Delft University of Technology

Faculty of Electrical Engineering,
Mathematics and Computer Science

System Validation Project

Authors:

Kaustubh Agarwal

Ana Aldescu

Aniket Ashwin Samant

Şerban Vădineanu

September 2018

Contents

Chapter 1

Introduction

The System Validation course of TU Delft proposes a project assignment concerning the design of a controller for a small distributed embedded system. The goal of the assignment is to design, model, and verify a control system for a simplified machine that prepares wafers for the production of microchips. Following the different stages of this project, we should be able to properly formulate system requirements and represent the model using labeled transition systems. This document presents the constituent phases of the project. Chapter 2 presents the requirements of the entire system that are formally verifiable. In chapter 3 we present the interactions taking place between the components defined in chapter 2. The architecture of the system is shown in chapter 4. Chapter 6 presents the modeled behaviour of all the controllers presented in the architecture. In chapter 5 the model is verified with the translated requirements presented in chapter 2. Conclusions and other remarks are presented in chapter 8.

For a better understanding of the system's operation (in the semantic domain), this chapter also presents a list of physical components present in the system that work together to satisfy the requirements, which will be presented in the later chapters. Moreover, an informal description of the sequences of actions performed by the various components is also presented before proceeding towards the formal requirements, modelling, and verification.

1.1 System Components

- An UV lamp L - projects the wafer when placed underneath it
- A vacuum chamber
- Two airlocks A1, A2
- Two sets of doors, one set per airlock
  - Inner doors: DI1, DI2
  - Outer doors: DO1, DO2
- Three robots for moving wafers around

  - Outside the vacuum chamber: R1, R2
  - Inside the vacuum chamber: R3
- Two sets of wafer stacks where wafers are loaded
  - Input stacks: I1, I2
  - Output stacks: O1, O2
- One lamp sensor - Lamp has wafer/does not have a wafer underneath
- Two airlock sensors - Corresponding airlock has/does not have a wafer
- Two input stack sensors - Corresponding stack is empty/not empty
- Two output stack sensors - Corresponding stack is full/not full

1.2 Sequences

We have identified two main sequences of the process. The first sequence describes the steps needed to move a blank wafer from the input stack(s) to the lamp, while the second sequence describes how a wafer reaches the output stack(s) after being projected by the lamp.

1.2.1 Wafer to lamp

1. Outer robot picks up a wafer from the corresponding input stack
2. Airlock's outer door opens
3. Robot drops wafer inside airlock
4. Airlock's outer door closes
5. Airlock's inner door opens
6. Inner robot picks up wafer from airlock
7. Inner robot drops wafer to lamp
8. Lamp turns on and projects the wafer

1.2.2 Wafer to output stack

1. Lamp turns off
2. Inner robot picks up wafer from lamp
3. Inner robot drops wafer inside airlock
4. Airlock's inner door closes
5. Airlock's outer door opens

6. Outer robot picks up wafer from airlock
7. Outer robot drops wafer in the corresponding output stack

Chapter 2

Requirements

The first step of the design process is to identify the requirements for the wafer projecting system we are considering. The components described in the previous chapter will have to follow some defined set of behaviors so as to meet all the requirements of the system, as will be presented hence.
The requirements are presented in the form of safety requirements and liveness requirements, and are verifiable using formal logic.

2.1 Safety Requirements

These requirements ensure that the system is always in a stable state, that is, each one of them needs to be satisfied by the components for us to say that the system is behaving normally; the violation of even one of these requirements would mean that the system has malfunctioned.

1. Airlocks:
   (a) DI1/DI2 can only be opened if DO1/DO2 is closed.
   (b) DO1/DO2 can only be opened if DI1/DI2 is closed.
2. Lamp:
   (a) Each wafer dropped below the light can be projected only once before being picked up.
3. Robots:
   (a) R1/R2 can only drop a wafer into O1/O2 if the stack is not full.
   (b) R1/R2 can only pick-up a wafer from I1/I2 if the stack is not empty.
   (c) R1/R2 can only pick-up a wafer from A1/A2 if DO1/DO2 is open.
   (d) R1/R2 can only drop a wafer in A1/A2 if DO1/DO2 is open.
   (e) R3 can only pick-up a wafer from A1/A2 if DI1/DI2 is open.

(f) R3 can only drop a projected wafer in A1/A2 if DI1/DI2 is open.

(g) After picking-up a wafer from A1/A2, R3 can only drop it into the lamp.

(h) After picking-up a wafer from the lamp, R3 can only drop it into A1/A2.

(i) R1 can pick-up wafers from I1 and drop them only into A1.

(j) R2 can pick-up wafers from I2 and drop them only into A2.

(k) R1 can pick-up wafers from A1 and drop them only into O1.

(l) R2 can pick-up wafers from A2 and drop them only into O2.

(m) A wafer can only be placed into the output stack only if it was projected by the lamp.

(n) The lamp can project a wafer only once.

2.2 Liveness Requirements

The following liveness requirements ensure the movement of the wafers throughout the system.

1. From the input stack, a wafer should always be able to reach the lamp.
2. From the lamp, a wafer should always be able to reach the output stack.
3. The system must be deadlock-free.

Chapter 3

Interactions

This chapter presents the interactions that take place between the components we have described previously.

3.1 Outer Robot Controllers

The outer robot controllers interact with the input stack controllers, the output stack controllers, and the airlock controllers (each outer robot controller interacts *only* with its corresponding controller).

s_robotState(rID, bool, l):	(rID: R1, R2 – Bool – l: I1, I2, O1, O2, A1, A2, L)	Sends the state of the robot to other controllers

3.2 Input Stack Controllers

The input stack controllers determine whether the input stacks are empty or not, and interact with the robot controllers to indicate whether wafers can be picked up by the robots (if the stacks aren't empty) or not. If not, an external interaction is needed to symbolize that the stacks are refilled.

s_inputStackState(id, state):	(id: I1 .. I2 – state: EMPTY/NEMPTY)	Provides the state of the input stack as empty/not.

3.3 Output Stack Controllers

The output stack controllers determine whether the output stacks are full or not, and interact with the robot controllers to indicate whether wafers can be dropped by the robots (if the stacks aren't full) or not. If not, an external interaction is needed to symbolize that the stacks are emptied to allow wafers to be dropped.

s_outputStackState(id, state):	(id: O1, O2) – state: FULL/NFULL)	Provides the state of the output stack as full/not.

3.4 Airlock Controllers

The airlock controllers control the airlock doors and provide information based on the status of the wafer sensors. Each airlock controller interacts with the R3 controller, and the corresponding outer robot controller to transfer wafers into and out of the vacuum chamber.

s_airlockState(id, hasWafer	(id: A1, A2 – hasWafer: Bool,) innerDoor, outerDoor: OPEN/CLOSED)	Send the state of the airlock and its doors.

3.5 Inner Robot Controller

The inner robot (R3) controller controls the movement of the robot to/from the airlocks from/to the lamp, and the picking up and dropping of wafers in the three locations. It interacts with the lamp controller and the airlock controllers to move wafers around and get them projected and transferred to the airlocks.

s_robotState(rID, bool, l):	(rID: R1, R2 – Bool – l: A1, A2, L)	Sends the state of the robot to other controllers

3.6 Lamp Controller

The lamp controller interacts with the R3 controller to receive one wafer at a time, and makes the lamp project the wafer before signalling to the R3 controller to take it out, and also the turning on and off of the lamp.

s_lampState(lampState, hasWafer, waferState):	(lampState: ON, OFF – hasWafer : Bool – waferState: (UN)PROJECTED)	Sends the state of the lamp as on/off, and the wafer info

3.7 External Interactions

The following actions can be performed by a user upon the components of the system. The system is not expected to have any user input, but some external interactions are needed in case the airlock doors get stuck, an input stack becomes empty, or an output stack becomes full.

user_fillStack(a):	(a: I1, I2)	Fill the corresponding stack with wafers.
user_emptyStack(a):	(a: O1, O2)	Empty the corresponding stack.
user_repairDoor(a):	(a: DO1, DO2, DI1, DI2)	Repair the corresponding door.

Chapter 4

Architecture

The controller system's architecture has been modelled such that the safety and liveness requirements are always met, by using the various sensors, actuators, and other components that are present in the system.

The main assumptions made here for simplicity are as follows. The robots perform the instructed actions without any error. The lamp performs the projecting process without malfunctioning. The other components may require the user to act upon them based on certain sensor data and the corresponding user inputs described as interactions in chapter 3.

The architecture of the system is represented graphically using the Figure 7.1. It consists of several components working in parallel to meet the requirements stated in chapter 2.

For the most part, the system is expected to function without the need for any user input, but certain cases have been identified in which "external interaction" is needed to meet the system's liveness requirements. These are indicated by green arrows in the diagram. There are various controllers for the components, and communication between the controllers is represented using blue arrows. These interactions take place between the controllers continuously and ensure that the system runs automatically.

The controllers send the state of their associated components and poll for the states of other components as required, and make state transitions by performing the appropriate actions. The state transitions should not end up causing a deadlock within the overall system, at any cost.

The system never reaches a state of "completion" since it is expected to run continuously till one of the conditions requiring an external interaction is met (see chapter 2). Theoretically, if the user continually replenishes the input stack with new wafers and clears the output stack at the same rate, and if the airlock doors do not malfunction, the system can run for an indefinite period of time.

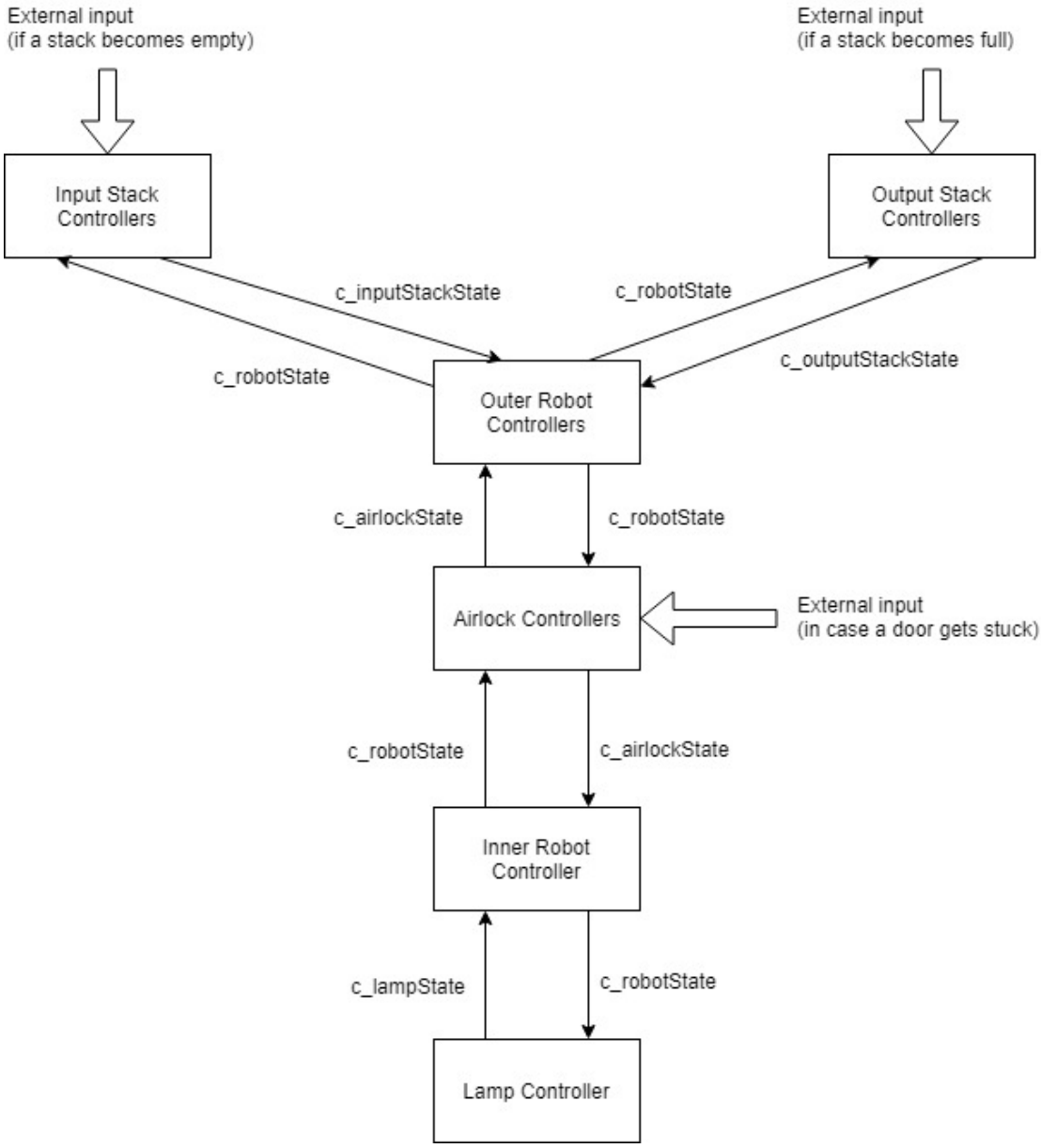

Figure 4.1: System architecture diagram

Chapter 5

Translated Requirements

The requirements stated in chapter 2 are translated to interactions between the controllers, and the resulting model is verified using μ-calculus. This chapter lists the requirements translated into the corresponding μ-calculus representation, taking one requirement at a time.

5.1 Translated safety requirements

1. Airlocks:
   (a) DI1/DI2 can only be opened if DO1/DO2 is closed.
      i. $[true*]\forall A : AirlockID[sense\_doorClose(matchOutputDoor(A), UNDETECTED)$
      $.a\_openDoor(matchInputDoor(A))]false$
      ii. $[true*]\forall A : AirlockID.[a\_openDoor(matchOutputDoor(A))$
      $.(!(a\_closeDoor(matchOutputDoor(A)).$
      $sense\_doorClose(matchOutputDoor(A), DETECTED))) * .$
      $a\_openDoor(matchInputDoor(A))]false$
      iii. $[true*]\forall A : AirlockID.[sense\_doorOpen(matchOutputDoor(A), DETECTED)$
      $.a\_openDoor(matchInputDoor(A))]false$
   (b) DO1/DO2 can only be opened if DI1/DI2 is closed.
      i. $[true*]\forall A : AirlockID.[sense\_doorClose(matchInputDoor(A), UNDETECTED)$
      $.a\_openDoor(matchOutputDoor(A))]false$
      ii. $[true*]\forall A : AirlockID.[a\_openDoor(matchInputDoor(A))$
      $.(!(a\_closeDoor(matchInputDoor(A))$
      $.sense\_doorClose(matchInputDoor(A), DETECTED))) *$
      $.a\_openDoor(matchOutputDoor(A))]false$

2. Lamp:
   (a) Each wafer dropped below the light can be projected only once before being picked up.
      i. $[true*][a\_project.(!a\_pickUpWafer(R3, LAMP)) * .a\_project]false$
      ii. $[true*][a\_project.a\_project]false$
3. Robots:
   (a) R1/R2 can only drop a wafer into O1/O2 if the stack is not full.
      i. $[true*]\forall R : RobotID.val(R == R1||R == R2) =>$
      $[sense\_outputStack(matchRobotOutputStack(R), FULL)$
      $.a\_dropWafer(R, matchOutputStackLoc(R))]false$
      ii. $[true*]\forall R : RobotID.val(R == R1||R == R2) =>$
      $[(!sense\_outputStack(matchRobotOutputStack(R), NFULL)) *$
      $.a\_dropWafer(R, matchOutputStackLoc(R))]false$
   (b) R1/R2 can only pick-up a wafer from I1/I2 if the stack is not empty.
      i. $[true*]\forall R : RobotID.val(R == R1||R == R2) =>$
      $[sense\_inputStack(matchRobotInputStack(R), EMPTY)$
      $.a\_pickUpWafer(R, matchInputStackLoc(R))]false$
      ii. $[true*]\forall R : RobotID.val(R == R1||R == R2) =>$
      $[(!sense\_inputStack(matchRobotOutputStack(R), NEMPTY)) *$
      $.a\_pickUpWafer(R, matchInputStackLoc(R))]false$
   (c) R1/R2 can only pick-up a wafer from A1/A2 if DO1/DO2 is open.
      i. $[true*]\forall R : RobotID.val(R == R1||R == R2) =>$
      $[sense\_doorState(matchRobotDoor(R), CLOSED)$
      $.a\_pickUpWafer(R, matchAirlockLoc(R))]false$
      ii. $[true*]\forall R : RobotID.val(R == R1||R == R2) =>$
      $[(!sense\_doorState(matchRobotDoor(R), OPENED)) *$
      $.a\_pickUpWafer(R, matchAirlockLoc(R))]false$
   (d) R1/R2 can only drop a wafer in A1/A2 if DO1/DO2 is open.
      i. $[true*]\forall R : RobotID.val(R == R1||R == R2) =>$
      $[sense\_doorState(matchRobotDoor(R), CLOSED)$
      $.a\_dropWafer(R, matchAirlockLoc(R))]false$
      ii. $[true*]\forall R : RobotID.val(R == R1||R == R2) =>$
      $[(!sense\_doorState(matchRobotDoor(R), OPENED)) *$
      $.a\_dropWafer(R, matchAirlockLoc(R))]false$

(e) R3 can only pick-up a wafer from A1/A2 if DI1/DI2 is open.

i. $[true*][sense\_doorState(matchRobotDoor(R3), CLOSED)$
$.a\_dropWafer(R3, matchAirlockLoc(R3))]false$

ii. $[true*][(!sense\_doorState(matchRobotDoor(R3), OPENED)) *$
$.a\_dropWafer(R3, matchAirlockLoc(R3))]false$

(f) R3 can only drop a projected wafer in A1/A2 if DI1/DI2 is open.

i. $[true*][sense\_doorState(matchRobotDoor(R3), CLOSED)$
$.a\_pickUpWafer(R3, matchAirlockLoc(R3))]false$

ii. $[true*][(!sense\_doorState(matchRobotDoor(R3), OPENED)) *$
$.a\_pickUpWafer(R3, matchAirlockLoc(R3))]false$

(g) After picking-up a wafer from A1/A2, R3 can only drop it into the lamp.

i. $[true*][(!a\_pickUpWafer(R3, matchAirlockLoc(R3))) *$
$.a\_dropWafer(R3, LAMP)]false$

ii. $[true*][a\_pickUpWafer(R3, matchAirlockLoc(R3))$
$.(!a\_dropWafer(R3, LAMP))*]false$

(h) After picking-up a wafer from the lamp, R3 can only drop it into A1/A2.

i. $[true*][(!a\_pickUpWafer(R3, LAMP)) *$
$.a\_dropWafer(R3, matchAirlockLoc(R3))]false$

ii. $[true*][a\_pickUpWafer(R3, LAMP)$
$.(!a\_dropWafer(R3, matchAirlockLoc(R3)))*]false$

(i) R1 can pick-up wafers from I1 and drop them only into A1.

i. $[true*][a\_pickUpWafer(R1, I1).!a\_dropWafer(R1, A1)]false$

ii. $[true*][!a\_pickUpWafer(R1, I1).a\_dropWafer(R1, A1)]false$

(j) R2 can pick-up wafers from I2 and drop them only into A2.

i. $[true*][a\_pickUpWafer(R2, I2).!a\_dropWafer(R2, A2)]false$

ii. $[true*][!a\_pickUpWafer(R2, I2).a\_dropWafer(R2, A2)]false$

(k) R1 can pick-up wafers from A1 and drop them only into O1.

i. $[true*][a\_pickUpWafer(R1, A1).!a\_dropWafer(R1, O1)]false$

ii. $[true*][!a\_pickUpWafer(R1, A1).a\_dropWafer(R1, O1)]false$

(l) R2 can pick-up wafers from A2 and drop them only into O2.

i. $[true*][a\_pickUpWafer(R2, A2).!a\_dropWafer(R2, O2)]false$

ii. $[true*][!a\_pickUpWafer(R2, A2).a\_dropWafer(R2, O2)]false$

(m) A wafer can only be placed into the output stack only if it was projected by the lamp.

i. $[true*]\forall R : RobotID.val(R == R1||R == R2) =>$
$[a\_pickUpWafer(R, matchRobotInputStack(R))$
$.(!a\_project) * .a\_dropWafer(R, matchRobotOutputStack(R))]false$

ii. $[true*]\forall R : RobotID.val(R == R1||R == R2) =>$
$[a\_pickUpWafer(R, matchRobotInputStack(R))$
$.a\_dropWafer(R, matchRobotOutputStack(R))]false$

(n) The lamp can project a wafer only once.

i. $[true*][sense\_lamp(lampState, YES, waferState)$
$.a\_dropWafer(R3, LAMP)]false$

ii. $[true*][(!sense\_lamp(lampState, NO, waferState)) *$
$.a\_dropWafer(R3, LAMP)]false$

5.2 Translated liveness requirements

1. From the input stack, a wafer should always be able to reach the lamp.

   (a) $[true*]\forall R : RobotID.val(R == R1||R == R2) =>$
   $[a\_pickUpWafer(R, matchInputStackLoc(R))] < true * .a\_project > true$

2. From the lamp, a wafer should always be able to reach the output stack.

   (a) $[true*]\forall R : RobotID.val(R == R1||R == R2) =>$
   $[a\_project] < true * .a\_dropWafer(R, matchOutputStackLoc(R)) > true$

3. The system must be deadlock-free.

   (a) $[true*] < true > true$

Chapter 6

System Modelling

The system has been modelled using controllers for each of the components. To avoid duplication of code, the two input stacks have the same controller code, and it is just that the two controllers are distinguished by means of input stack IDs. Likewise, the controllers for both output stacks, the controllers for the two outer robots, and the controllers for the two airlocks have the same code in each case, and the respective controllers are differentiated using output stack IDs, robot IDs, and airlock IDs respectively.

Each controller has a set of states defined by variables as required to determine the uniqueness of the states. The controller communicates with one or more controllers and exchanges state information so as to make transitions by means of actions based on the state of the other concerned controllers' states. For instance, the outer robot controller changes its state from one having *hasWafer* as `true` to one having it as `false` based on the airlock controller's state (to get its sensor information) after the robot drops its wafer.

Since there are lots of "IDs" involved, lots of maps have been used to *match* the various IDs across controllers (for instance, when referring to interactions between robot 1 and airlock 1, the IDs will always be matched as R1 -¿ A1 or A1 -¿ R1.)

The `mcrl22lps` command is used to generate the lps for the mclr2 code, and is translated to an lts using the `lps2lts` command. Given the number of parallel components, it is difficult to get a good understanding of the system based on its LTS, and hence the `ltsview` command is used to visualize the system and its states, and most importantly, to ensure that no deadlocks are present.

Chapter 7

Verification

The system can be verified through a visual check, and through μ-calculus formulas. `mcf` files are used to verify the requirements stated previously.

7.1 Tools used

The MCRL2 version used for performing the modelling and verification of the system is 201808.0. mcrl22lps is used for generating the lps file, and the lps2lts command is used for generating the lts file.

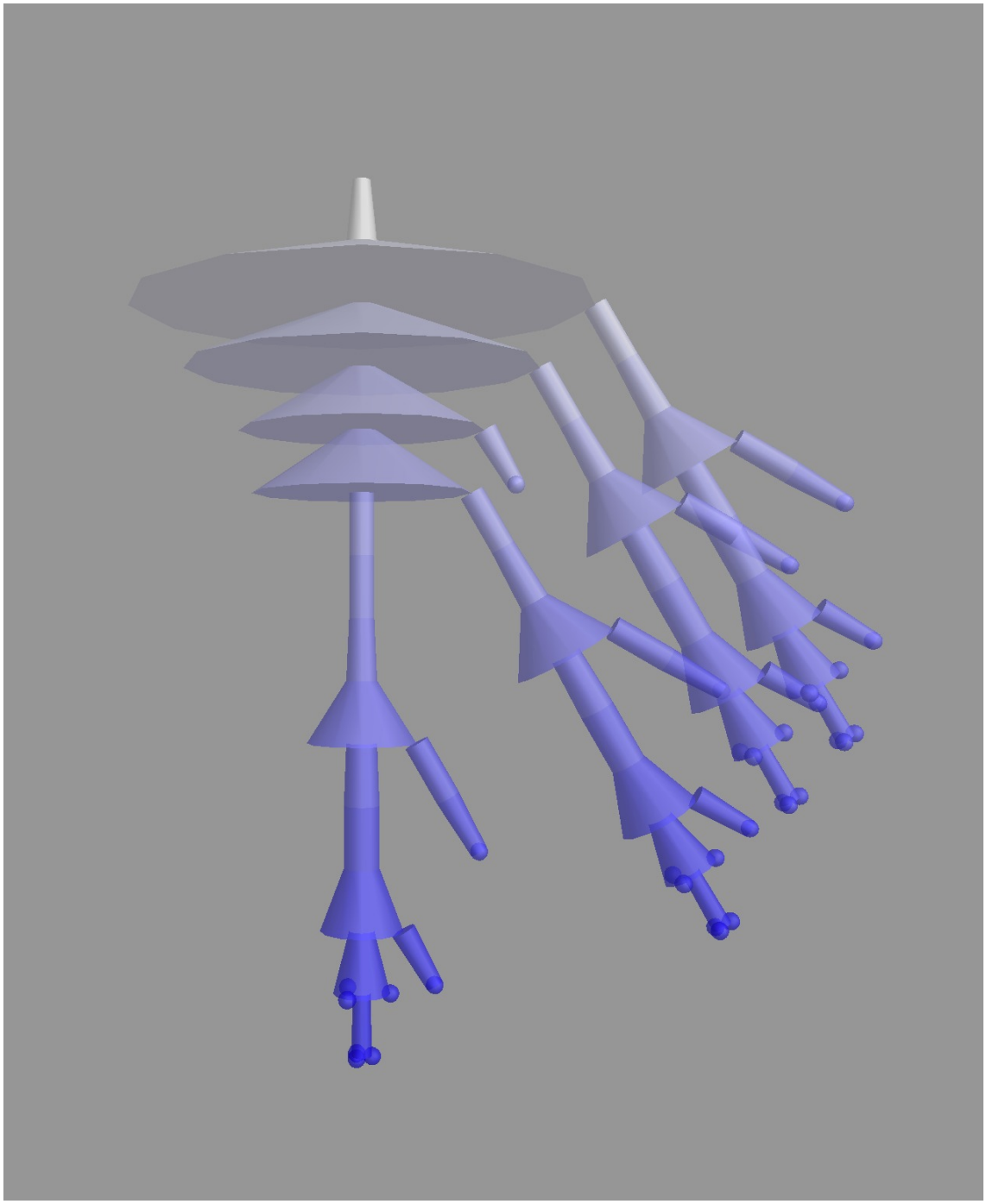

Figure 7.1: LTS view

7.2 Deadlocks

The ltsview generated to represent the system visually shows that

Chapter 8

Conclusions