
University of
Southampton

University of Southampton

Designing Exception Handling using Event-B

Asieh Salehi Fathabadi, **Colin Snook,** Thai Son Hoang, Robert Thorburn, Michael Butler, Leonardo Aniello, Vladimiro Sassone

26th June 2024, Bergamo, Italy

Outline

University of Southampton

- Motivation + Background
- Case study – SBB electronic voting system
  - normal behaviour
  - exceptional behaviour
  - verifying recovery
- Conclusions
- Future directions

Motivation

University of Southampton

- Systems must deal with failures and attacks - Exceptions
  - Failures can occur in machinery that is part of the system
  - Security Attacks can target code that is part of the controller of the system
    - E.G ARM Morello - raise exceptions for capability violations <<<< HD-Sec project
- Designers naturally focus on normal behaviour first, but …
- Exceptions must be handled in a way that maintains system integrity properties
- Formal Analysis of exceptional behaviour would verify exception handling
  - But Event-B has no explicit support/structures/methods for exception handling 
- We are exploring approaches to model/verify exception handling in Event-B

Background - Exceptions

University of Southampton

- We consider two types of exception detected by the following interrupt signals
  - SIGPROT: a memory protection exception can be generated by capability hardware when a pointer is used outside of its protected range (representing a possible memory attack).
  - SIGALARM: a timeout exception can be raised when an expected response from the environment (e.g. a machine or user) fails to occur within a time limit.

Background - Event-B

University of Southampton

```
1  context c
2  constants SA SB
3  sets states
4  axioms @def−states: partition(states, {SA}, {SB})
```

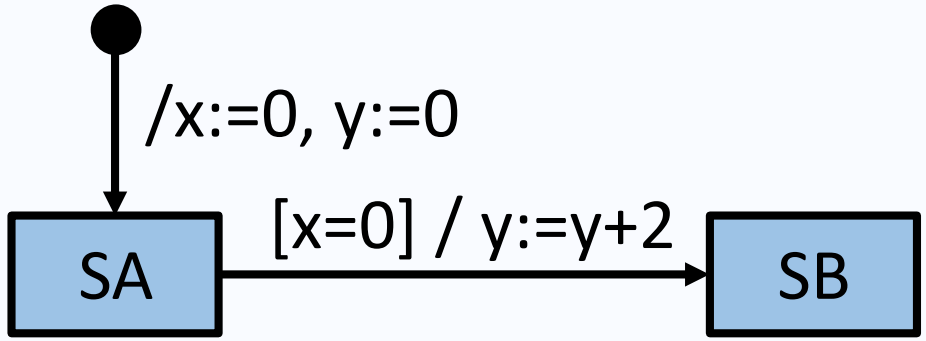
Context: Static aspects of the model

```
1  machine m, sees c
2  variables st // state
3  invariants @typeof−st: st ∈ {SA, SB}
4  events
5   event INITIALISATION
6   then @init−st: st := SA
7   end
8   event t1
9   where @source: st = SA  // event t1 guard
10  then @target−st: st := SB  // event t1 action
11 end
```

Machine: Dynamic aspects of the model

- Used to model specifications of systems
- Proof of model consistency with respect to
  - Invariant properties
  - Abstract levels
- Event-B models have two parts:
  - Contexts: Contain sets, constants and axioms
  - Machines: Variables, events, and invariants
- Refined to add more details to the model
  - Superposition refinement: additional variables
  - Data refinement: abstract variables replaced with new concrete variables
  - New events: Refines implicit abstract event that does nothing “skip” – i.e. cannot modify old variables

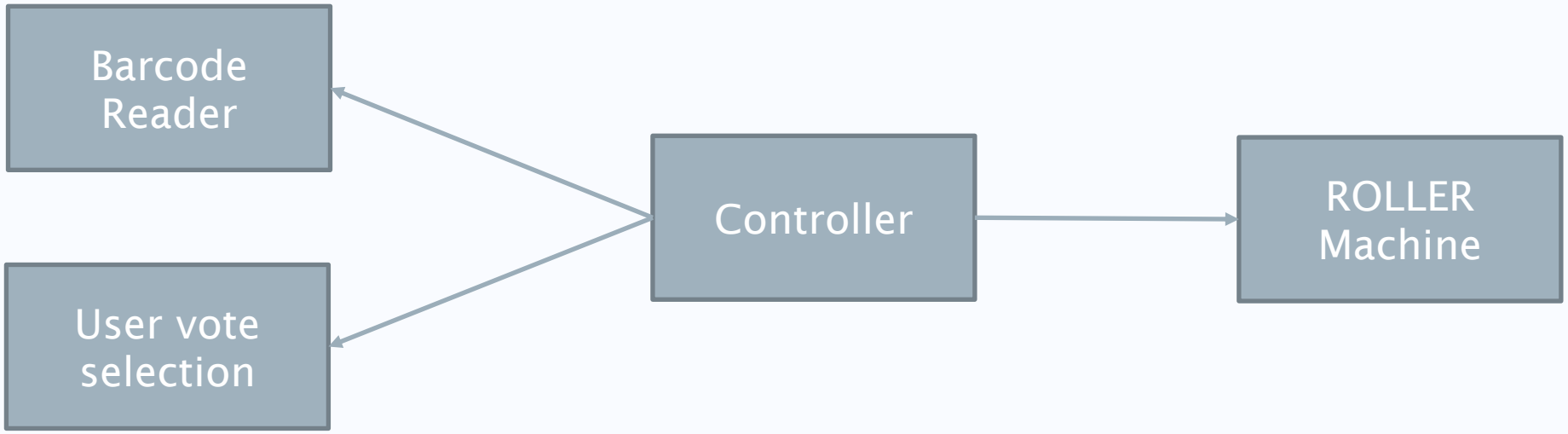
Background – UML-B state machines



- Transitions:
  - enabled when their source state is active and change the active state of the system
    - [..] Guards: Extra conditions that must be satisfied for a transition to be enabled
    - / …. Actions: updates performed as transition fires
- Variables - Used in guards and actions
- Parallel Statemachines - can synchronise transitions
- Hierarchical - States may contain nested state machines
- UML-B generates Event-B automatically

Case Study – SBB voting system

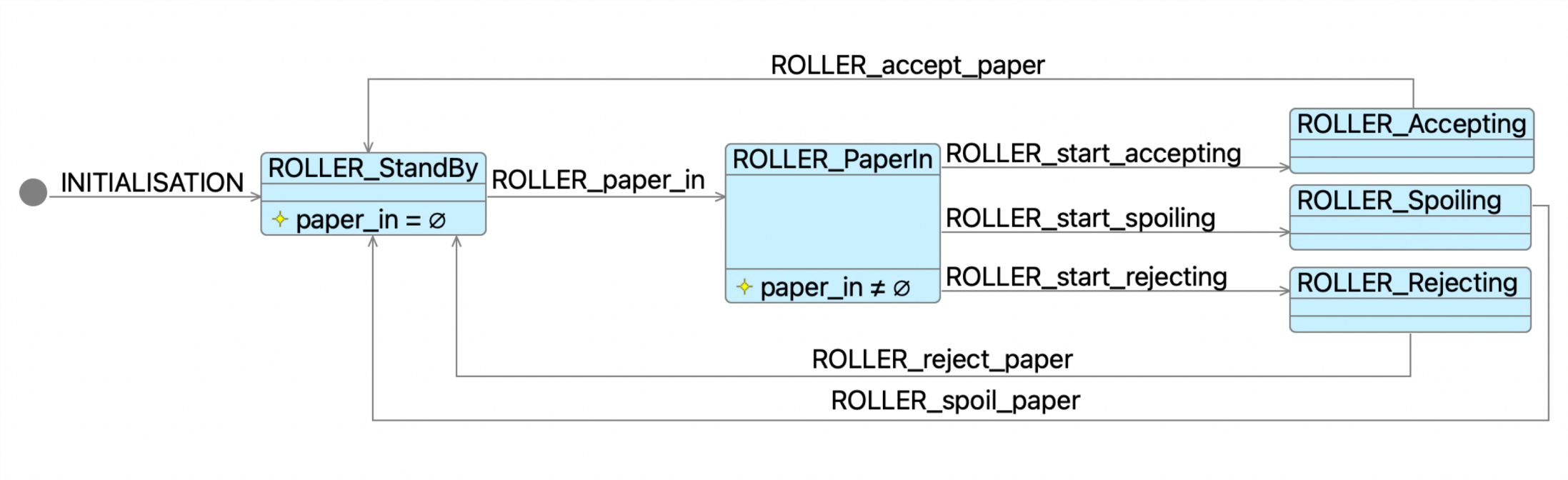
University of Southampton



Case Study – Normal Behaviour

University of Southampton

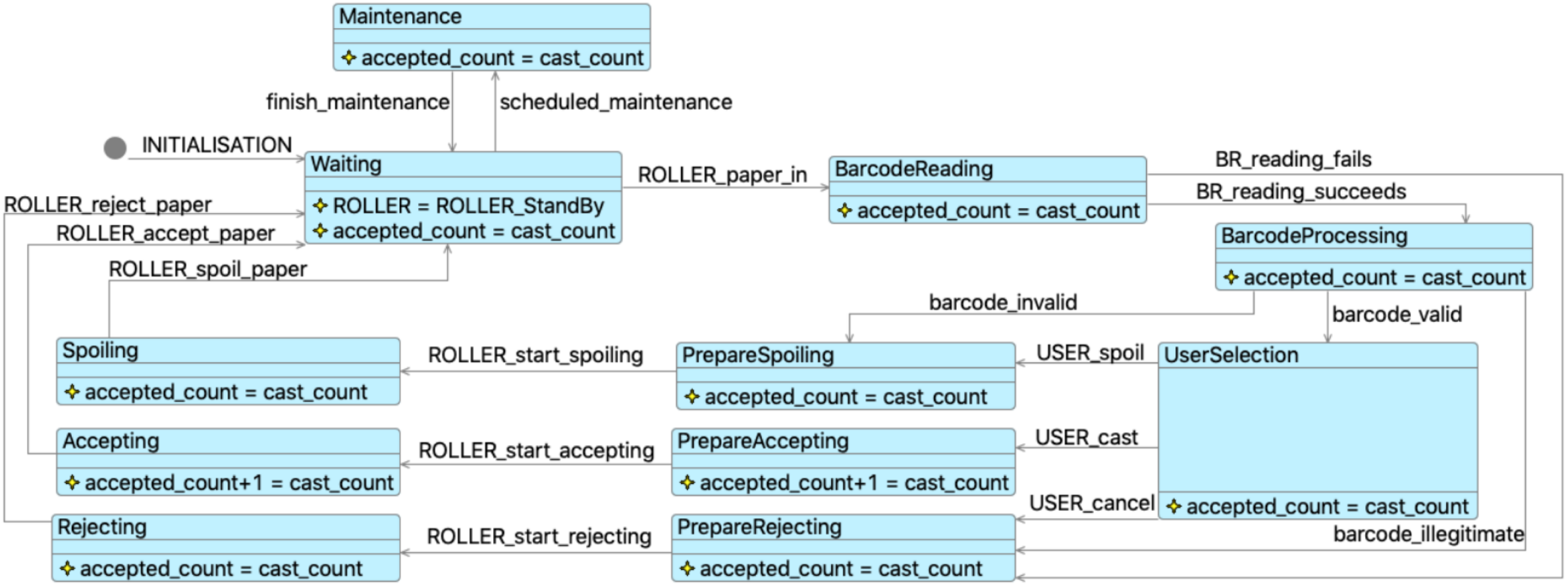
- Model of the ROLLER equipment.



Case Study – Normal Behaviour

University of Southampton

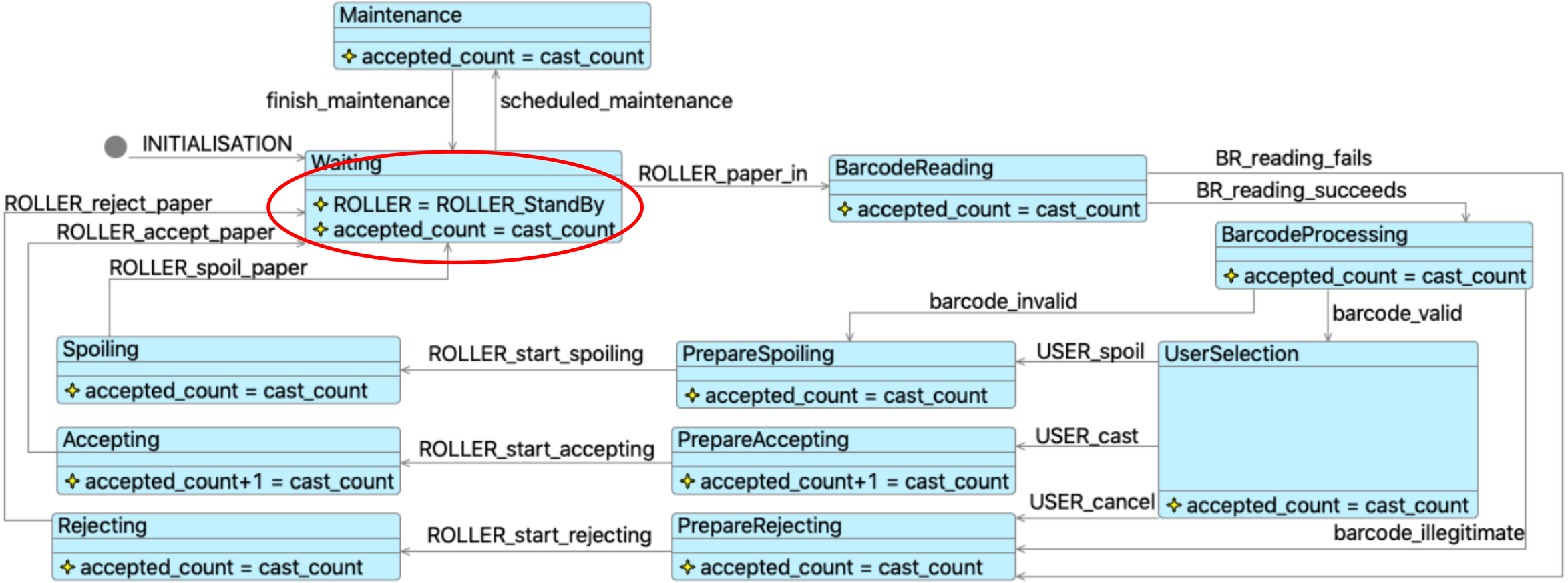
- Model of the main controller.
- Some transitions synchronise with the ROLLER state machine.



Case Study – Normal Behaviour

University of Southampton

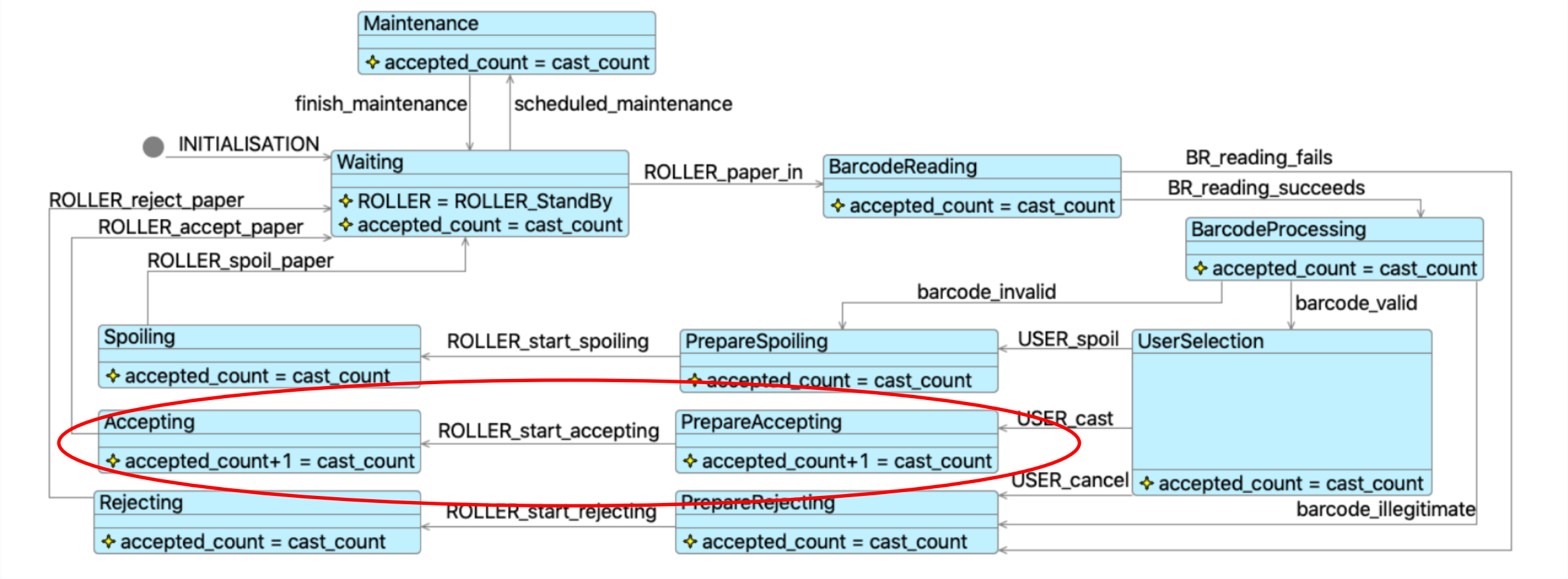
- In the waiting state
- The roller should be in the state ROLLER Standby so that it is ready to take another paper.
- The count of votes cast by the user should be the same as the count of papers accepted by the roller.



Case Study – Normal Behaviour

University of Southampton

- Other invariants needed to achieve proof
  - But note one area where cast count is out of step with accepted count



Case Study – Exceptional Behaviour

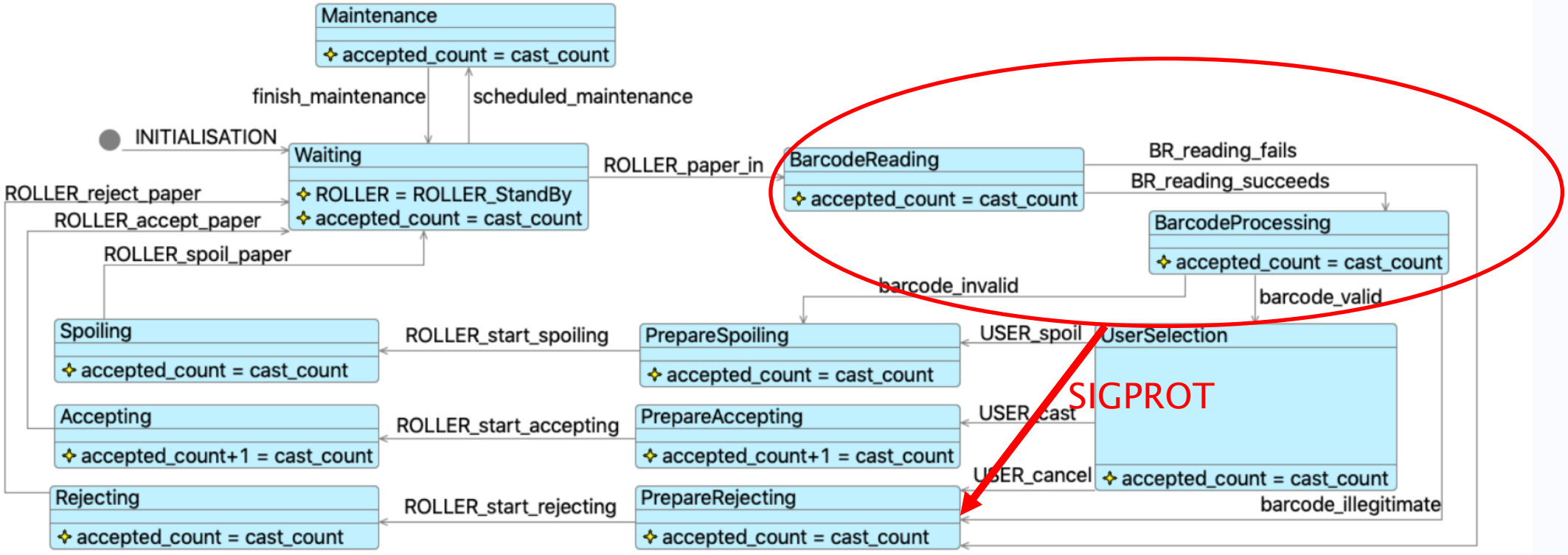
University of Southampton

Exception	Signal	States	Handling	Rollback
Memory protection error	SIGPROT	BarcodeReading, BarcodeProcessing, UserSelection	reject ballot	-
User does not enter selection	SIGALRM	UserSelection	reject ballot	-
Roller jammed	SIGALRM	Accepting	maintenance	cast count
Roller jammed	SIGALRM	Spoiling, Rejecting	maintenance	-

Case Study – Exceptional Behaviour

University of Southampton

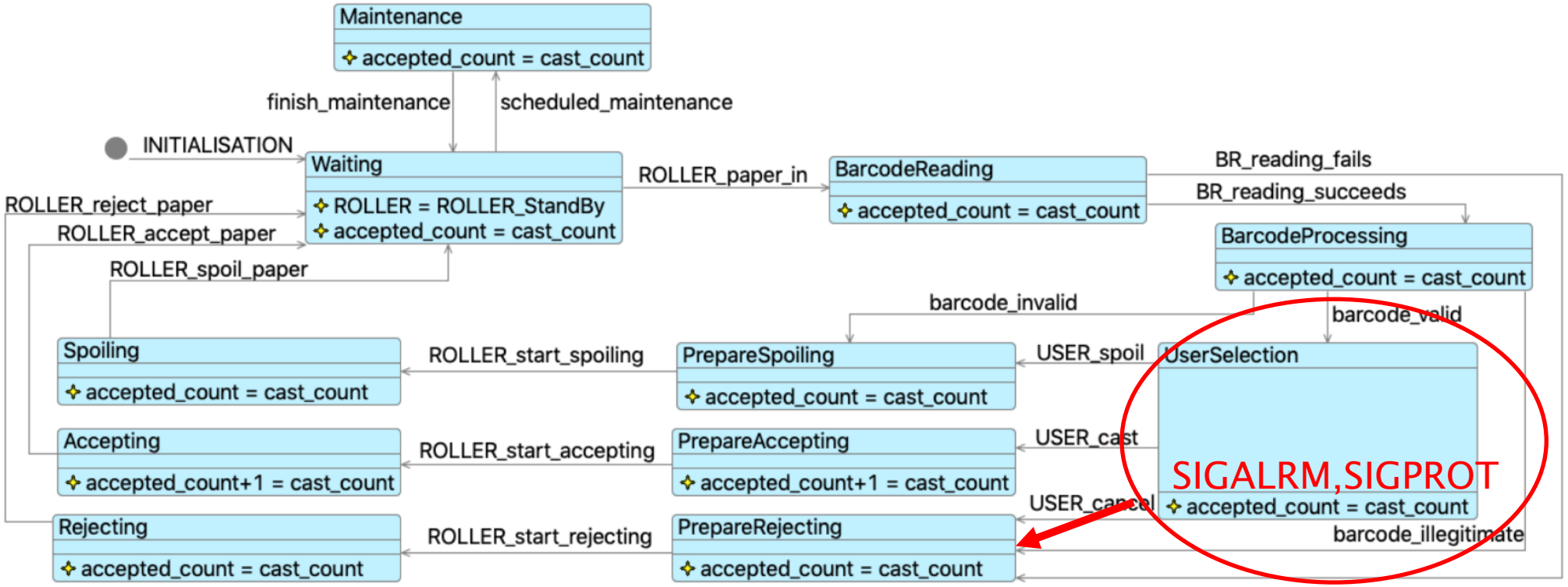
Exception	Signal	States	Handling	Rollback
Memory protection error	SIGPROT	BarcodeReading, BarcodeProcessing, UserSelection	reject ballot	-



Case Study – Exceptional Behaviour

University of Southampton

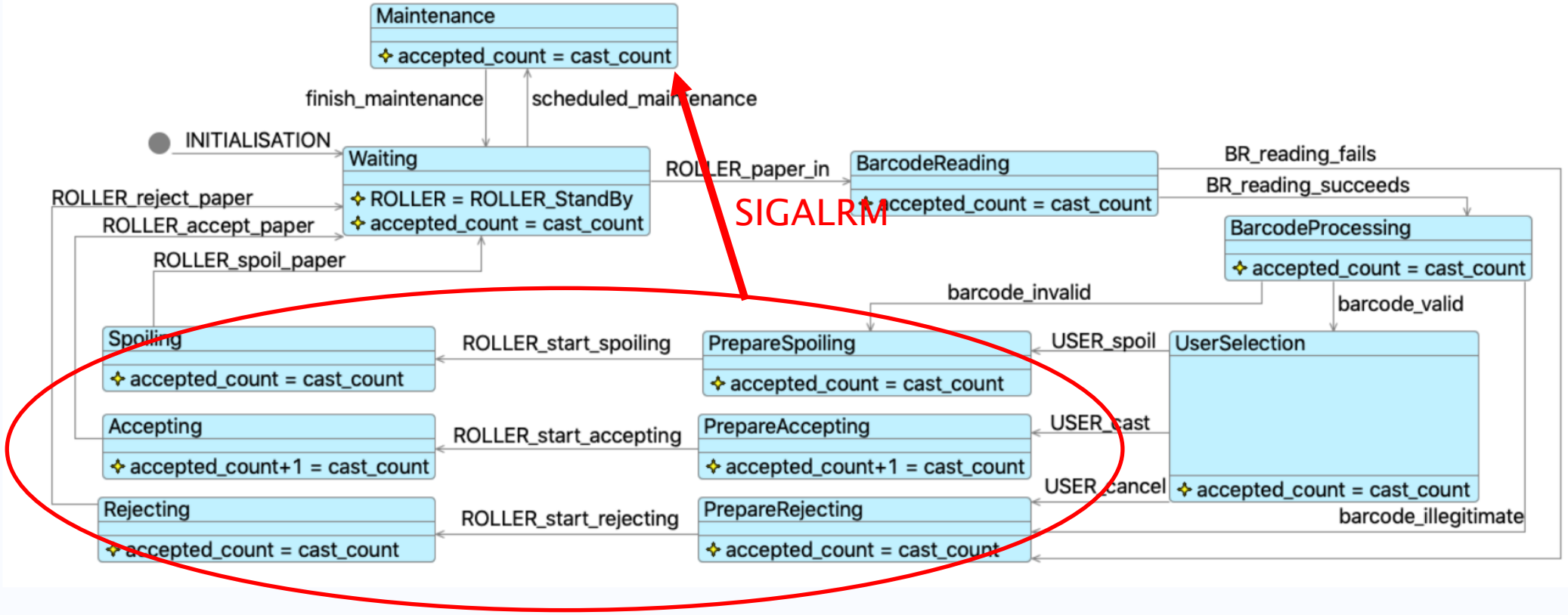
Exception	Signal	States	Handling	Rollback
User does not enter selection	SIGALRM	UserSelection	reject ballot	-



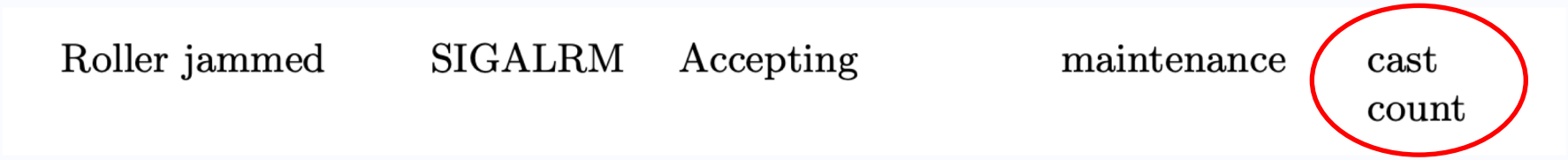
Case Study – Exceptional Behaviour

University of Southampton

Exception	Signal	States	Handling	Rollback
Roller jammed	SIGALRM	Accepting	maintenance	cast count
Roller jammed	SIGALRM	Spoiling, Rejecting	maintenance	-



Verifying Recovery – rollback actions



Need for rollback detected by provers

(initially omitted from model to illustrate the point)

- Proof obligations for exception handler event –
  - preservation of invariants concerning cast count = accepted count.
- Most discharged automatically
  - (after splitting the event and adding some theorems to distinguish cases)
- One case remains… s=SIGALRM, SBB=Accepting
  - corresponds to the case where we need to add a rollback of the cast count.
- *Event-B verification identifies any missing rollback actions and discovers the exact case where they are needed*.

Verifying Recovery – recovery states

University of Southampton

Exception	Signal	States	Handling	Rollback
Memory protection error	SIGPROT	BarcodeReading, BarcodeProcessing, UserSelection	reject ballot	-

- Proof obligations for exception handler event –
  - preservation of invariants concerning external machines after state change.

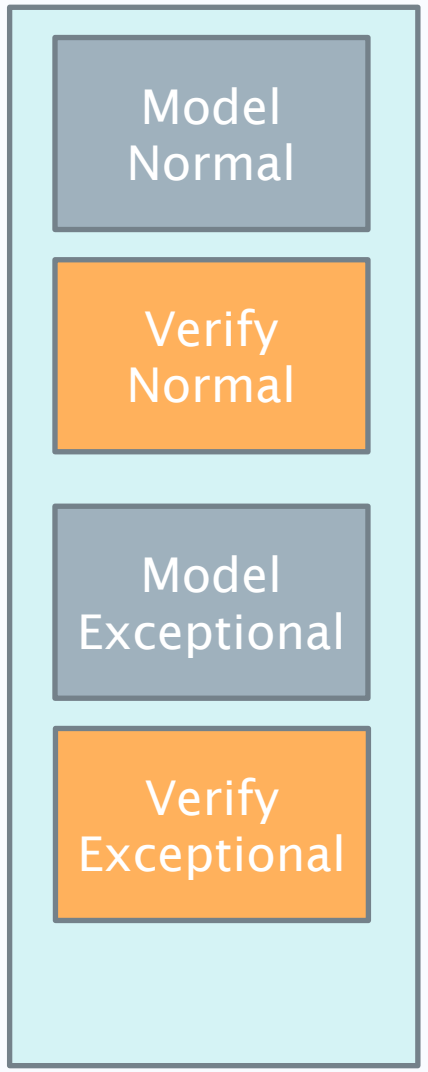
- Initially, we specified recovery from BarcodeReading to Waiting
  - The ROLLER would be left in the state ROLLER PaperIn violating the safety invariant.
  - We could not prove this unsafe design - counter-example using the ProB model checker.
  - Since the Roller is an external system it cannot be easily changed like the cast count.
    - Use 'PrepareRejecting' as recovery state - allows ROLLER sub-system to reject the paper.

- Similarly, for Roller timeout, maintenance state allows repair and manual paper removal
- *State invariants identify inappropriate handling recovery states*.

Conclusions

University of Southampton

- Initially we design *and verify* the system normal behaviour
  - States of the controller
  - Consistency with state of external machinery
  - Consistency with ancilliary variables (e.g. counters)
- Exceptional behaviour added later (although not a refinement)
  - Different types of exceptions
  - Recovery state
- Event-B verification detects
  - Inappropriate recovery states - external machinery in wrong state
  - Need for rollback - ancilliary variables in wrong state



Future Directions

University of Southampton

- Visualisation of exception handling on UML-B statemachines
  - Generation of Event-B exception handling events etc.
- Code generation including Exception handling – ‘C’
- Explore SIGPROT generated on Morrello processor
  - Capabilities
  - Detecting security violations

University of Southampton

QUESTIONS