



Managing Complexity in Modelling Embedded Systems

Saad Zafar¹ and R. G. Dromey²

¹Institute for Integrated and Intelligent Systems,

²Software Quality Institute,

Griffith University

s.zafar/g.dromey@griffith.edu.au

Topics

- Overview
- Background
- The GSE Design Strategy
- AIP – Case Study
- Analysing the AIP Design
- Conclusion
- References
- Glossary of acronyms
- Acknowledgements
- Questions and comments

Overview

- Increase dependence on embedded systems
 - Need for easy to use modelling techniques that help manage complexity of modern system
 - Genetic Software Engineering (GSE)
 - A development methodology that help control complexity
 - Based on simple graphical notation called Behavior Trees (BT)
 - Supports early defect detection in requirements
 - Supports formal verification of critical properties of the system
- Ambulatory Infusion Pump – A case study
 - The GSE method is described using the case study

Background

- Modern embedded systems
 - Are responsible for critical functions in automobiles, aircrafts, and medical devices
 - Must ensure availability of critical services
 - Must reliably maintain a number of safety conditions
 - Have serious consequences of failure

Background

- Increased complexity of modern systems^[1,2]
 - Needs assurance of critical system properties
 - The design process should be augmented by better and more formal methods
- The use of formal method is limited^[3]
 - Hard to comprehend
 - Difficult to model, analyse and verify the *complete* system
 - Gaps in informal and formal specification
 - Gaps in deriving software requirements from system requirements

The GSE Design Strategy [4]

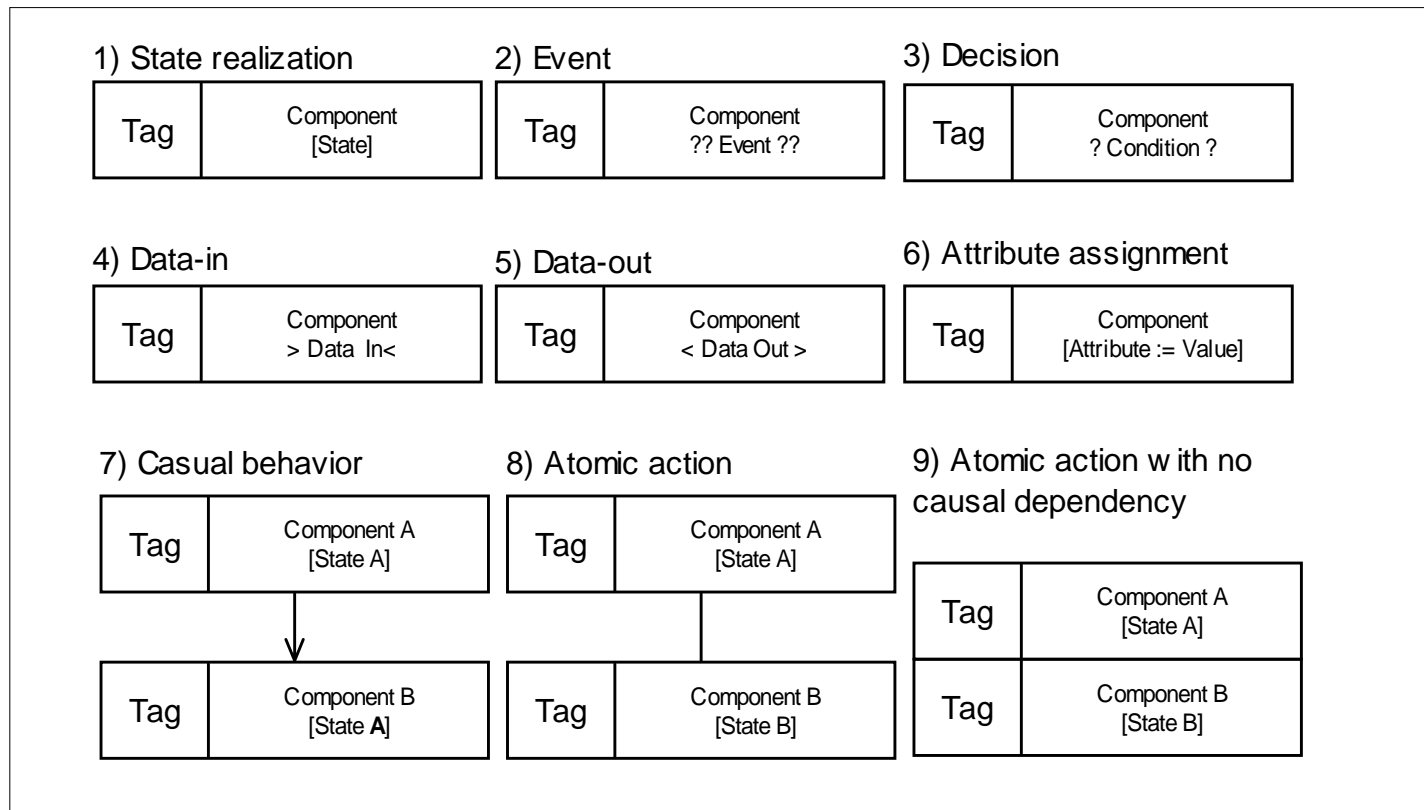
- Bridging the gap between formal and informal requirements specification
 - Building system *out-of-its-requirements*
- Early defect detection
 - Requirements translation
 - Requirements integration
- Bridging the gap between problem domain and solution domain
 - Systematic refinement of integrated requirements
- Requirements traceability
 - Traceability tags in the BT notations are used to support forward and backwards traceability

The GSE Design Strategy

- Managing complexity
 - Reducing the load on our short-term memory
 - Use of simple graphical notation with formal semantics
- Formal verification of requirements
 - Automated translation of system specification into CSP^[5] and SAL^[6] specification languages
 - Model checking the specification for system properties of interest
- Easy validation of requirements
 - Simple graphical notation intended to support easy validation of requirements
 - Any ambiguities and incompleteness in requirements are clearly marked

The GSE Design Strategy

■ A summary of BT notation syntax



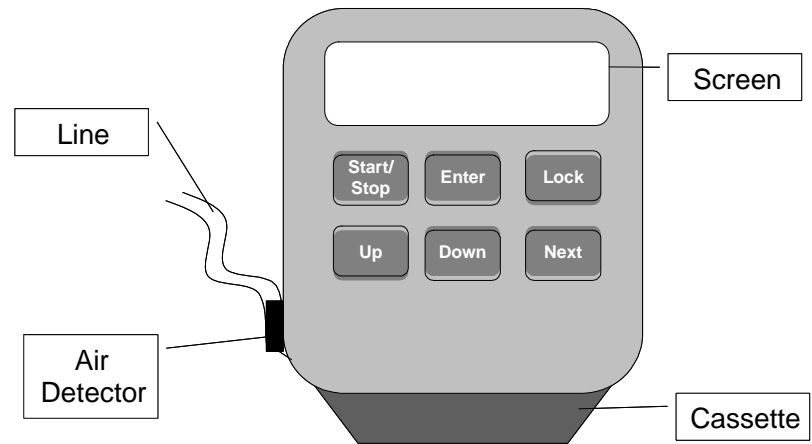
AIP – Case Study

- An overview

- A safety-critical device^[7]
- Used for drug therapy for patients who are away from direct care of medical practitioners
- Delivers the drug to the patient at the programmed infusion rate

- System hazards

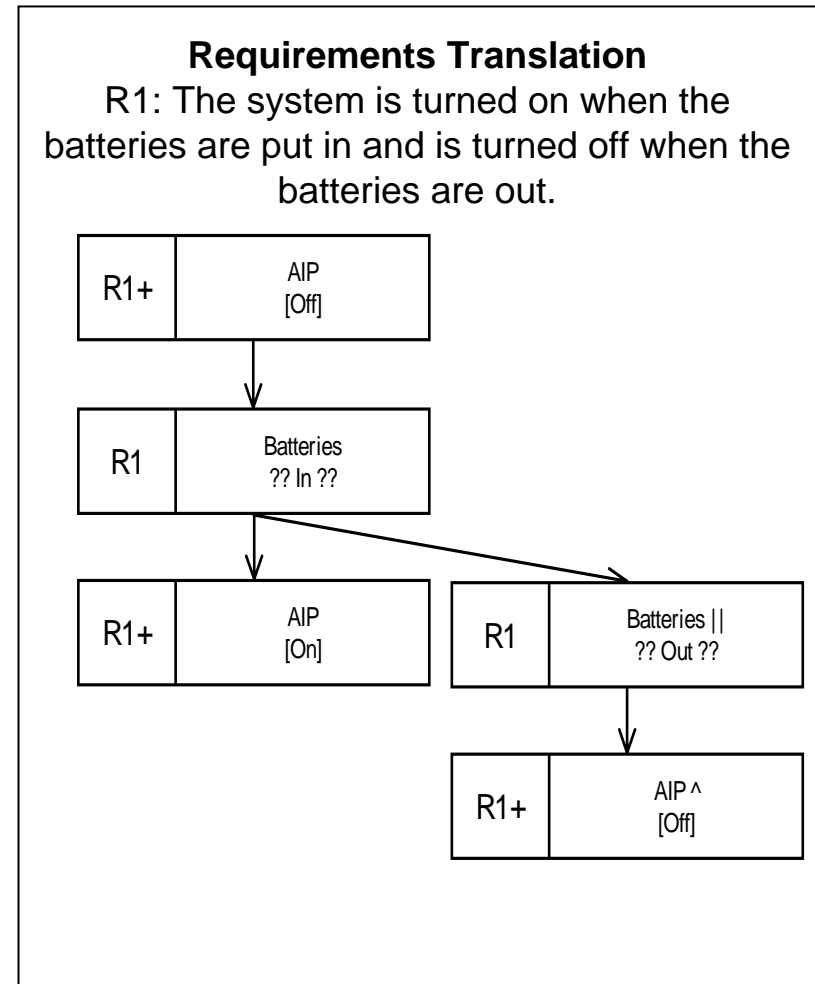
- Drug under/non-delivery
- Air embolism
- Drug overdose



Hazard	Cause
Under-delivery or non-delivery of drug	<ul style="list-style-type: none"> - Obstruction or kinks in the line - Incorrect calculation drug infused
Air embolism	<ul style="list-style-type: none"> - Presence of air in the line
Over delivery of drug	<ul style="list-style-type: none"> - Incorrect calculation of drug infused

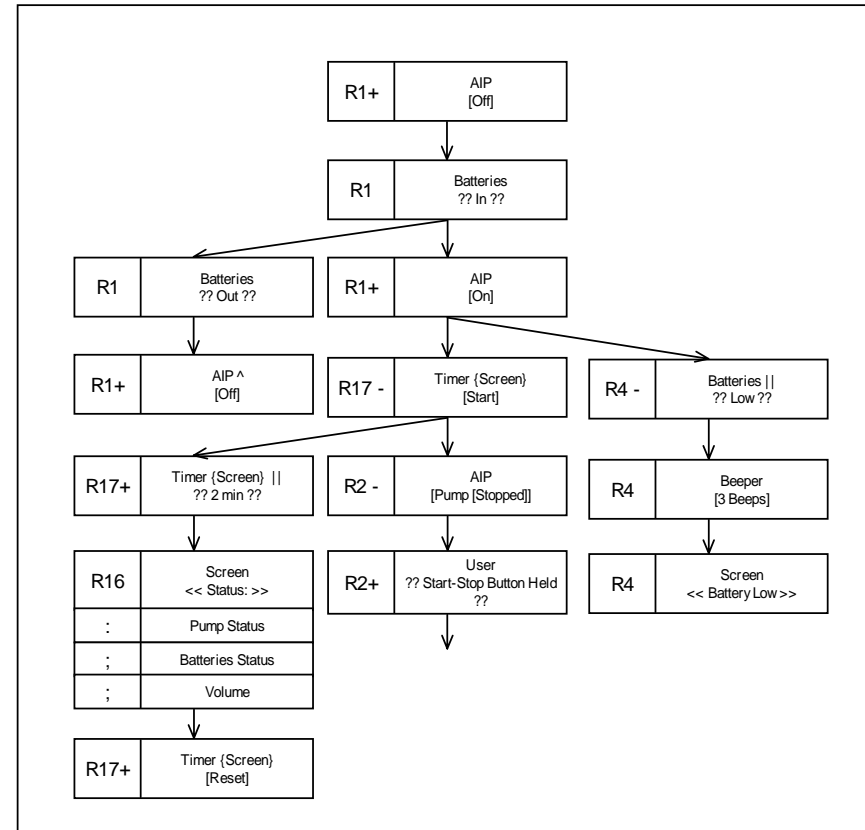
AIP – Case Study

- Requirements translation
 - Controlling complexity
 - Dealing with one requirement at a time
 - Early defect detection
 - Any implied or missing behaviour is clearly marked during translation
 - Formal specification
 - Requirements specified in BT notation



AIP – Case Study

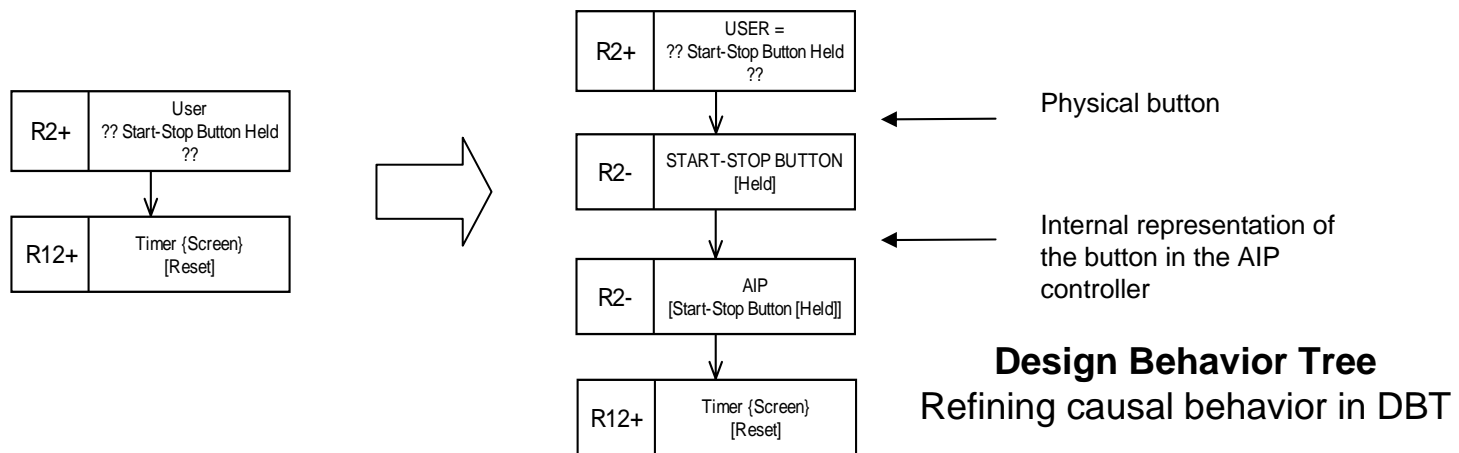
- Requirements Integration
 - Controlling complexity
 - Integrating requirements together like pieces of *jig-saw-puzzle*
 - Early defect detection
 - Resolving all the integration problems (identification of pre- and post-conditions)



Requirements Integration
Integration of R4 into an IBT

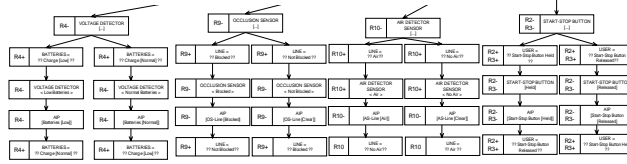
AIP – Case Study

- Developing a Design Behavior Tree
 - Controlling complexity
 - Systematically refining the IBT
 - Impact of all the design decisions is readily apparent
 - Identification of defects
 - Requirements incompleteness and inconsistencies

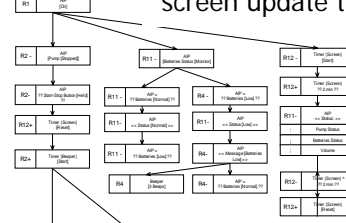


AIP initialization

Threads monitoring user and sensor inputs



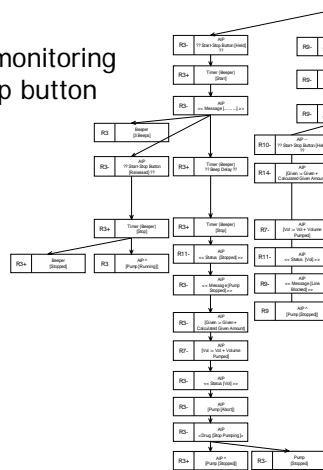
Threads monitoring batteries status and screen update timer



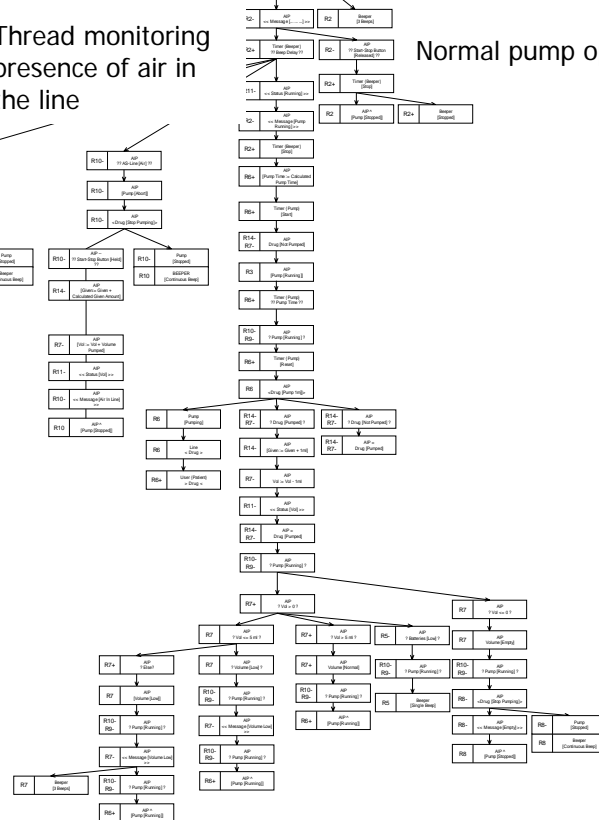
Thread monitoring blockage in the line

Thread monitoring presence of air in the line

Thread monitoring start-stop button



Normal pump operation



The integrated and refined DBT for AIP system

AIP – Case Study



■ Formal Verification

- Automated translation of BT into SAL specification language^[8]
 - A BT is represented in a single SAL transition system module
 - The components and their states are declared as state types in the module
 - The BT events are translated as input variables
 - Program counters are used to keep track of concurrent state transitions in the tree
- SAL Tools^[9]
 - sal-sim (SAL Simulator), sal-path-finder (produces random traces), dead-lock-checker (checks for dead locks), sal-smc (checks LTL formulas), etc.

Analysing AIP Design

AIP Design

- The pump operation occurs in a loop controlled by an internal timer which activates the pump at given times to pump measured doses of drug based on a set infusion rate (thread 4)
- The amount of drug infused is subtracted at each pump cycle
- During execution of each pump cycle the controller checks the remaining volume and warns if the pump is empty or running on low drug volume or low batteries
- In parallel to the pump operation there are three threads that wait for:
 - Start-stop button to be held down (thread 1)
 - The occlusion sensor to sense blockage in the line (thread 2)
 - Air-detector sensor to sense air in the line (thread 3)

Analysing AIP Design

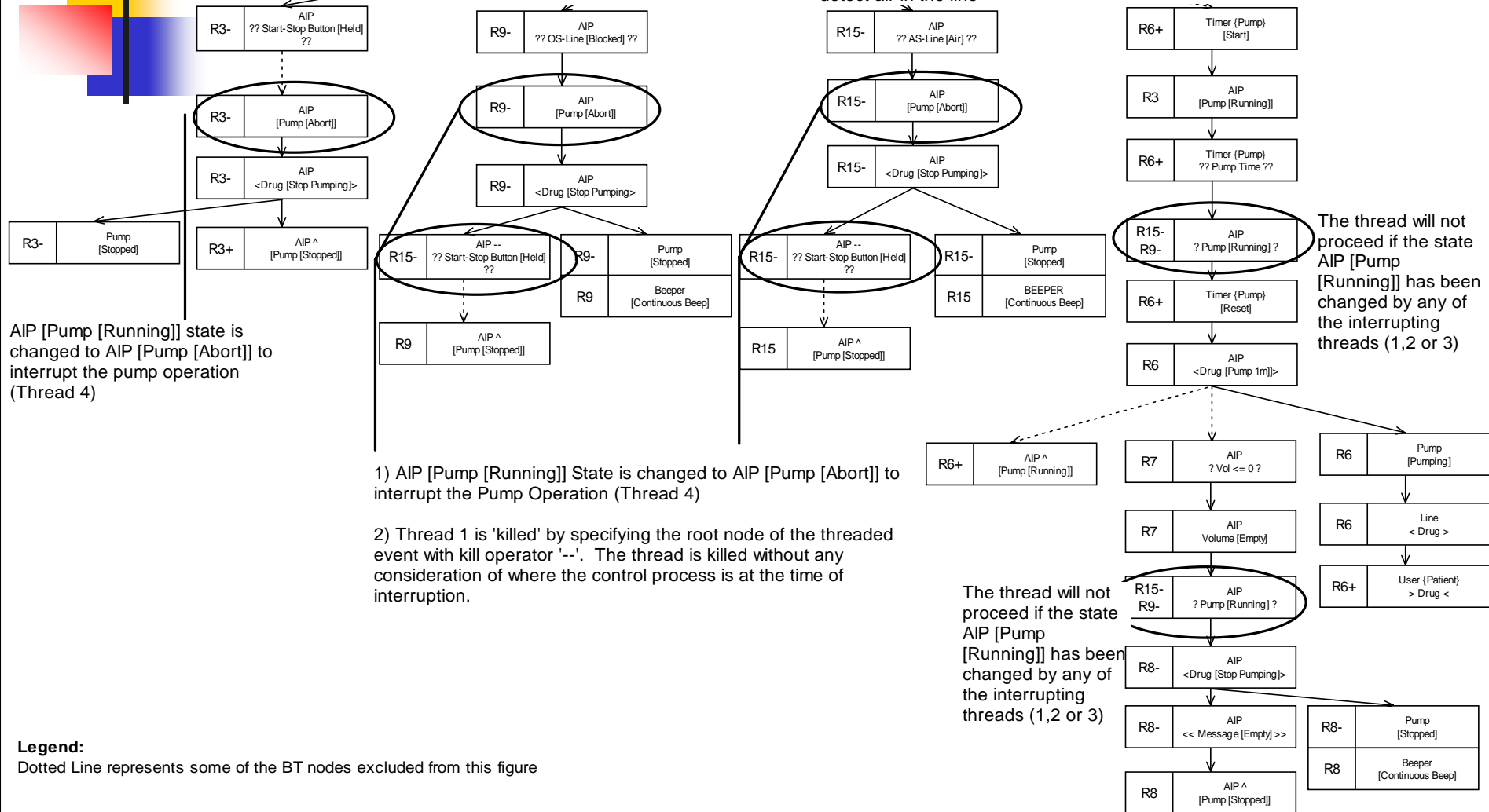
- Interrupt handling
 - Interrupts normal pump operation (thread 4) when
 - Start-stop button is held long enough (thread 1)
 - Occlusion sensor detects line blockage (thread 2)
 - Air detector senses air in the line (thread 3)
 - Interruption techniques
 - Interrupting threads changes the guard in the normal pump operation (AIP [Pump [Running]]) to (AIP [Pump [Abort]])
 - Interrupting thread explicitly kills the thread with '--' operator

Thread 1: Waiting for start-stop button to be held long enough to stop the pump operation

Thread 2: Waiting for occlusion sensor to detect blockage in the line

Thread 3: Waiting for air-detector sensor to detect air in the line

Thread 4: Pump Operation



Legend:
Dotted Line represents some of the BT nodes excluded from this figure

Analysing AIP Design

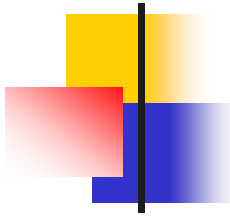
- Verifying system properties
 - Automated translation of DBT into SAL specification language
 - Model check safety properties of the AIP system

No.	Condition	Ref.	LTL Formula	Outcome
1.	If there is air in the line the pump should not pump	Th1.	$G((\text{line}=\text{air}) \Rightarrow (X(\text{pump}=\text{pStopped})))$	Proved
2.	If there is blockage of the line the pump should not pump	Th2.	$G((\text{line}=\text{blocked}) \Rightarrow (X(\text{pump}=\text{pStopped})))$	Proved
3.	If the pump is stopped then the drug volume must be re-calculated	Th3.	$G((\text{aIP_Drug}=\text{stopPumping}) \Rightarrow (\text{aIP_Vol}=\text{vCalculated}))$	Proved
4.	If the drug volume is zero then pump must not pump the drug	Th4.	$G((\text{pump}=\text{running}) \Rightarrow \text{NOT}(\text{aIP_Volume} = \text{empty}))$	Not Proved

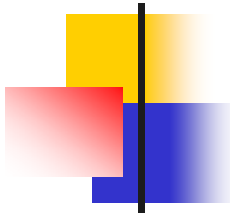
Conclusion

- GSE strategy for controlling complexity
 - Reducing the amount of information that needs to be processed at a given time
 - Requirements translation
 - Translating one requirement at a time
 - Requirements integration
 - Putting together individual requirement BTs and resolving integration problems one at time
 - Requirements refinement
 - The impact of all the design decisions are readily visualized

Conclusion

- 
- Early defect detection
 - Requirements translation
 - Requirements incompleteness and ambiguities
 - Requirements integration
 - Resolving all the integration defects
 - Requirements validation
 - All requirements are clearly marked for easy validation
 - Traceability of the requirements is maintained using traceability tags
 - Simple graphical notation makes it easy to validate requirements
 - Formal verification
 - Automated translation of DBT into formal specification languages like SAL and CSP

Future Directions

- 
- The GSE method
 - Support for timing and performance analysis
 - Support for threat and hazard analysis
 - Safety and security requirements specification
 - Automated code generation
 - Automated failure mode and effect analysis^[8]

References

1. Perrow, C 1984, *Normal accidents : living with high-risk technologies*, Basic Books, New York.
2. Leveson, NG 1995, *Safeware: System Safety and Computers*, Addison-Wesley Publishing Company.
3. Knight, JC 2002, 'Safety Critical Systems: Challenges and Directions', paper presented to 24th International Conference on Software Engineering, ICSE 2002, Orlando, Florida, 19-25 May.
4. Dromey, 2005, 'Genetic Design: Amplifying Our Ability to Deal With Requirements Complexity', *Lecture Notes in Computer Science*, vol. 3466, pp. 95-108.
5. Winter, K 2004, 'Formalising Behavior Trees with CSP', paper presented to International Conference on Integrated Formal Methods, IFM'04.
6. Shankar, N 2000, 'Combining Theorem Proving and Model Checking through Symbolic Analysis', paper presented to CONCUR'00: Concurrency Theory, August.
7. Deltec 2005, *CADD-Legacy® 1 Ambulatory Infusion Pump, Model 6400*, Smiths Medical MD, Inc., viewed April 2005, <<http://www.deltec.com/products.cfm>>.
8. Grunske, L., P. Lindsay, et al. (2005). 'An Automated Failure Mode and Effect Analysis based on High-Level Design Specification with Behavior Trees. Fifth International on Integrated Formal Methods (IFM2005), Eindhoven, The Netherlands.
9. Moura, Ld 2004, *SAL: Tutorial*, SRI International.

Glossary/Acronyms

- AIP – Ambulatory Infusion Pump
- ARC – Australian Research Council
- BT – Behavior Tree
- CSP – Communicating Sequential Processes
- DBT – Design Behavior Tree
- DCCS – Dependable Complex Computer-based Systems
- GSE – Genetic Software Engineering
- IBT – Integrated Behavior Tree
- LTL – Linear Temporal Logic
- SAL – Symbolic Analysis Laboratory

Acknowledgements



- Funding

- This work is partially funded by Australian Research Council (ARC) under the ARC Centres of Excellence program.

- Contribution

- Assistance by Dr. Lars Grunske and Ninsansala Yatapanage in model checking the AIP design using SAL specification

- Suggestions

- Constructive suggestions by our colleagues in the Dependable Complex Computer-based Systems (DCCS)



Questions and discussion
