VERIFYING THERAPY SAFETY INTERLOCK SYSTEM WITH SPIN

Motlatsi Seotsanyana[1], Jaco Geldenhuys[2]

[1]Council for Scientific and Industrial Research, Mobile Intelligent Autonomous Systems,
Modelling and Digital Science Dept., P.O. Box 395,
Pretoria 0001, SOUTH AFRICA.

[2]Department of Mathematical Sciences, Computer Science Division,
University of Stellenbosch Private Bag X1,
7602 Matieland, SOUTH AFRICA

Abstract — The ever-increasing reliance of society on computers has led to a need for highly reliable systems. Computer systems perform critical functions in a number of areas ranging from online transaction processing (such as banking systems) to embedded environments (such as nuclear power plant safety control systems). Their development requires a higher level of attention than many others, and the use formal methods is one way to ensure that they are as correct as possible. This paper reports on the successful use of model checking in the design and verification of the Safety Interlock System (SIS) at iThemba LABS. SIS is part of proton therapy control system (TCS) and its main task is to monitor and evaluate the safety conditions in the TCS as a whole. It looks after other dynamic systems and electronic units which may join and leave the whole TCS either predictably or unpredictably, and oversees their distributed interactions. An appropriate design pattern in this kind of setup is the Observer, also known as Publish-Subscribe or Dependents. Although not new, the Observer pattern is receiving increasing interest because of its usefulness in event-driven systems. It encompasses a well-established communications paradigm that allows any number of subjects (publishers) to communicate with any number of observers (subscribers) asynchronously and anonymously via event channels. The study focuses on the development of an abstract communication model between the SIS and the systems it monitors. A number of important correctness properties are verified with the SPIN model checker.

1. Introduction

The ever-increasing reliance of society on computer systems has led to a need for highly reliable software and hardware systems. There are a number of areas where computers perform critical functions ranging from on-line transaction processing systems, such as banking systems and airline reservation systems, to embedded computer systems, such as manufacturing systems, automobiles, air traffic and space vehicle control systems, nuclear power plant safety control systems, medical and military applications. In these areas the failure of a computer system may result in just mere inconvenience, economic disruption, loss of time or even loss of life. It is clear that the development of such systems requires a higher level of attention than any other type of system and it is also clear that the need for these kind of systems will continue to grow. The appropriate approach in this situation is known as formal methods such as model checking.

Formal methods refer to the use of mathematical techniques for the specification, development and verification of software and hardware systems. A number of success stories about the use of formal methods have been reported in the literature [1, 3, 4]. Model checking is an automatic verification technique for finite state concurrent systems, and this paper reports on the successful use of model checking in the design and verification of the Safety Interlock System (SIS) at iThemba LABS. Model checking is one formal method that checks whether a model of a software/hardware system is correct. The system is described using a formal notation similar to a programming language, and the desired correctness properties are expressed as formulas in a linear temporal logic (LTL) [7]. A model checking tool determines whether or not the system satisfies the properties and, if not, exactly how the properties are violated. This valuable feedback can then be used to improve the system so that the violations are eliminated. This paper describes how model checking was used in the design and verification of the Safety Interlock System at iThemba LABS.

The iThemba LABS (http://www.tlabs.ac.za) is a multidisciplinary research facility involved in basic and applied research using particle beams, particle radiotherapy for the treatment of cancer, the supply of accelerated-produced radioactive isotopes for nuclear medicine and research, and similar activities. Currently, iThemba is engaged in a new project called “Second Beam Line Project” (2BL), that involves an additional

beam line for the treatment of cancer using protons, and the development of a system referred to as the Therapy Control System (TCS). An important feature of the TCS is that independent parts are interlocked (i.e., synchronized); the SIS is responsible for ensuring that the parts work together safely and smoothly, and is therefore central to the operation of the TCS.

As one might expect, the TCS and its specification is large and complex. Because it is a concurrent system, it is easy to miss subtle timing errors using traditional testing techniques, and model checking, which exhaustively investigates all possible orderings of events, offers a valuable supplement. This paper describes the use of the SPIN model checker [6] in the design of the SIS. The design is also enhanced through the use of a well-established design pattern known as an Observer (sometimes called Publish-Subscribe). This pattern can play a powerful role in event-driven systems, and is enjoying renewed interest, also in the model checking community [2, 5, 8, 9].

The rest of this paper is organised as follows: Section 2 focuses on the specification of the SIS in relation to other TCS systems. Section 3 gives an overview of a SPIN specification of the system, and Section 4 describes some properties of interest and presents the results of their verification. Finally, Section 5 concludes and discusses some future work.

2. Safety Interlock System

The main task of the safety interlock system (SIS) is to monitor and evaluate the safety conditions in the system as a whole, using inputs from the therapy safety bus (TSB) and also hard wires from all over the TCS. The components of the system consist of a comprehensive arrangement of relays, switches and transistor-controlled circuits which ensure correct and safe operational conditions at all times. There are two categories of interlocks: personnel interlocks and machine interlocks that ensure safe operation of the whole TCS. These two classes are not completely independent; personnel interlocks provide safe and accurate delivery of patient treatment prescription and protect the radiotherapy staff and the general public, while machine interlocks ensure that the machines are operated safely and prevent damage to equipment. In this paper, both personnel and machine interlocks are referred to as input interlocks and are classified as either discrete or non-discrete. Discrete interlocks communicate with the SIS via hard (physical) wires while non-discrete interlocks communicate indirectly through the TSB. Indirect communication refers to the fact that the exact detail of which interlock failed is not known to the SIS. The TSB status only indicates that some interlock has failed.

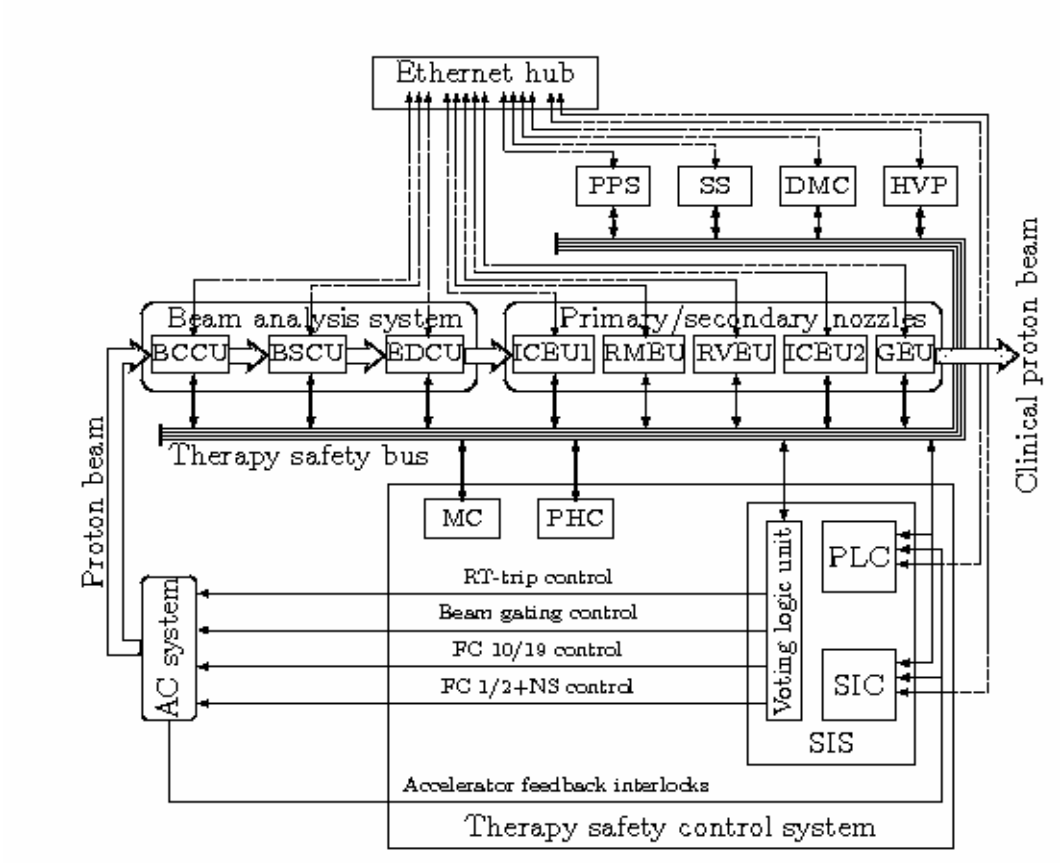

Figure 1: Architecture of the 2BL

Figure 1 shows an overview of the system and some of the electronic units and subsystems involved. The thick, dotted arrow that runs from left to right in the middle of the figure shows the path of the physical beam. It originates in the accelerator control (AC) system and passes through the beam analysis and control system (through the beam current, beam steering, and energy degrader controller units), and through the primary and secondary treatment nozzles (through the two ionisation chambers, range modulator, range verifier, and general electronic units) before emerging to treat the patient. At the top right is the patient positioning system, the supervisory system, the dose monitoring system, and the high voltage power supply unit.

The heart of the system is the therapy safety control system, shown in the bottom right of Figure 1. It consists of the safety interlock system and the master and physics consoles (described in Section 2.3). All these parts are connected in two ways: by a 13-line bus and an Ethernet network. The main purpose of the TSB is to provide a fast means by which any system of the TCS can communicate its functional and hardware failures to the rest of the system and, most importantly, to the SIS. Sections 2.1 and 2.2 present a detailed description of the discrete and non-discrete interlocks, respectively, and Section 2.3 explains two types of consoles that are used to manipulate the interlocks. Sections 2.4 — 2.6 discuss systems which directly interact with the SIS.

2.1 Non-Discrete Interlocks

Electronic units in TCS indirectly communicate system status, requests and failures to the SIS via the TSB. TSB consists of a number of discrete wires which will henceforth be called TSB lines and each of these lines represents a specific status or request in the TCS. The TSB lines CONSOLE-ON, PRIMARY-NOZZLE, SECONDARY-NOZZLE, and BEAM-ON communicate current system configuration status to the SIS while the TSB lines SABUS and HIGH-VOLTAGE-PSU communicate interlock failures. The TBS lines RF-TRIP OFF, BEAM DEFLECTOR OFF, FC 1/2 & SHUTTER OUT, FC 10/19 OUT, PHYSICS MODE, and TEST MODE are used to communicate requests to and from the SIS. A current source is attached to each of these lines, so a TSB line has a value true if the current is flowing through the line, and a value false, otherwise. The systems and components that communicate their requests and status through continuous (i.e., Non-Discrete) interlocks are beam current controller unit (BCCU), beam steering controller unit (BSCU), energy degrader controller unit (EDCU), ionization chambers electronic unit 1 (ICEU1), range modulator electronic unit (RMEU), range verifier electronic unit (RVEU), ionization chambers electronic unit 2 (ICEU2), general electronic unit (GEU), high voltage power supply unit (HVP), patient positioning system (PPS) and dose monitoring system (DMC). All of these units can change the status of the line, but the TSB does not record the details of exactly which interlock failed, and so it is not possible for the SIS to identify the exact source of failure. If the SIS detects a failure from any of the systems and/or electronic units, the systems can send the interlock statuses via the LAN to the supervisory system for displaying purposes.

2.2 Discrete Interlocks

Discrete interlocks are directly connected to the SIS via hard wires. The components and systems that communicate in this way include general interlocks (GI), primary and secondary treatment nozzles and beam line interlocks (TNBL), accelerator group interlocks (AGI), and room clearance interlock (RCI). All these categories of interlocks are manipulated by the TCS — they will henceforth be called TCS interlocks — except the AG category which is manipulated by the accelerator control systems. The interlocks can either have a value true or false and can also be overridden depending on the mode in which the system is operating in. The status of the discrete interlocks is transmitted via LAN to the supervisory system for displaying purposes.

2.3 Master and Physics Consoles

In addition to the SIS, the TCS also includes master and physics consoles. The purpose of the master console (MC) is to provide the user (radiation therapist or physicist) with a simple interface to select between physics, test and clinical mode, to start and stop the beam, and to perform an emergency stop at any time. The physics console (PHC) is physically separated from the MC but it performs the same functions as the MC. Only one of the consoles must be active at any given time. The master console also provides the radiation therapist with feedback of the beam characteristics and dose delivery to the patient. The console also indicates the status of the TSB and the room clearance system status. The room clearance system is part of the therapy safety control system which ensures the safe evacuation.

2.4 Supervisory system

Just like the master and the physics consoles, the purpose of the supervisory system (SS) is to provide the user (radiation therapist or physicist) with the following: (1) a simple interface to select between physics, test and

clinical mode, (2) to select between primary and secondary treatment nozzles, and (3) to start or stop the control system. In addition, it can change the status of the TSB lines just like any other system attached to the bus lines (see Figure 1). The SS is also responsible for displaying the status of the TSB lines, room clearance system interlocks, TCS interlocks and accelerator control system interlocks.

2.5 Room clearance system

There is a fixed procedure that should be followed to make sure that the treatment room is armed (i.e., ready for patient treatment). The room clearance system (RCS) is responsible for ensuring that the following actions are carried out: (1) There are eight emergency buttons that are distributed around the treatment vault. All eight buttons must be in the "normal" position for the treatment to begin. If at any point in time one of these buttons is pressed, the room must revert to a safe condition, that is, neither armed nor primed. The term "prime" refers to the intermediate preparations of the room, e.g., if an access door is ready to be closed. (2) There is a gate placed across the entrance to the basement of the treatment room and this gate must be closed for the treatment to begin. If at any time this gate is open, the room must return to the safe condition. (3) There is a door to the annex off the maze and this door must be closed for the treatment to begin. If at any time this door is open, the treatment room must revert to the safe condition. (4) There are two access doors in the partition on either side of the beam-line. These doors must be primed and closed 10 seconds before the room is primed. When any of these doors is open, its circuit is open and when it is closed its circuit is closed. If any of these doors is not primed and closed within 10 seconds the treatment vault must return to the safe condition. (5) If all the conditions from 1 to 4 are satisfied, that is, a circuit for each device is closed, then the room may be primed for evacuation. The priming is done by pressing an exit button in the treatment room. (6) Once the room has been primed, the operators have 40 seconds to leave the room and close the boom gate at the maze exit. The boom gate must be primed and closed for the treatment to begin and if at any time the boom gate is open the room must be neither primed nor armed. If the boom gate is not closed within 40 seconds, the room must return to the safe condition after which the room may be primed once again. (7) If the boom gate is primed within the allowed 40 seconds, the room may be armed at any time after the closing of the boom gate. This means that an Ok signal is sent to the SIS.

2.6 Accelerator control system

There are two types of beam stop devices: a Faraday cup, which is a cup shaped piece of copper and a neutron shutter, which is a steel cylinder for shielding radiation. Both kinds of devices have two micro-switches, associated with each extreme movement of the device, which are used to detect whether the device is in the beam line or not. There are five of these beam-stop devices: (1) Faraday Cup 1: can be in or out of the beam line and it is located at the end of the beam line, that is, it is the last beam stop device just before the patient. (2) Faraday Cup 2: can be in or out of the beam line and it is located between the cyclotron and the neutron shutter. (3) Faraday Cup 10: can be in or out of the beam line and it is located next to the injector cyclotron. (4) Faraday Cup 19: can be in or out of the beam line and it is located next to the main cyclotron. (5) Neutron Shutter: can be in or out of the beam line and it is located in the wall of the treatment room, that is, between Faraday cups 1 and 2. The accelerator control (AC) system is responsible for extracting and inserting these devices out of and into the beam line in response to commands. The system produces ten feedback outputs to the SIS to indicate whether a device is in or out.

3. Model Checking with SPIN

SPIN (Simple Promela INterpreter) is a model checking tool developed at AT&T's Bell Labs by Gerard Holzmann during the early 1990's. It is still widely used and there is a yearly workshop organized around it. While some other model checkers are more suited to the verification of hardware (on the gate and circuit level), SPIN focuses on software verification of operating systems, data communication protocols, etc. As mentioned in the introduction, a model checker performs an exhaustive search of a system's executions, but SPIN includes a number of additional features such as guided and random simulations, and static analysis of C programs. The tool is easy to start to use, and its operation is largely automatic: given a system description and correctness property as input, it either outputs the "all-clear" if the property holds, or generates an example of a violating execution path if it fails to hold. A graphical user interface allows users to inspect the behaviour of models and follow violating executions interactively. A review of internal workings of SPIN is beyond the scope of this paper. Interested readers are referred to Holzmann's book [6] and the tool website (http://spinroot.com).

3.1 An overview of PROMELA

The input language of the SPIN tool is called PROMELA (PROtocol MEta-LAnguage). It is mostly based on the C programming language and Dijkstra's guarded command language. The basic elements are local and global variables, processes, and communication channels, but there is also some provision made for structured data, functions, and macros. (A recent extension of the tool also allows users to embed C code directly in models.) Inside processes, the behaviour of the system is described using assignments and if- and while-statements, and processes can communicate over synchronous and asynchronous bounded-capacity channels using send (!) and receive (?) operators. The shared-memory paradigm is supported through global variables.

All in all, the semantics of PROMELA is close to that of a regular programming language, apart from a few subtle differences in semantics. In particular, an important and powerful feature of PROMELA is non-deterministic choice: this allows users to abstract away irrelevant details and to concentrate on the essence of a large system.

3.2 PROMELA Specification of the SIS

The observer design pattern defines a one-to-many dependency between interacting objects so that when one object (the subject) changes state, all its dependents (the observers) are notified and updated automatically. Figure 2 depicts the interaction of the SIS with the TCS's systems and other electronic units, all based on the observer notification services. We have, in fact, built several PROMELA models of the SIS to capture different aspects of the system. (We omit the complete model source code due to space constraints and provide an overview of the source code shown in Table 1 to Table 3.) In the particular model discussed here, the focus is on the exchange of event messages between the SIS and other components. Its overall structure is shown in Figure 2.

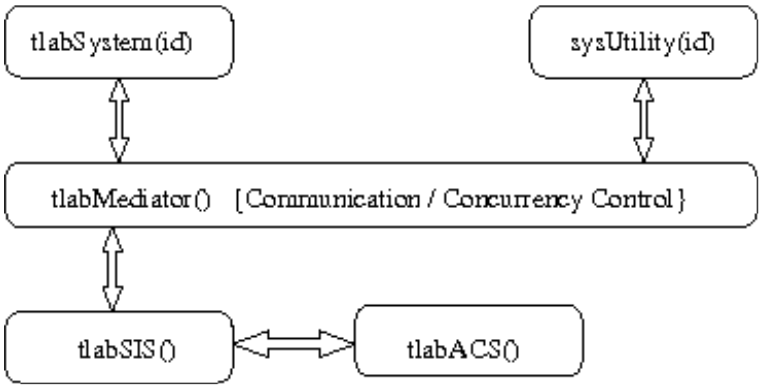

Figure 2: PROMALA Specification

The tlabSystem component represents TCS systems and electronic units. It connects to the TSB and TCS lines by sending a message *register* to the tlabsMediator component and unconnect from these lines through an *unregister* message. It also changes lines to either true or false and emits two different messages (*update* and *notify*) to the tlabMediator component. The sysUtility component takes care of administrative work for the tlabSystem components, including receiving the *display* messages from other components. The tlabMediator component encapsulates the interaction between components and promotes loose coupling of components. It allows components to refer to each other explicitly, and sends and receives messages to and from the components except tlabACS component. The tlabACS component is not part of the TCS and communicates with the tlabSIS component directly. It receives control commands from the tlabSIS component and sends feedback information back to it.

tlabSystem	**sysUtility**
1 active[3] proctype tlabSystem(byte id){ 2 bool registered[numInterlockTypes] = false, 3 interlock[numInterlockTypes]; 4 do 5 :: id != 1 && registered[tsb] -> 6 system2mediator ! readinterlock(id, tsb);	1 active proctype sysUtility(byte id){ 2 byte fid; 3 do 4 :: med2sysUtility[id] ? notify(fid) -> 5 :: med2sysUtility[id] ? display(fid) -> 6 :: ...

```
7       mediator2system[id] ? interlockread(tsb)
8     ....
9   :: id == 1 && (!interlock[rcs] && registered[rcs]) ->
10      interlock[rcs] = true;
11      system2mediator ! update(id,rcs);
12      system2mediator ! display(0,rcs)
13  :: id == 1 && (interlock[rcs] && registered[rcs]) ->
14      interlock[rcs] = false;
15      system2mediator ! update(id,rcs);
16      system2mediator ! display(0,rcs)
17  od
18 }
```

```
7   od
8 }
```

Table 1: the tlabSystem and sysUtility components

tlabMediator

```
1 active proctype tlabMediator(){
2     tlabSystems system[numSystems];
3     byte id, sysid,fid;
4    do
5    :: system2mediator?register(id,fid) ->
6       if
7       :: (id == 0) ->
8       :: (id == 1) ->
9       :: (id == 2) ->
10      fi;
11      system[id].registered[fid] = true
12   :: system2mediator?unregister(id,fid) ->
13      if
14      :: (id == 0) ->
15      :: (id == 1) ->
16      :: (id == 2) ->
17      fi;
18      system[id].registered[fid] = false
19   :: system2mediator ? display(id,fid) ->
20      if
21      :: (id == 0) ->
22      :: (id == 1) ->
23      :: (id == 2) ->
24      fi;
25      med2sysUtility[id] ! display(fid)
26    :: system2mediator ? update(id,fid) ->
27       if
28       :: (id == 0) ->
29       :: (id == 1) ->
30       :: (id == 2) ->
31       fi;
32      mediator2sis ! notify(fid);
33      sis2mediator ? notified;
34      sysid = 0;
35      do
36      :: (sysid < numSystems) ->
37         if
38         :: (sysid != id && fid != rcs) ->
39            med2sysUtility[sysid]!notify(fid) ->
40            if
41            :: (id == 0) ->
42            :: (id == 2) ->
43            fi
44         :: else -> skip
45         fi;
46         sysid++
47      :: else -> break
48      od
49    :: system2mediator ? readinterlock(id,fid) ->
50       mediator2system[id] ! interlockread(fid);
51    :: ...
52   od
53 }
```

Table 2: the tlabMediator component

tlabSIS

```
1 active proctype tlabSIS(){
2   byte fid;
3   do
4   :: mediator2sis ? notify(fid) ->
5      ...
6      sis2acc ! commands(ctrl);
7      acc2sis ? feedback(fdb);
8      sis2mediator ! notified
9   od
10 }
```

tlabACS

```
1 active proctype tlabACS(){
2   do
3   :: sis2acc ? commands(ctrl) ->
4      ...
5      acc2sis ! feedback(fdb)
6   od
7 }
```

Table 3: the tlabSIS and tlabACS components

4. Verification

The results reported in this section were obtained with SPIN version 5.2.0 running on a laptop with 3 gigabytes of RAM and a 2.0 GHz CPU. Execution times reflect the work performed by the model checker and are independent of the machine load. There are two distinct categories of verification runs.

Systems	States explored	Memory (MB)	Time (sec)
2	23,229	187.857	0.29
3	3,884,287	190.298	14.70
4	35,022,775	457.284	209.53

Table 4: Results for a full search of the state space

The first category focuses on a full search of the state space for deadlocks, i.e., situations where the TCS reaches a state where it "hangs". Table 4 shows the outcome of this deadlock search in the TCS model for two, three, and four systems. That is, the numbers "2", "3", and "4" in the first column of the table refer to the number of the tlabSystem components in the PROMELA model. These runs tell us that the TCS model is free from deadlocks. The values in the second column (labeled "States explored") reflects the number of unique situations that the model checker investigates. As reflected in the table, this value grows exponentially as the size of the model grows — a phenomenon known as "state explosion". In fact, this is the main challenge that prevents the model checking of very large systems.

The second category of verification runs deals with correctness properties that are specific to the TCS. They include: (1) The SIS is notified for every update of either the TSB or TCS lines. (2) The TCS systems and electronic units will not be denied reading TSB or TCS lines forever, if they are connected to them. (3) After the SIS has sent control commands to the accelerator control system, it will eventually receive feedback information about the status of the beam devices. (4) A system does not update interlocks if it is not registered (i.e., not connected to either TCS or TBS lines). (5) A system does not receive a notification if it is not registered (i.e., connected) to either TSB or TCS lines. And (6) the supervisory system does not send a display message if it changes either the TSB or TCS lines — it just displays the information. But, other systems and electronic units registered for the interlocks are notified of the change. The rest of this section discusses the verification of the first four of the above properties in more detail. These properties address the core functionality of the SIS. For example, the first property can be adjusted to verify that, for every system that modifies either the TSB or TCS lines, a notification messages is eventually sent to the SIS.

Properties	Depth reached	States stored	Memory (MB)	Time (sec)
P1	1,801,371	15,273,580	193.754	141.08
P2	5,138,511	9,903,514	849.675	85.65
P3	4,736,991	10,314,812	815.398	99.70
P4	6,702,307	14,767,202	808.580	124.92

Table 5: Results for correctness properties

P1: Line notification

The first property states that the SIS is notified for every update of either the TSB or TCS lines by the TCS systems and electronic units connected to the lines as shown in Figure 1. In this case, supervisory system is used to verify the property. The following labels are added to the PROMELA specification: SSRegistered (SS is connected to the lines), SSUnregistered (SS is unconnected to the lines), ASystemUpdate (the SS updates the lines), and SISNotified (the SIS is notified of the change). The LTL formula is then as follows:

```
[]!(tlabMediator[1]@SSRegistered && (<> tlabMediator[1]@ASystemUpdate) &&
(!tlabMediator[1]@SSUnregistered U (tlabMediator[1]@ASystemUpdate &&
[]!stlabMediator[1]@SISNotified)))
```

SPIN is run to verify this property and the results are shown in Table 5, in the first row. The results show that even though it consumed more time and explored more states than other two properties, the depth reached and memory consumed were less than other two properties.

P2: No infinite read delays

The second property states that whenever a system wants to read the TSB and TCS lines, it will eventually get an opportunity to do so. Just like in the first property, the specification is augmented with labels readTSB and TSBread in the tlabMediator component. These labels are not added to the tlabSystem component after the

request to read and after the system receives the response, since (in this case) tlabMediator mimics the behaviour of connected wires (not communication over a computer network). The LTL specification is:

[](tlabMediator[1]@readTSB -> <>tlabMediator[1]@TSBread)

As in the first case, this property is verified with SPIN against the PROMELA specification. The results summarised in Table 5 show that this property consumed more memory than the other three properties (i.e., P1, P3, and P4).

P3: Eventual feedback
The third property addresses another important aspect of the SIS' tasks. For every notification to the tlabSIS component, the tlabSIS sends control commands to the accelerator control system and in response, the tlabACS will eventually receive feedback about the status of the beam devices. To verify this property, the labels are added to the specification. The label NotifySIS is added after the tlabMediator has sent a notify message to the tlabSIS, the label SISsendCtrl is added after the tlabSIS has sent a control message to the tlabACS, and the label SISgetFeedback is added after the tlabSIS receives a feedback message. The LTL specification is:

[]((tlabMediator[1]@NotifySIS && tlabSIS[8]@SISsendCtrl) ->
<> tlabSIS[8]@SISgetFeedback)

The property is verified with SPIN and the results depicted in Table 5 show that the property produces almost the same results as the second property.

P4: An event implies another event
The fourth property addresses another important invariant requirement of the SIS' tasks. The property states that if the tlabSystem is not connected to the TCS or TSB lines, it will not update the lines. Thats is:

tlabSystem[4]@SYSUnregister -> ! tlabMediator[1]@SYSUpdate,

where the label SYSUnregister is added after the tlabSystem component has sent unregister message and the label SYSUpdate is added after the tlabMediator has received an update message. However, this property is not valid when verified with SPIN due to asynchronous behaviour of the components, instead the following stability property is verified**;**

[] ((tlabSystem[4]@SYSUnregister && tlabMediator[1]@SYSUpdate &&
<> ! tlabMediator[1]@SYSUpdate) -> (tlabMediator[1]@SYSUpdate U [] ! tlabMediator[1]@SYSUpdate))

As in the first three properties, the property is verified with SPIN and the results are shown in Table 1, in the last row. The results show that the property consumed almost the same memory as the second and the third properties. However the states stored and the time taken to verify this property are close to that of property one, while the depth reached is just above that one of property two.

5. Conclusion and Future Work

This paper presents ongoing work on the design and development of the TCS system. It described an overview of the TCS and its components, and specifically the SIS and its interaction with the rest of the system. PROMELA models of the system were built and then formally verified with the SPIN tool. Our experience with basing the models on the observer pattern has been positive, and no errors in the TCS models were found.

It should be pointed out that model checking relies on several key assumptions: that the model is accurate, that all important properties are correctly formulated, and that no important correctness property is omitted. However, this is true of all approaches to validation, including testing. The value of model checking is not limited to the fact that it can formally "prove" that the properties hold; often just building the formal model already provides valuable insights for the system developers about how their system operates. In this regard, the maturity of the iThemba LABS specification of the TCS was particularly helpful in constructing our models.

Since the underlying structure of the TCS has been modelled and verified successfully, in the future more detailed modelling and verification of each component can be addressed. In particular, we would like to focus on the details of the SIS, the supervisory system, and the room clearance system.

6. Acknowledgements

The research conducted and reported on in this paper was funded by the Council for Scientific and Industrial Research (CSIR), South Africa, The authors would like to thank Lebelo Serutla at iThemba LABS with his advice and interpretation of the therapy safety interlock system specification.

7. References

[1] Bowen J. P. and Hinchey M.G. (1997) The use of industrial-strength formal methods. IEEE Computer Society Press, pages 332–337, 13–15 August.

[2] Mauro Caporuscio, Paola Inverardi, and Patrizio Pelliccione. (2002) Formal analysis of clients mobility in the siena publish/subscribe middleware.

[3] E. M. Clarke and R. P. Kurshan. Computer-aided verification. IEEE Spectrum, 33:61–67, 1996. Invited article.

[4] Edmund M. Clarke and Jeannette M. Wing. (1996) Formal methods: State of the art and future directions. ACM Computing Surveys, 28(4):626–643.

[5] William Deng, Matthew B. Dwyer, John Hatcliff, Georg Jung, Robby, Gurdip Singh, and Gurdip Singh. (2003) Model-checking middleware-based event-driven real-time embedded software. In InProceedings of the 1st International Symposium on Formal Methods for Components and Objects, pages 154–181.

[6] Gerard J. Holzmann. (2004) The Spin Model Checker: Primer and Reference Manual. Addison-Wesley, Lucent Technilogies Inc. Bell Laboratories.

[7] Zohar Manna and Amir Pnueli. (1992) The Temporal Logic of Reactive and Concurrent Systems. Springer-Verlag New York, Inc., New York, NY, USA.

[8] David Garlan Serge, Serge Khersonsky, and Jung Soo Kim. (2002) Model checking Publish-Subscribe systems. pages 166–180.

[9] Maurice H. tek Beek, Mieke Massink, Diego Latella, Stefania Gnesi, Alessandro Forghieri, and Maurizio Sebastianis. (2005) Model Checking Publish/Subscribe Notification for thinkteam®, Proceedings of the Ninth International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2004), pages 275–294.