the minimum safe limit (6), mDD is the maximum daily dose, sCC is the summation of the computed and cumulative doses, and nP is the number of pulses issued by the controller to deliver the insulin.
– Evaluating the combination of function values for each control action under four contexts: 'provided', 'provided too early', 'provided too late' and 'not provided'. The evaluation process is performed through asking a question to an expert of the following form: if the controller receives a certain combination of function values, will (provide, provide too early/too late, not provide) the action in the next state by the controller lead to a hazard?. The results of the evaluation are documented in Table 1. This Table is only for the run warning alarm. The no/fun answer represents no actual hazard will happen, but there is a flaw with the system function, e.g. it is not hazardous if the alarm action is provided earlier than realizing that the current glucose is less than the minimum safe limit.
– Translate each combination that has a yes answer in the table into informal safety requirements using the phrases "must" (for 'not provided') and "must not" (for 'provided', 'provided too early', 'provided too late'. According to Table 1, we have 6 safety requirements corresponding to the 6 yes answers.

Table 1: The context table for the run alarm action with warning conditions

Process Model							Hazardous Action?			
btn	spn	cR	cS	nP	iA	sCC	Provided	Provided too early	Provided too late	Not provided
ON	false	any	DLVR	=0	⩽4mSD	any	no	no/fun	yes	yes
ON	false	any	DLVR	>0 and ⩽4	⩽4mSD	any	no/fun	no/fun	no/fun	no
ON	false	any	DLVR	=0	>4mSD	any	no/fun	no/fun	no/fun	no
ON	false	any	SNS	any	any	any	no/fun	no/fun	no/fun	no
ON	false	any	SFTCK	any	any	⩽mDD	no/fun	no/fun	no/fun	no
ON	false	any	SFTCK	any	any	>mDD	no	no/fun	yes	yes
ON	false	<sMin	ANLSCAL	any	any	any	no	no/fun	yes	yes
ON	false	⩾sMin	ANLSCAL	any	any	any	no/fun	no/fun	no/fun	no
ON	true	any	any	any	any	any	no/fun	no/fun	no/fun	no
MANUAL	any	any	any	any	any	any	no/fun	no/fun	no/fun	no
OFF	any	any	any	any	any	any	no/fun	no/fun	no/fun	no

5.4 Formalizing the STPA Safety Requirements

Here we formalize the elicited requirements. The formalization steps are:

– Determine the combination of the function values that have yes answers in the 'not provided' condition only. The purposes for this are: to ensure that

the action is provided with these combinations, and to avoid repetition, e.g. the combination that has a yes answer when the action is 'not provided' is the same as that which has a yes answer when the action is 'provided too late'. Regarding Table 1, the combinations that have been identified are: (1) btn=ON, spn=false, cS=DLVR, nP=0, and iA⩽4mSD. (2) btn=ON, spn=false, cS=SFTCK, and sCC>mDD. (3) btn=ON, spn=false, cS=ANLSCAL, and cR<sMin.
– Formulate these combinations, using the following formula:

$$\Box((com_{i1} \vee com_{i2} \vee ...com_{in}) \leftrightarrow \bigcirc(CA_i)) \tag{1}$$

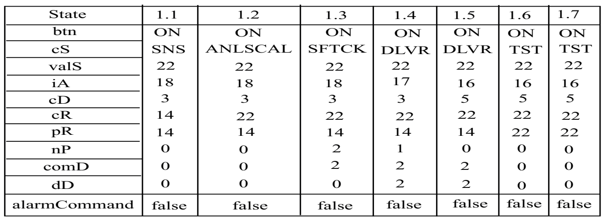
Where: CA_i is the ith control action, com_{in} is the nth combination that relates to the ith action, and the formula informally means that the control action is always provided in the next state, if and only if one of the determined combination occurs. The $\leftrightarrow$ operator puts a strong condition on providing the action, i.e. the action will not be provided with another combination or later/earlier than satisfying the determined combination. Furthermore, employing the $\leftrightarrow$ and $\vee$ operators helps to reduce the number of properties to be verified (6 safety requirements are reduced to only 1) .
– Rewriting the formulated requirements into ones accepted by the AsmetaSMV tool via its propositional and future-time connectives.

5.5 Verifying the STPA Safety Requirements

This stage is intended to verify the resultant formulated requirements against the AsmetaL model, to improve it. As we here use the ASmetaSMV tool, we rewrite the resultant formulated requirements into other ones accepted by this tool. We will present only the verification results from the AsmetaSMV tool, for the properties that are not met, as follows:

– LTLSPEC g(((**btn=on and spn=false and cS=dlvr and nP=0 and iA<=4mSD**) or (btn=ON and spn=false and cS=SFTCK and sCC>mDD) or (btn=ON and spn=false and cS=ANLSCAL and cR<sMin)) iff x(alarmCommand =true)). This property informally means that the warning run alarm action is always provided in the next state if and only if one of the warning combinations occurs. The bold font for the first combination in this property indicates that this combination is the reason for the unsatisfied property. In Figure 3 we show the failing trace for providing the run alarm action when the available insulin quantity is equal or less than 4 maximum single doses. The new abbreviation that we use in this figure is: pR (previous glucose reading). From state 1.1, onwards the system is operating under the automatic mode shown by the value ON. At state 1.1, there is no alarm action (alarmcommand=false) and the insulin quantity is 18 (iA=18). At state 1.2 the controller receives the current glucose value (cR=22) from the sensor (valS=22), and it computes the dose at state 1.3 (comD=(22(cR)-14(pR))/4(mSD)). Delivering the dose starts at state 1.4, and at state 1.5 it finishes and the available insulin becomes 16 which is equal to (4×(maxSingleDose=4)). The loop starts at state

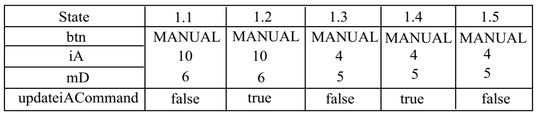
1.6 showing that the run alarm action is not provided (alarmCommand=false), when iA=16. This happens because the initial version of the AsmetaL model relies on the specification in [22], which does not consider running the alarm at cautionary situations for the available insulin quantity.



State	1.1	1.2	1.3	1.4	1.5	1.6	1.7
btn	ON	ON	ON	ON	ON	ON	ON
cS	SNS	ANLSCAL	SFTCK	DLVR	DLVR	TST	TST
valS	22	22	22	22	22	22	22
iA	18	18	18	17	16	16	16
cD	3	3	3	3	5	5	5
cR	14	22	22	22	22	22	22
pR	14	14	14	14	14	22	22
nP	0	0	2	1	0	0	0
comD	0	0	2	2	2	0	0
dD	0	0	0	2	2	0	0
alarmCommand	false	false	false	false	false	false	false

Fig. 3: Failing trace for the running alarm action when the available insulin is equal to the 4 maximum single doses

– LTLSPEC g((btn=MANUAL and iA<=100 and mD!=0) iff x (updateiACommand =true)). Where the mD is the manual dose, and the property informally means that the action of updating the available insulin according to the manual dose is always provided in the next state, if and only if the system works under the manual mode, the available insulin is less than or equal the capacity (100 units), and there is a manual dose. In Figure 4 we provide a failing trace for providing the update available insulin action when the system is in manual mode. From state 1.1 onwards, the system is in the manual mode via the value MANUAL. At state 1.1, the insulin quantity is 10 (iA=10), the manual dose is 6, and updating the available insulin action is not provided (updateiACommand=false). At state 1.2, the action is provided and has been executed at state 1.3 through changing the value of iA to 4. The loop starts at state 1.3 showing that insulin quantity is not updated, when the iA=4 and the mD=5. The loop arises from a lack of a constraint, in the [22] specification, on the available insulin before delivering the manual dose (see r_manualdelivering rule in Code 1). This constraint must check if the available insulin is equal or greater than the maximum manual dose (5) before delivering it[3].



State	1.1	1.2	1.3	1.4	1.5
btn	MANUAL	MANUAL	MANUAL	MANUAL	MANUAL
iA	10	10	4	4	4
mD	6	6	5	5	5
updateiACommand	false	true	false	true	false

Fig. 4: Failing trace for updating the available insulin action when the manual dose is greater than the available insulin

[3] All the modified specifications are available online at http://cgi.csc.liv.ac.uk/∼hsfalsha/Insulin_Pump_Correct_Version.txt

6 Evaluation

In this section, we present two comparisons. The first is comparing results of the development procedure for the IPCS, in [23] with ours. The second is between the formalization process for the STPA requirements of [3] and ours.

With regard to the development methodology for IPCS, we can compare our methodology's results with the results in [23]. Our methodology starts with specifying the system via AsmetaL, while [23] employs the Z language for specification. Our specification tries to represent the timing aspects for the system via using an abstract time representation, while the [23] specification uses the input variable clock? to obtain the current time, but it does not specify how the implementation of RUN and TEST schemas responds to this variable. In our methodology, we use the validation and verification tools to develop a safe system, whereas [23] utilizes the safety arguments method for performing manual verification. This method starts with an unsafe state, then all paths in the system code must be proven to be contradictory to this state. This method does not address the unsafe conditions determined by our methodology, which includes: (1) The patient does not take the automatic dose when the sum of the computed dose and the cumulative dose equals the maximum daily dose. (2) The system can deliver a manual dose even if it exceeds the available insulin. (3) The system does not give an alarm if the insulin reservoir is less than the sum of 4 maximum single doses. We believe that these unsafe conditions are not highlighted by other methods.

Regarding the formalization process for the STPA requirements, in [3] four types of safety requirements have been elicited and formalized, which are:

– The control action must always be provided at the next state (without being too early or too late) when a combination occurs. It has been formalized as:

$$\square\,(Com_{ij} \rightarrow \bigcirc\,(CA_i)) \tag{2}$$

Where: CA_i is the ith control action, and Com_{ij} is the jth combination that relates to the ith action. Such formula is formulated for each combination presented in a line of context table with a yes answer in the 'not provided' column.
– The control action must always be provided no later than a certain combination occurrence. This requirement is elicited according to the combination line with a yes answer in the 'provided too late' column of the context table. The corresponding safety property is formulated as:

$$\square\,((Com_{ij} \rightarrow CA_i) \wedge \neg(Com_{ij}\ U\ CA_i)) \tag{3}$$

The authors of [3] claim that this formalization of the requirement "the software controller should always (...) not provide a control action CA_i too late while the occurrences of the critical set of combinations has become previously true in the execution path.". However, a simple semantic analysis does not support their claim. Indeed, the right hand side of conjunction ensures that either (1) no action occurred, or (2) an action should be occurred, such that at some point

before that a combination should not hold, which is different from the statement of the claim.
– The control action must always be provided not earlier than the occurrence of a combination. This requirement is elicited according to the combination line with a yes answer in the 'provided too early' column of the context table. Regarding safety requirement is formulated as:

$$\square\,((CA_i \rightarrow Com_{ij}) \wedge \neg(CA_i\; U\; Com_{ij})) \tag{4}$$

The authors of [3] claim that this formalization of the requirement "a software controller should always (...) not provide control action CA_i before the occurrence of critical combinations set (...) still not become true in the execution path and that it well provides the CA_i when the combination of (...) holds. ". Now again, a simple semantic analysis does not support their claim. Furthermore, the left hand side of conjunction can not be ensured when the control action emerges from more than one combination.
– The control action must always not be provided when a combination occurs. It has been formalized in the following form:

$$\square\,(Com_{ij} \rightarrow \neg CA_i) \tag{5}$$

This formalization is formulated for each combination line with a yes answer in the 'provided' column of the context table.

In our approach all these requirements are captured by a single formula (1): $\square((com_{i1} \vee com_{i2} \vee ...com_{in}) \leftrightarrow \bigcirc(CA_i))$. In this formula, the if and only if, always, and or operators strict providing the control action only with the determined combination of the function values, not with another one. Furthermore, the biconditional (if and only if) operator ensures that when one of the determined combinations is satisfied then the control action is provided in the next state (not too late), and when the control action is provided, then one of the determined combinations must be satisfied at the previous state (the action is not provided earlier than satisfying the combination).

7 Related Work

Here, we discuss related work that uses hazard analysis techniques, formal methods, or both for analysing safety-critical systems. In [25], an integrated approach for combining the results of Fault Tree Analysis (FTA) and Failure Mode and Effects Analysis (FMEA) techniques into the requirements specification. The FTA results are the identification of combinations of component failures, while the FMEA identifies the failure modes and the minor errors that lead to component failure. That paper uses statecharts to bridge the semantics gap between the results of safety analysis and software requirements. In [21], a method for formalizing and verifying the safety requirements elicited by the FTA technique, is presented.

The safety analysis techniques that have been used for eliciting safety requirements in these papers rely mainly on component failure, and only partially on unintended interactions between system's components. Leveson [19] presents the STPA technique to identify the safety requirements for inadequate control actions that affect whole system functions and its components' behavior. According to STPA, the accidents do not simply arise from sequences of component failures, rather, they arise when the safety constraints related to the functional interactions among system components are not enforced. In [1,3] the authors propose a software safety verification methodology based on the STPA technique to elicit the safety requirements and verify them to identify software risks. First, they elicit and formalize the STPA requirements (with respect to providing and not providing actions) into LTL properties and they verify them based on an SMV manual constructed model. Next they formalize the STPA requirements (with respect to providing actions too early and too late), and they build a safe behavior model of a software controller constrained by the STPA results with UML statechart, as well as they provide an algorithm to transform the safe model into an input model of the NuSMV model checker. However, the formalization process does not reflect the requirements for too early/late actions. In our work, we reformulate the four STPA requirements ('provide', 'provide too early', 'provide too late', and 'not provide') into one formula capturing these requirements, and we exploit ASMs to model the functional behavior of the system and we do not constrained the ASM model by STPA results. We choose ASM method as it supports several characteristics, including: flexibility in modelling any algorithm at an appropriate level of abstraction, and feasibility of being used in an automatic and tool supported manner during the system development process. Furthermore, ASMs have simple and well-defined formal semantics [11].

The advantages of using formal methods for developing safety-critical systems have been shown in [14]. In [26] a Structured Object-Oriented Formal Language (SOFL) is adopted to build a formal specification for the IPCS. That paper shows that the SOFL provides an effective means to allow the developer to take a gradual process to build a formal specification for the system, but it does not show how to verify or validate the resulted specifications. In [16], timed automata is chosen to model the railyard interlocking system, and UPPAAL model checker is used to verify the safety properties of that system. On one hand, UPPAAL, unlike ASM, lacks structuring mechanism to achieve abstraction [20], and on the other hand, UPPAAL does not fully support CTL model checking [10]. In [5,8], it is shown how the ASM method serves in supporting the design, validation, and verification activities within the ASMETA framework. However, in this work the verification of safety requirements is guided only by the modeller experience, not by a safety analysis technique. Our approach utilizes the same framework (ASMETA) for developing systems, but we employ the STPA procedure for deriving the safety requirements.

8 Conclusion and Future Work

In this paper, we combine the ASM method and STPA technique in a development methodology. Our methodology shows how functional requirements validation and STPA requirements verification help us to modify the ASM specification. We have demonstrated how to capture the four STPA requirements adequately via using disjunction and if and only if operators in our formalization for the requirements. The next step will be formalizing and generalizing the STPA requirements in terms of Allen's interval algebra [4].

We have shown how the timing aspects for the IPCS have been modelled in an abstract manner. We modelled the start point of the controller activities via using two controlled functions mC10 and sC30, and we modelled the time passing since last activity by a boolean monitored function pas. This abstract handling specifies when the activity starts but it ignores dealing with durative action, while a certain activity is performed, e.g. run alarm for 10 seconds during running activity. In the future, we hope to use improved abstractions to deal with timing aspects.

In the specification analysis presented here, we did not consider the static analysis for the completeness and consistency properties. In the future work, we are going to address this by applying the AsmetaMA tool [7] to the specification.

To further our methodology we intend to design an algorithm to automate the part of eliciting STPA requirements. Although an automatic tool has been proposed to achieve this [2], it seems only to work for up to 6 variables in a process model for the software controller. Hence, we plan to make the integration between ASM and STPA automatic, without the need for user input.

Acknowledgments

We gratefully acknowledge Dr. Paolo Arcaini for his advice on ASMETA framework.

References

1. Abdulkhaleq, A., Wagner, S.: Integrated safety analysis using systems-theoretic process analysis and software model checking. In: International Conference on Computer Safety, Reliability, and Security. pp. 121–134. Springer (2015)
2. Abdulkhaleq, A., Wagner, S.: XSTAMPP: An extensible STAMP platform as tool support for safety engineering. In: 2015 STAMP Workshop, MIT, Boston, USA. Stuttgart University (2015)
3. Abdulkhaleq, A., Wagner, S.: A Systematic and Semi-Automatic Safety-Based Test Case Generation Approach Based on Systems-Theoretic Process Analysis. arXiv preprint arXiv:1612.03103 (2016)
4. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM 26(11), 832–843 (1983)
5. Arcaini, P., Bonfanti, S., Gargantini, A., Mashkoor, A., Riccobene, E.: Formal validation and verification of a medical software critical component. In: Formal Methods and Models for Codesign (MEMOCODE), 2015 ACM/IEEE International Conference on. pp. 80–89. IEEE (2015)
6. Arcaini, P., Gargantini, A., Riccobene, E.: AsmetaSMV: a way to link high-level ASM models to low-level NuSMV specifications. In: International Conference on Abstract State Machines, Alloy, B and Z. pp. 61–74. Springer (2010)

7. Arcaini, P., Gargantini, A., Riccobene, E.: Automatic Review of Abstract State Machines by Meta-Property Verification. In: NASA Formal Methods Symposium. pp. 4–13. NASA (2010)
8. Arcaini, P., Gargantini, A., Riccobene, E.: Modeling and analyzing using ASMs: The landing gear system case study. In: International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z. pp. 36–51. Springer (2014)
9. Arcaini, P., Gargantini, A., Riccobene, E., Scandurra, P.: A model-driven process for engineering a toolset for a formal method. Software: Practice and Experience 41(2), 155–166 (2011)
10. Behrmann, G., David, A., Larsen, K.: A tutorial on UPPAAL. Formal methods for the design of real-time systems pp. 33–35 (2004)
11. Börger, E., Stärk, R.: Abstract state machines: a method for high-level system design and analysis. Springer Science & Business Media (2012)
12. Carioni, A., Gargantini, A., Riccobene, E., Scandurra, P.: A scenario-based validation language for ASMs. In: International Conference on Abstract State Machines, B and Z. pp. 71–84. Springer (2008)
13. Gargantini, A., Riccobene, E., Scandurra, P.: A metamodel-based language and a simulation engine for abstract state machines. J. UCS 14(12), 1949–1983 (2008)
14. Gerhart, S., Craigen, D., Ralston, T.: Experience with formal methods in critical systems. IEEE Software 11(1), 21–28 (1994)
15. Gurevich, Y., et al.: Evolving algebras 1993: Lipari guide. Specification and validation methods pp. 9–36 (1995)
16. Khan, U., Ahmad, J., Saeed, T., Mirza, S.H.: On the real time modeling of interlocking system of passenger lines of Rawalpindi Cantt train station. Complex Adaptive Systems Modeling 4(1), 17 (2016)
17. Leveson, N.: A new accident model for engineering safer systems. Safety science 42(4), 237–270 (2004)
18. Leveson, N., Thomas, J.: An STPA primer. Cambridge, MA (2013)
19. Leveson, N.G.: A new approach to hazard analysis for complex systems. In: International Conference of the System Safety Society (2003)
20. Ouimet, M., Berteau, G., Lundqvist, K.: Modeling an Electronic Throttle Controller Using the Timed Abstract State Machine Language and Toolset. In: MoDELS Workshops. vol. 4364, pp. 32–41. Springer (2006)
21. Santiago, I.B., Faure, J.M.: From Fault Tree Analysis to Model Checking of Logic Controllers. IFAC Proceedings Volumes 38(1), 86–91 (2005)
22. Sommerville, I.: Insulin Pump – Z schemas, http://iansommerville.com/software-engineering-book/files/2014/07/Insulin-Pump-Z-schemas.pdf
23. Sommerville, I.: Software Engineering. Addison Wesley, 9th edn. (2010)
24. Thomas, J.: Extending and Automating a Systems-Theoretic Hazard Analysis for Requirements Generation and Analysis. Ph.D. thesis, Massachusetts Institute of Technology (2013)
25. Troubitsyna, E.: Elicitation and Specification of Safety Requirements. In: Systems, 2008. ICONS 08. Third International Conference on. pp. 202–207. IEEE (2008)
26. Wang, J., Liu, S., Qi, Y., Hou, D.: Developing an insulin pump system using the SOFL method. In: Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific. pp. 334–341. IEEE (2007)