

# Integrating Safety and Security Requirements into Design of an Embedded System

Saad Zafar  
Griffith University  
s.zafar@griffith.edu.au

R. G. Dromey  
Griffith University  
g.dromey@griffith.edu.au

## Abstract

*Most modern embedded systems are now required to satisfy seemingly divergent critical properties like safety and security. It is therefore becoming increasingly important that any systems development methodology employed should support modeling of system requirements in a manner that it facilitates validation and verification of such critical properties. In the paper we present the result of applying the Genetic Software Engineering (GSE) method to design an ambulatory infusion pump (AIP) which must satisfy a number of safety and security properties. The safety and security requirements are integrated with the rest of the systems requirements in the form of integrated behavior tree (IBT), which is systematically refined into a design behavior tree (DBT). The integrated behavioral view of the requirements provides a platform for requirements conflict resolution, defect detection and requirements validation. The formal semantics of the behavior tree (BT) notation, used to specify the requirements, makes formal verification of critical properties in the final design possible.*

## 1. Introduction

Modern embedded systems are required to satisfy different critical properties like safety and security [1]. Software based controllers in automobiles, airplanes and medical devices are examples of such systems. As reliance on such systems is increasing, the consequences of failures are becoming more serious [2]. Therefore, developers of these systems are now required to provide an acceptable level of assurance regarding the critical services they provide [3, 4].

In the past, it has been a common practice to perform safety and security requirements engineering in isolation from each other and in isolation from more general systems engineering. It has been argued that this practice may lead to a number of problems because it assumes that safety and security problems can be analyzed in isolation from each other and from other systems requirements [5]. These problems include

inadequate understanding of safety and security semantics [6], lack of common terminology and disparate processes, which makes it difficult to take advantage of commonality between the two disciplines [7]. Furthermore, the separation of safety and security requirements engineering may also lead to incompleteness in requirements, incompleteness in essential requirements characteristics like verifiability and ambiguity in requirements [8].

At the same time both safety and security are being recognized as highly related disciplines [3, 4]. In both the disciplines, the overall objective is to protect valuable assets from harm [8]. Both safety and security requirements are generally specified as constraints on the system and are considered system-level properties that typically require a very high level of assurance [2].

Therefore, both safety and security requirements must be integrated with the rest of the system requirements for validation and verification from the early stages of the systems development [9]. They cannot be simply added on at a later stage. To this end an integrated and uniform systems engineering approach is required that not only facilitates integration of safety and security requirements with other requirements but also supports easy validation and formal verification of these requirements.

The GSE method aims to address these needs by introducing a simple and systematic systems design approach that not only seeks to bridge the gap between informal and formal representation of requirements but also provides for formal verification of requirements [10]. A simple graphical notation, called *behavior trees* (BT), is used to make the requirements easy to formally specify and validate. An integrated view of the system is first generated by integrating all the individually specified requirements into a single tree, referred to as *integrated behavior tree* (IBT) before deriving the system design in the form *design behavior tree* (DBT) by systematically refining the IBT. The automated translation of a DBT into other formal specification languages like *Communication Sequential Process* (CSP) [11] and *Symbolic Analysis Laboratory* (SAL) [12] makes it possible to formally verify the

critical system properties like safety and security conditions, liveness, deadlocks, etc. A component-based architecture and component behavior may be derived from the DBT in the form *component interaction network* (CIN) and *component behavior tree* (CBT) diagrams.

In this paper we present results of applying the GSE method to design a medical device called an *ambulatory infusion pump* (AIP). The AIP system has a number of safety and security concerns. Violation of any of these safety and security conditions may cause serious injury or death to patients. We formally verify the critical properties of our design by model checking the automated translation of the AIP's DBT into SAL specification.

The rest of the paper is organized as follows. Section 2, illustrates the basic GSE design process using examples from the AIP case study. Section 3, illustrates how the safety and security properties were modeled in the specified design. In section 4, we present the results of formal verification of safety and security properties. In section 5, we conclude our discussion and describe other related research work along with future direction of our research.

## 2. An overview of GSE method

In this section we provide a brief overview of the GSE method and use the AIP case study as a running example to illustrate the various steps in the method.

The GSE development process may be summarized as follows: 1) A given set of informal requirements is first formally specified as RBTs. A number of ambiguities and incompleteness defects are typically resolved at this stage. 2) All the individual RBTs are integrated into an IBT to generate an integrated view of requirements. All the requirements integration defects are resolved at this stage. 3) The IBT is systematically refined into a DBT. The impact of each design decision on the complete system is readily visualized as the changes are applied to the integrated view of the requirements. 4) The IBT and/or DBT may be translated into SAL or CSP for formal verification of the specifications. 5) Finally, a component-based architecture and individual component behavior can be derived from the DBT, which are typically represented in the form of CIN and CBT diagrams.

The tool support for the GSE method is under development. At present, a Behavior Tree Editor (BTE) is available for drawing and editing BTs [13] in a graphical environment. The tool stores the BT specification in XML format and supports automated translation of the BT specification into SAL specification language and CSP representation.

## 2.1 Introduction to the case study

An ambulatory infusion pump (AIP) is a medical device used for drug therapy for patients who are away from direct care of health professionals. The device is programmable to allow health professionals to configure it to meet the patient's needs. The hazards associated with the device are drug non-delivery or under-delivery, drug overdose and a serious medical condition called air embolism which is caused by accidental injection of air bubbles through the device.

**Table 1: AIP requirements**

No	Requirement
R1	The system is turned on when the batteries are put in and is turned off when the batteries are out.
R2	To start the pump, when in stopped state, start-stop button is held down until it beeps three times and (... ..) is displayed on the screen.
R3	To stop the pump, when in running state, start-stop button is held down until it beeps three times and (... ..) is displayed on the screen.
R4	When the battery is low, the system sends three beeps and displays battery low message on the screen.
R5	Every time the system pumps 1 ml of drug when the battery is low it sends a single beep alarm.
R6	After a set time pump activates to pump 1ml of drug through the line.
R7	When the volume reaches 5ml the system does three beeps and displays volume low message every 1ml as it counts down to empty.
R8	When there is no drug left, the pump enters stopped mode and the system sounds a continuous beeping alarm.
R9	When the line is closed/blocked or kinked the system does a constant alarm beeping if it is in the running mode.
R10	The security mode of the pump can be changed as follows: 1) the pump must be in stopped mode, 2) the current security mode is displayed by pressing the lock button, 3) the up and down arrow buttons are used to select the desired security mode (patient, clinic or program), 4) the enter is pressed to save the selected mode, 5) once the enter button is pressed 000 appears on the screen, 6) the up and down arrow buttons are used to select the correct password, 7) the enter button is used to select the displayed password and as a result displayed security mode is also selected as the current security mode.
R11	The pump's volume can be reset to a preset value as follows: 1) The pump must be in stopped mode, 2) the pump can be in any of the security modes, 3) the next is used to display volume on the screen, 4) pressing the enter button resets the volume to a preset value.
R12	The pump's upper limits of the pump's infusion rate can be set as follows: 1) The pump must be in stopped mode, 2) the security mode on the pump must be set to program mode, 3) the next button is used to display Infusion Rate Upper Limit on the screen, 4) the up and down arrow buttons are used to select the desired infusion rate, 5) pressing the enter button sets infusion rate upper limit to the displayed limit on the screen.
R13	The pump's infusion rate can be set as follows: 1) The pump must be in stopped mode, 2) The security mode of the pump must be set to clinic mode, 3) The next button is used to display Infusion Rate on the screen, 4) The up and down arrow button are used to select the desired infusion rate, 5) Pressing the enter button sets the infusion rate to the infusion rate displayed on the screen
R14	The amount of drug given can be cleared as follows: 1) The pump must be in stopped mode, 2) The security mode of the pump must be set to clinic mode, 3) The next button is used to display given on the screen, 4) Pressing the enter button clears the given amount to zero
R15	Whenever air is detected in the line, by the air detector sensor, the pump is stopped and the beeper a continuous beep
R16	The main screen displays the pump status (running/stopped), battery status (normal/low) and drug volume.
R17	If no key is pressed for 2 minutes then the screen is reset to the main screen

The drug non-delivery or under-delivery may be caused by an infusion rate lower than required or by an undetected blockage of the line. Similarly, a higher

rate of infusion may lead to the hazard of drug overdose. A discrepancy between the actual amount of drug infused and the amount of drug calculated as given by the AIP controller may also lead to the hazard of drug under-delivery or over-delivery. All of these hazards may lead to serious illness or death of the patient. The safety and security concerns related with the device are discussed in detail in section 3.

The requirements for AIP in this case study have been derived from the user manual of a commercial product CADD-Legacy® Ambulatory Infusion Pump [14]. The requirements have been simplified to make the case study easy to understand (table 1).

## 2.2. Requirements specification

One of the most difficult parts in systems engineering is recognized to be capturing and preserving the intent behind building the system in the system specification [15]. Another challenge is bridging the gap between informally specified requirements and their formal representation [4]. The GSE approach aims to address these issues by building the system right *out-of-its-requirements* by systematically translating one given requirement at a time [10].

Figure 1, illustrates the BT representation of the first requirements from table 1. The “^” symbol on one of the leaf nodes in the example is a reversion symbol to indicate that the system reverts back to a state higher in the tree. A summary of syntax of BT notation is provided in figure 2.

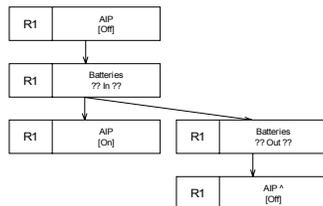


Figure 1: Requirements translation into BT

## 2.3. Requirements Integration

The process of integrating individual RBTs into an evolving IBT is analogous to putting the pieces of a *jig-saw-puzzle* together. The process involves locating where the root node of one BT occurs in another tree and grafting the two trees together at that point [10]. The process continues until all the RBTs have been grafted together into a single IBT. However, it is not always possible to integrate all the RBTs together. The requirements will only perfectly integrate if all the pre- and post-conditions have been clearly defined and there is no incompleteness in the requirements. Therefore, the integration step also plays an important

role in early detection of defects by forcing us to resolve requirements problems like incompleteness and inconsistency. The integration techniques along with integration axioms are presented in [10].

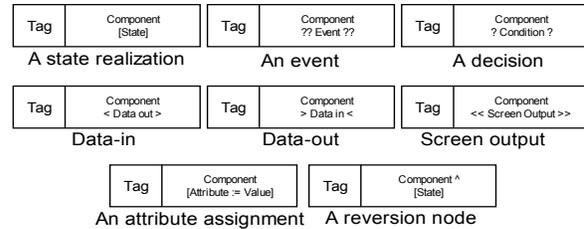


Figure 2: A summary of BT syntax

Once the IBT has been developed, it is systematically refined into a DBT. The process is aimed at assisting us in bridging the gap between problem domain (represented by the IBT) and solution domain (represented by the DBT). The impact of all the design decisions is readily visualized as the changes are applied to an integrated view of the whole system. This step reduces the complexity in the development process by reducing the information that needs to be kept in mind during requirements refinement process [10].

The DBT can play a significant role in development of mission critical systems as the critical requirements like safety and security are analyzed, refined and implemented throughout the development process. In the next sections we discuss how the safety and security requirements for the AIP system are modeled using BTs.

## 2.4. Deriving system design from requirements

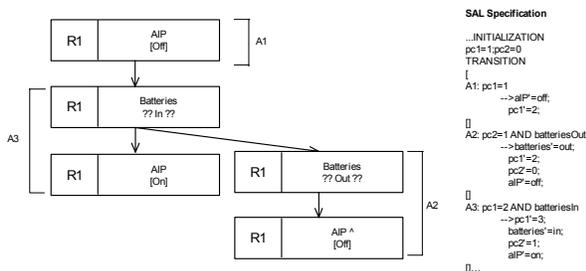
A number of refinement techniques are used to refine the IBT in an evolving DBT. These include, decoupling of operator, sensor and actuator behavior, separation of synchronous and asynchronous message passing, identification of atomic actions, identification of critical regions and interrupt handling, among others. Asynchronous messages to passive components in a DBT are depicted by leaf nodes in the tree. The atomic actions with causal dependencies are specified by joining the BT nodes using a line with no arrow, while atomic actions with no causal dependency are specified by boxing the BT nodes together.

In the case study we were able to find a number of defects of a critical nature in the AIP’s requirements during the refinement process. These include the missing requirement for calculating the amount of remaining drug if the pump operation is aborted to ensure that there is no discrepancy between the amount of drug calculated as given and the actual drug infused. Other missing requirements include displaying critical

state changes during the pump operation like ‘line blocked’ and ‘air in line’, etc. Due to lack of space a reduced form of the complete DBT is presented in figure 7. The figure is only intended to give the reader a feel of size and structure of the final design.

## 2.5 Model checking

The static analysis of GSE models is possible by automated translation of the BT specification into a SAL specification language. SAL is an integrated environment of static analysis tools that include tools for model checking and theorem proving [16]. In the SAL environment the systems are specified using a description language for state transition. The system properties of interest are calculated from SAL based on the system expressed as a transition system in this description language. In the SAL environment a number of tools provide support for abstraction, program analysis, theorem proving and, model checking. A detailed description of SAL translation rules is provided in [12]. Figure 3 provides a simple example of BT translation into SAL specification.



**Figure 3: Translation of BT into SAL**

The SAL specification can be checked for deadlocks using dead-lock-checker tool. Useful properties of the system can be specified using linear temporal logic (LTL) or computation tree logic (CTL). These properties can then be model checked using sal-smc and sal-bmc tools. A BT specification can also be translated into a CSP representation [11]. The FDR model-checker can be used to model check the CSP specification to verify the BT model.

## 3. Integrating safety and security requirements

An important aspect of designing mission critical systems is that the important systems concerns are not only implemented but are readily apparent in the design to ensure easy and effective validation and verification of critical requirements [4]. In this section we briefly describe how the safety and security properties are modeled in the DBT of the AIP system.

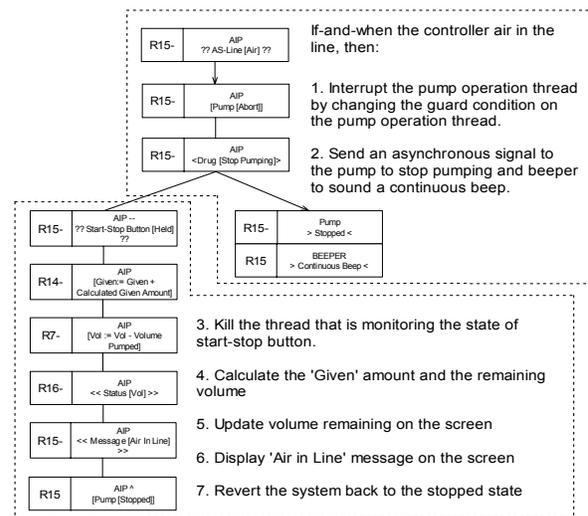
We also present the results of formal verification of critical properties of the system.

### 3.1 Modeling safety properties

The safety requirements of the AIP system include 1) the pump must be stopped if air is detected in the line, 2) the pump must be stopped if there is blockage in the line, and 3) there should not be any discrepancy between the amount of drug calculated as given and the actual amount infused.

In the design these conditions are implemented during the drug pumping operation when the pump is in *running* state. This pumping operation occurs in a loop which is controlled by a timer component named ‘Timer {Pump}’. The timer activates the pump every time it counts up to ‘pump-time’. The calculation of value for pump-time is based on the programmed infusion rate of the pump. At ‘pump-time’ the controller sends a signal to the pump to infuse 1ml of drug every time. Once the drug has been pumped, 1ml is subtracted from the amount of drug volume. The user is warned if the remaining drug level is low or the drug has reached the finished state.

In parallel to the pump operation described above, we have specified three threads that wait for; 1) the air-detector sensor to sense air in the line, 2) the occlusion sensor to sense a blockage in the line, and 3) start-stop button to stop the pump operation. The first two threads immediately interrupt the pump operation and the last thread only interrupts the pump operation if the button is held long enough.



**Figure 4: Modeling safety properties in BT**

Figure 4, illustrates the BT representation of the first thread. If and when the controller detects presence of air in the line it interrupts the pump operation by changing the ‘AIP [Pump [Running]]’ state to AIP ‘[Pump [Abort]]’. The thread running the pump

operation can only proceed if the ‘AIP [Pump]’ state is set to *running*. In addition, the interrupting thread sends asynchronous signal to the pump to stop and beeper to sound a continuous sound. The thread monitoring the stop-stop button is also killed (using the "--" operator). The amount of drug given and the remaining volume of the drug are calculated. The remaining volume is displayed on the screen along with ‘air in line’ message before the system reverts back to the ‘AIP [Pump [Stopped]] ^’ state.

This interrupt handling satisfies the first safety conditioned mentioned above. The second safety condition is specified in the similar manner. To ensure that there is no discrepancy between the actual amount infused and the amount calculated by the controller, the amount of drug pumped is calculated either at the end of normal cycle of pumping or when the pump operation is interrupted. If the pump operation is interrupted the amount of drug calculated as a function of time since the last pump time (recorded by the ‘Timer {Pump}’ component).

### 3.2 Modeling access control

The ability to verify that no rights are leaked to unauthorized user is an important characteristic of an access control model. To support verification of access control models, *constraint* expressions are typically added to the model [17]. These constraints describe the access control requirements of a configuration to ensure that access is not granted to any unauthorized user. Figures 5 and 6 illustrate how these constraints are expressed in a BT model.

**Table 2: Access control table for AIP functions**

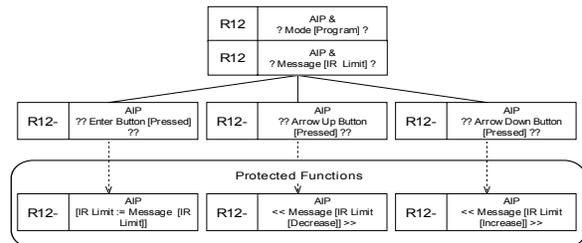
Screen	Patient			Clinic Mode			Program Mode		
	Buttons			Buttons			Buttons		
	↵	↑	↓	↵	↑	↓	↵	↑	↓
IR Upper Limit	x	x	x	x	x	x	Set	Inc.	Dec.
IR	x	x	x	Set	Inc.	Dec.	x	x	x
Given	x	x	x	Clear	x	x	x	x	x

Legend: ↵ = Enter, ↑ = Arrow-up, ↓ = Arrow-down, IR = Infusion rate

The advantage of using a BT to model access rights is two-fold; the often complex constraints are expressed graphically, which makes it is easier to specify and comprehend and that the constraints can be formally verified to ensure the correctness of the model. Furthermore, we get an integrated view of the complete system on which the access control is applied so that the impact of the security requirements is readily understood. Table 2 summarizes the access control for various AIP programming functions. The access to critical functions in clinic mode and program

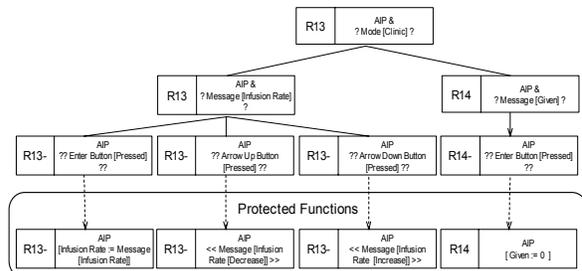
mode is protected through pre-programmed password in the controller.

In our design we have modeled the access control requirements by factoring out the constraints on each of the programming function. Figure 5, illustrates the constraint on setting upper limit for infusion rate. The root node in the diagram serves as the constraint for the all the behavior that proceeds the node. If the pump is in program mode (AIP ?Mode[Program]?) and the screen is set to display infusion rate upper limit (AIP ?Message[IR Limit]?) only then the limit can be changed using arrow up and arrow down keys or it can be set to the value displayed on the screen using the enter key.



**Figure 5: Functions accessible in program mode**

The access control for clinic mode is modeled in a similar fashion by factoring out the constraints on the function (see figure 6). In the clinic mode (AIP ?Mode[Clinic]?) the user can increase or decrease the infusion rate or set the infusion rate of the pump to the rate displayed on the screen (AIP ?Message[Infusion Rate]?). The other function that is accessible in the clinic mode is clearing of the amount of drug given to the patient. Clearing the amount means setting the ‘Given’ attribute in the controller to zero. The constraints on this function are that the pump should be in clinic mode (AIP ?Mode[Clinic]?) and the screen should be displaying the amount of drug given (AIP ?Message[Given]?).



**Figure 6: Functions accessible in clinic mode**

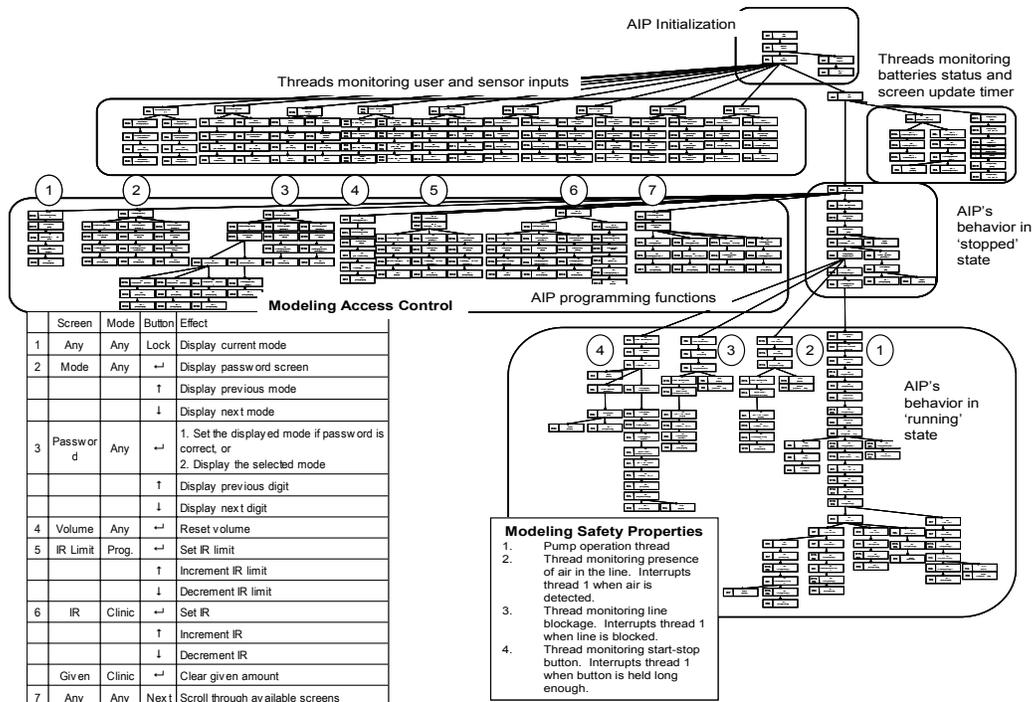


Figure 7: AIP's DBT with integrated safety and security requirements

Modeling access control by factoring out the constraints serves two important purposes. The constraints on various functions clearly stands out at the beginning of the functions, which improves readability and can assist in validation of requirements. The other benefit is from a usability point of view. Often the user interface functionality is context driven. For instance, in the case of AIP the enter button, arrow-up, and down-arrows have multiple functions. By modeling the constraints in the manner shown in figures 5 and 6, the context of the buttons is readily apparent making the implementation from usability point of view easier. A snapshot of the DBT that models the access control for AIP is illustrated in figure 7.

#### 4. Formal verification of security and safety properties

The AIP case study presented in this paper is a simplified version of a typical modern embedded system. As in most modern embedded systems the consequences of failure in this system are very serious. Therefore, it is important that the design of such a system must be formally verified.

Let us consider the hazard of drug non/under-delivery. This hazard may occur due to violation of number of safety and security properties. The hazard may occur as a result of; 1) the pump is not stopped if there is a blockage in the line, 2) the controller

software calculates the drug given more than the drug actually infused, and 3) pump's infusion rate is set to a rate slower than the required infusion rate by an unauthorized user. These conditions correspond to the AIP properties 2, 3, 7 and 9 listed in table 3.

The first safety property is represented as LTL formula (Th2 in table 4). This LTL formula states that it is globally true (represented with 'G' operator) that when there is a line blockage (line=blocked) then in the next step (represented by 'X' operator), the pump should be stopped (pump=pStopped). Using the sal-smc tool in the SAL environment we were able to prove this formula. Similarly, the safety property 3 was proved using LTL formula th3 (table 4).

The other condition that may lead to the hazard is the violation of security properties number 7 and 9 (table 3). These properties state that infusion rate must not be allowed to change when the pump is either in patient mode or program mode, i.e. it must only be set in clinic mode. The corresponding LTL formulas (Th7 and Th9) verified that our design does not violate these conditions.

Similarly, the hazard of drug overdose may be caused by violation of one or more safety and security properties. In this case, the properties 3, 5 and 8 must be ensured in the system to mitigate the risks of drug overdose. In addition, the violation of security properties 6 and 10 may also lead to drug overdose hazard. As illustrated in table 4, the LTL formulas for these properties were proved in the SAL environment.

**Table 3: Safety and security properties for AIP**

No.	Property	Ref.
Safety Properties		
1.	If there is air in the line the pump should not pump	Th1.
2.	If there is blockage of the line the pump should not pump	Th2.
3.	If the pump is stopped then the drug volume must be re-calculated	Th3.
4.	If the drug volume is zero then pump must not pump the drug	Th4.
Security Properties		
5.	The upper limit for infusion rate cannot be set in patient mode	Th5.
6.	The amount of drug 'given' must not be reset in patient mode	Th6.
7.	The infusion rate must not be set in patient mode	Th7.
8.	The upper limit for infusion rate cannot be set in clinic mode	Th8.
9.	The infusion rate must not be set in program mode	Th9.
10.	The amount of drug 'given' must not be reset in program mode	Th10.

**Table 4: Formal verification of AIP Design**

Ref.	LTL Formula	Outcome
Safety Properties		
Th1.	$G((line=air) \Rightarrow (X(pump=pStopped)))$	Proved
Th2.	$G((line=blocked) \Rightarrow (X(pump=pStopped)))$	Proved
Th3.	$G((aIP\_Drug=stopPumping) \Rightarrow (aIP\_Vol=vCalculated))$	Proved
Th4.	$G((pump=running) \Rightarrow NOT(aIP\_Volume = empty))$	Not Proved
Security Properties		
Th5.	$G((aIP\_Mode=patient) \Rightarrow NOT(aIP\_IRLimit=setLimit \wedge aIP\_MSG=mIRLimit))$	Proved
Th6.	$G((aIP\_Mode=patient) \Rightarrow NOT(aIP\_Given=reset \wedge aIP\_MSG=mGiven))$	Proved
Th7.	$G((aIP\_Mode=patient) \Rightarrow NOT(aIP\_InfusionRate=setInfusionRate) \wedge aIP\_MSG=mInfusionRate))$	Proved
Th8.	$G((aIP\_Mode=clinic) \Rightarrow NOT(aIP\_IRLimit=setLimit \wedge aIP\_MSG=mIRLimit))$	Proved
Th9.	$G((aIP\_Mode=program) \Rightarrow NOT(aIP\_InfusionRate=setInfusionRate) \wedge aIP\_MSG=mInfusionRate))$	Proved
Th10.	$G((aIP\_Mode=program) \Rightarrow NOT(aIP\_Given=reset \wedge aIP\_MSG=mGiven))$	Proved

The last hazard identified in the case study is the hazard of air embolism. The safety property to avoid this hazard requires the pump to be stopped if there is air detected in the line (safety property 1). This property was verified using LTL formula (Th1). However, during safety analysis another safety property was identified which requires that the pump should not attempt to infuse drug if the pump runs out of drug. If the system attempts to pump when no drug is present in the drug cartridge then the hazard of air embolism may occur (safety property 4). This is only

possible in combination of violation of safety property 1, i.e. the air in the line is not detected. We were not able to prove the corresponding LTL formula (Th4) due to flaw in our design. There is no guard to prevent system from stop pumping when there is no drug left in the drug cartridge.

The fact we were able to uncover a design flaw in the DBT during formal verification reinforces the importance of using formal methods in design of critical applications. Another important observation from the case study is that an integrated view of safety and security requirements has the potential to play a crucial role in safety and security analysis of system. For instance, in the AIP system we identified both safety and security properties that must be implemented in the system to avoid the drug overdose hazard.

## 5. Conclusion

Safety and security engineering is an iterative process that is performed through out the systems development process. The requirements are continuously analyzed and refined during the process. Isolation of safety and/or security requirements engineering from systems engineering may result in incomplete understanding and late resolution of problems. An integrated and uniform approach to support this process should be aimed at resolving conflicts between competing requirements as early as possible.

In this paper we have presented the process of designing a safety critical medical device using the GSE method. The integrated view generated using the method provided a useful platform for integrating safety and security requirements. The IBT and DBT can potentially play an important role in the designs of critical systems as the impact of each requirement on the complete system is readily apparent in the integrated view. The simple graphical notation can play an important role in validation of requirements and formal semantics of the notation leads to formal verification of critical properties of the system. The gap between the informal requirements specification and their formal representation was bridged by the systematic translation technique and transition from problem domain to solution domain was achieved by methodically refining the IBT into the final DBT.

The fact that we were able to uncover a subtle design flaw in the DBT during formal verification reinforces the importance of using formal methods in design of critical applications. Another important observation from the case study is that integrated analysis of critical properties like safety and security is necessary for effective mitigation of risks associated with the system.

The research work in the area of integration of safety and security requirements have been either on conceptual level [6, 8] or limited to using techniques from one discipline to another [7]. The KAOS framework [18] and the DISCOS methodology [5] attempt to integrate safety and security requirements engineering with more general requirements engineering. The GSE method distinguishes itself from these and other similar approaches on the basis of its support for translation of informal requirements into formal specification, presenting an integrated view of requirements and facilitating the design process in a systematic manner.

Our future research work is aimed at providing support for hazard and threats analyses in the GSE method. The support for timing and performance analysis of DBT is also being investigated. In related work, an automated failure mode and effect analysis is being developed to support safety analysis in the GSE method [12].

## Acknowledgement

This work is partially funded by Australian Research Council (ARC) under the ARC Centers of Excellence program. We would also like to acknowledge the contribution of Dr. Lars Grunske and Nisansala Yatapanage for their assistance in model checking our design specification using SAL specification language.

## References

- [1] J. C. Knight, "Safety Critical Systems: Challenges and Directions," presented at 24th International Conference on Software Engineering, ICSE 2002, Orlando, Florida, 2002.
- [2] N. G. Leveson, *Safeware: System Safety and Computers*: Addison-Wesley Publishing Company, 1995.
- [3] P. T. Devanbu and S. G. Stubblebine, "Software Engineering for Security: A Roadmap," presented at International Conference on Software Engineering (ICSE 2000) - Future of SE Track, 2000.
- [4] R. Lutz, "Software engineering for safety: A roadmap," presented at The Future of Software Engineering, 2000.
- [5] I. Sommerville, "An Integrated Approach to Dependability Requirements Engineering," presented at 11<sup>th</sup> Safety-Critical Systems Symposium, Bristol, 2003.
- [6] A. Burns, J. McDermid, and J. Dobson, "On the Meaning of Safety and Security," *Computer Journal*, vol. 35, pp. 3-15, 1992.
- [7] J. Rushby, "Critical System Properties: Survey and Taxonomy," Computer Science Laboratory, SRI International SRI-CSL-93-1, 1994.
- [8] D. G. Firesmith, "Common Concepts Underlying Safety, Security, and Survivability Engineering," Software Engineering Institute, Carnegie Mellon University CMU/SEI-2003-TN-033, December 2003.
- [9] N. G. Leveson, "Intent Specifications: An Approach to Building Human-Centered Specifications," *IEEE Transactions on Software Engineering*, vol. SE-26, 2000.
- [10] R. G. Dromey, "Genetic Design: Amplifying Our Ability to Deal With Requirements Complexity," *Lecture Notes in Computer Science*, vol. 3466, pp. 95-108, 2005.
- [11] K. Winter, "Formalising Behavior Trees with CSP," presented at International Conference on Integrated Formal Methods, IFM'04, 2004.
- [12] L. Grunske, P. Lindsay, N. Yatapanage, and K. Winter, "An Automated Failure Mode and Effect Analysis based on High-Level Design Specification with Behavior Trees," accepted in Fifth International Conference on Integrated Formal Methods (IFM2005), Eindhoven, The Netherlands, 2005.
- [13] C. Smith, K. Winter, I. Hayes, R. G. Dromey, P. Lindsay, and D. Carrington, "An Environment for Building a System Out of Its Requirements," presented at Tools Track, 19th IEEE International Conference on Automated Software Engineering, Linz, Austria, 2004.
- [14] Deltec, "Operator's Manual, CADD-Legacy ® 1 Ambulatory Infusion Pump, Model 6400," vol. 2005: Smiths Medical MD, Inc., 2005.
- [15] F. P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer*, vol. 20, pp. 10-19, 1987.
- [16] N. Shankar, "Combining Theorem Proving and Model Checking through Symbolic Analysis," presented at CONCUR'00: Concurrency Theory, 2000.
- [17] T. Jaeger and J. E. Tidswell, "Practical Safety in Flexible Access Control Models," *ACM Transactions on Information and System Security*, vol. 4, pp. 158-190, 2001.
- [18] C. Ponsard, P. Massonet, A. Rifaut, J. F. Molderez, A. v. Lamsweerde, and H. T. Van, "Early Verification and Validation of Mission-Critical Systems," presented at FMICS'04, Linz, Austria, 2004.