IN4387 System Validation

Design & Verification of Controller for a Package Storage System

S. Balasubramanian, #0785610
Voudouris. P, #0788565
Gozek. E, #0786244

Eindhoven University of Technology
Department of Embedded Systems

October 5, 2012

Contents

Chapter 1

Introduction

The project described in this report discusses the design and verification of a small packet storage system. The packet storage system is inspired by the distributed controller of an operational product manufactured by Vanderlande, a Dutch company. The aim of this project is to understand and formulate the requirements of such a system, design the system and in the end verify that the system would meet the proposed requirements when in operation under any circumstance.

The packet storage system consists of two conveyor belts, one for receiving a packet into the system and the other for delivering the packet out of the system. The system consists of several racks where packets can be stored. The packets are stored by means of two elevator platforms (situated serially, one above the other). Moreover, there are five controllers in the system which run in parallel. Figure 1.1 shows the diagram of such a system. The controller C1 controls the input conveyor belt, controller C2 controls the output conveyor belt, conveyor C3 and C4 operate its respective elevators and controller C5 keeps track of all the information about racks, packet storage and its dispatch.

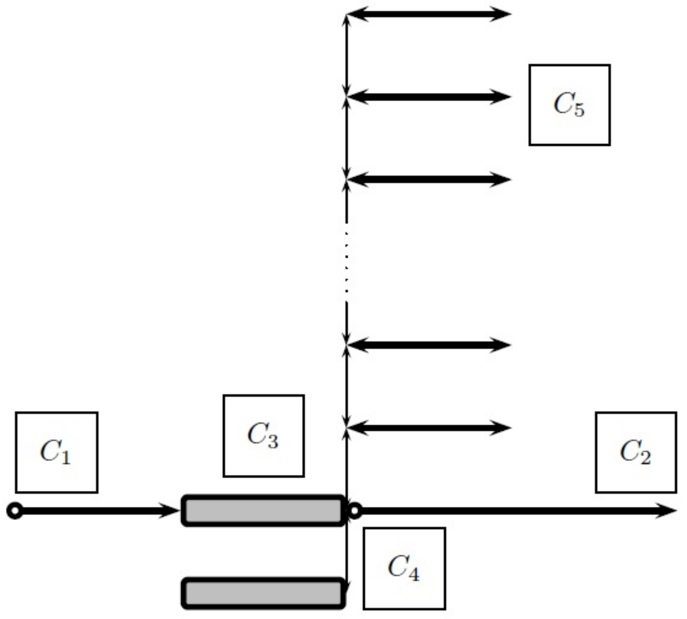

Figure 1.1: Packet storage system, courtesy [1]

There are however, certain constraints to be understood for the given system. For example, the number of packets on the conveyor belt, rack slots and elevator platform can carry only one packet at a time, etc. Further constraints are discussed in the next section under global requirements.

The sections described in the report are the global requirements, external interactions in the system, translated requirements, the architecture of the system transpired, discussion on controllers with its labelled transition diagram(LTS) and finally discuss the verification of its global requirements. The system is to be modelled to be intelligent to fulfil the basic requirements desired from such a system.

Chapter 2

Global requirements

Global Requirements

In the section, we describe the global requirements to required initially for the design of the controller:

1. Each elevator, rack and conveyor belt contains at most one packet.
2. Packet is exchanged only when the elevator platform is at the same level as that of a conveyor belt.
3. Packet is exchanged only when elevator platform is at the same level as that of a rack.
4. The two elevators cannot be at the same position.
5. The lower elevator must never pass the upper one.
6. Packets are always delivered in the same order as requested.
7. If a packet is ready to enter and there is a free position at the rack(s), it will be eventually accepted.
8. If a requested packet is in the system, it will be eventually delivered.
9. If a packet is unable to be located, a unique alarm must be generated.
10. The number of packets in the system can at most be equal to the number of racks.

Chapter 3

System interactions

This section lists the external and internal interactions for the packet storage system. These are the actions that are essential for the design of the system. The following describe in brief the external and internal interactions used in the system; the action, it parameters and its description.

External Interactions

The external interactions are listed under table **??**. These actions are visible to the outside world or the user controlling the system.

- **WantIn(id: packetID)** Environment requires a packet with packet id packetID to be inserted to the system. This is action is managed by controller C1.
- **AcceptPacket(id: packetID)** Controller C1 initiates the AcceptPacket action in order to accept the packet and put it on the conveyor.
- **RejectPacket(id: packetID)** This action is executed by controller C1 in order to inform the user that the packet with *packedID* has not been inserted into the system.
- **MoveElevator(eId: elevID,p: pos)** Controllers C3 and C4 initiate the moveElevator action to instruct the real hardware to move the elevator to the target position *p*.
- **WantOut(id: packeID)** User requests the packet with packet with an *id* to be delivered by the system. This action is initiated by controller C2.
- **PacketNotFound(id: packetID)** Controller C2 informs the user that the packet with a give *id* was not found.
- **DeliverPacket(id: packetID)** Controller C2 initiates DeliverPacket action to deliver the packet with an *id*.
- **ConvInToElev(eId: elevID)** With this action, controller C1 instructs the real hardware to transfer the packet from input conveyor to the required elevator.
- **ConvInToElev(eId: elevID)** Controllers C3 or C4 instruct its hardware to transfer the packet from the elevator platform to the rack. Note: We assume that elevator platform and a rack would have the same position during any exchange.
- **RackToElev(eId: elevID)** Controller C5 instructs the hardware to transfer the packet from rack to the required elevator.

- **ElevToConvOut(eId: elevID)** Controllers C3 or C4 instructs its hardware to transfer the packet from the required elevator platform to the output conveyor.
- **GetPosition (eId: elevID, p: pos)** Get the position *p* of an elevator with an identifier *id*.

Internal interactions

The internal interactions are listed under table **??**. These actions are hidden to the outside world or the user controlling the system and take place as interactions within/between individual controllers.

- **queryRackSpace(b: bool)** Queries if a position is available in racks. This is an internal action that is initiated by controller C1.
- **commPacketOnConveyor(id: packetID, cId: convID)** This is a synchronizing action whereby controller C1 informs C5 of the packet on conveyor. This action is completed with the acknowledgement from controller C5.
- **commOrderMoveElevator(e:ElevID, p: pos)** Controller C5 synchronizes with an elevator controller (C3/C4) to move it to a target position *p*.
- **commAckElevatorMoved(b: bool)** Elevator controller acknowledges controller C5 of the completion of elevator movement.
- **commConveyorToElev(eId: elevID, cId: convID)** Controller C5 informs C1 to move the conveyor belt and exchange the packet with the elevator with a given *elevID*.
- **commPacketOnElevAck(id: packetID)** Elevator controllers acknowledge controller C5 once packet is received by the elevator.
- **commElevToRack(eId: elevID)** Controller C5 communicates with elevator controllers in order to load to packet to the rack.
- **commPacketLoadedToRack(id: packetID)** Elevator controllers inform C5 that loading of the packet to the rack action is completed.
- **commRequestPacket(id: packetID)** A synchronization action between controllers C2 and C5 to request a packet with a given *id*.
- **commPacketExists(b: bool)** Synchronization action that governs the knowledge of the packet availability from rack controller C5 to C1.
- **commRackToElev(eId: elevID)** This action governs the communication of reception of packet from conveyor belt.
- **commElevToConveyor(eId: ElevID)** This action governs the communication of unloading a packet to the conveyor belt.
- **commElevToRack(eId: elevID)** Elevator communication with controller C5 after an outgoing transaction is complete.

External Interactions

Data types

The data types used are described in table 3.1.

Datatype	OfType	Range/Members
packetID	$Nat+$	1..N
elevID	$Nat+$	1, 2
pos	Int	-1..N
b	*bool*	true, false

Table 3.1: Data types in our design: Packet storage system

Chapter 4

Translated requirements

This section lists the requirements from section 2 in terms of interactions described in section 3.

1. *Each elevator, rack and conveyor belt contains at most one packet*

   Conveyor Belt

   - Whenever an AcceptPacket(id: packetID) action is performed it is not possible to perform another sequence of AcceptPacket(id: packetId) unless an ConvToElevator(e: elevID) action is performed(where, e can be one of the two elevators)
   - Whenever ElevToConvOut(e: elevID) action is performed it is not possible to perform another sequence of ElevToConvOut(e: elevID) unless a DeliverPacket(id: packetID) action is performed.

   Elevator

   - Whenever a ConvToElev(e: elevID) or RackToElev(e: elevID) action is performed it is not possible to perform another ConvToElev(e: elevID) or RackToElev(e: elevID) unless an ElevToRack(e: elevID) or ElevToConv(e: elevID) action is performed.

   Rack

   - Whenever an ElevToRack(p: pos) action is done it is not possible to perform another ElevToRack(p: pos) unless a RackToElev (p: pos) action is performed.

2. *Packet is exchanged only when the elevator platform is at the same level as that of a conveyor belt*

   - Whenever a ConvInToElev(e: elevID) or a ElevToConvOut(e: elevID) is done a MoveElevator(e :elevId, 0: pos) must be performed first.(Assumption: Input and output conveyor are at position zero).

3. *Packet is exchanged only when elevator platform is at the same level as that of a rack*

   - ElevatorToRack(e: elevID) action can only be performed after MoveElevator(e: elevID, p: pos) action.

4. *The two elevators cannot be at the same position*

   - The GetPosition(e1: elevId, p: pos) action and GetPosition(e2: elevID, p: pos) actions cannot occur simultaneously (from the two controllers C3/C4).

Note: The target position for both elevators has been indicated as p.

5. *The lower elevator must never pass the upper one*
   - Controller C4 cannot initiate the MoveElevator(e1: elevID, p1: pos) action when controller C3 is performs GetPosition(e2: elevID, p2: pos) action.
   - Controller C3 cannot perform MoveElevator(e2: elevID, p2: pos) action when controller C4 performs GetPosition(e1: elevID, p1: pos) action.
     Note: Here, p1<p2 and e1, e2 is the upper elevator and lower elevator, respectively.
6. *Packets are always delivered in the same order as requested*
   - Reception of latest requested packet cannot happen before the reception of the previously 'unprocessed' requested packet.
7. *If a packet is ready to enter and there is a free position at the rack(s), it will be eventually accepted*
   - If there is no free position in racks, packet must not be accepted.
   - If there is a free position in the racks and the packet is ready to enter, it must be eventually stored.
8. *If a requested packet is in the system, it will be eventually delivered*
   - If a packet is requested and it is in the system, it must be delivered eventually.
   - If a packet is requested and it is not in the system, no packet is delivered.
9. *If a packet is unable to be located, a unique alarm must be generated*
   - If the requested packet is not located in the racks, a alarm is generated.
10. *The number of packets in the system can at most be equal to the number of racks*
    - The number packets in the racks cannot be greater than the number of racks in the system.

Chapter 5

Architecture

Chapter 6

Modelling the controller

Chapter 7

Verification

Chapter 8

Experimental results

Chapter 9

Conclusions and recommendations

Bibliography

[1] Project statement.

Appendix A

Source Code Structure