6SENG002W Concurrent Programming

FSP Process Composition Analysis & Design Form

Name	Hasal Fernando
Student ID	w1697758
Date	20/12/2020

1. FSP Composition Process Attributes

Attribute	**Value**
Name	SHARED_PRINTER_SYSTEM
Description	This composite process models a scenario involving a printer, a technician and two students. Students are using the printer to print documents while the technician (t) is trying to refill the papers when it runs out of papers to print. Student 1 (st1) prints 3 documents while Student 2 (st2) prints 2 documents.
Alphabet (Use LTSA's compressed notation, if alphabet is large.)	{{st1, st2}.{{acquire, emptyAlert, notEmpty}, print[1..3], printerCheck.{acquire, release}, {release, terminate}}, t.{{acquire, emptyAlert, notEmpty}, print[1..3], printerCheck.{acquire, release}, {refill, release}}}
Sub-processes (List them.)	PRINTER, STUDENT, TECHNICIAN
Number of States	124
Deadlocks (yes/no)	No Deadlocks.
Deadlock Trace(s) (If applicable)	None.

2. FSP "main" Program Code

The code for the parallel composition of all of the sub-processes and the definitions of any constants, ranges & process labelling sets used. (Do not include the code for the other sub-processes.)

FSP Program:

```
set USERS = { st1, st2, t }

//SHARED PRINTER COMPOSITE PROCESS
|| SHARED_PRINTER_SYSTEM = ( st1: STUDENT(3) || st2: STUDENT(2) ||
t: TECHNICIAN || USERS :: PRINTER ) / {
   t.acquire / { st1.t.acquire, st2.t.acquire, t.t.acquire },
   t.refill / { st1.refill, st2.refill }
} .
```

3. Combined Sub-processes

(Add rows as necessary.)

Process	Description
STUDENT(3)	Represents student 1 who is trying to print 3 documents.
STUDENT(2)	Represents student 2 who is trying to print 2 documents.
TECHNICIAN	Represents technician who is repeatedly checking whether printer has run out of papers and trying to refill papers to maximum capacity (3 papers), when the printer runs out of papers to print.
PRINTER	Represents the printer which can hold maximum of 3 papers in its paper tray. Printer can be accessed by both students and the technician.

4. Analysis of Combined Process Actions

- **Synchronous** actions are performed by at least two sub-process in the combination.
- **Blocked Synchronous** actions cannot be performed, since at least one of the sub-processes cannot preform them, because they were added to their alphabet using alphabet extension.
- **Asynchronous** actions are preformed independently by a single sub-process.

Group actions together if appropriate, for example if they include indexes,
e.g. in[0], in[1], …, in[5] as in[1..5].

(Add rows as necessary.)

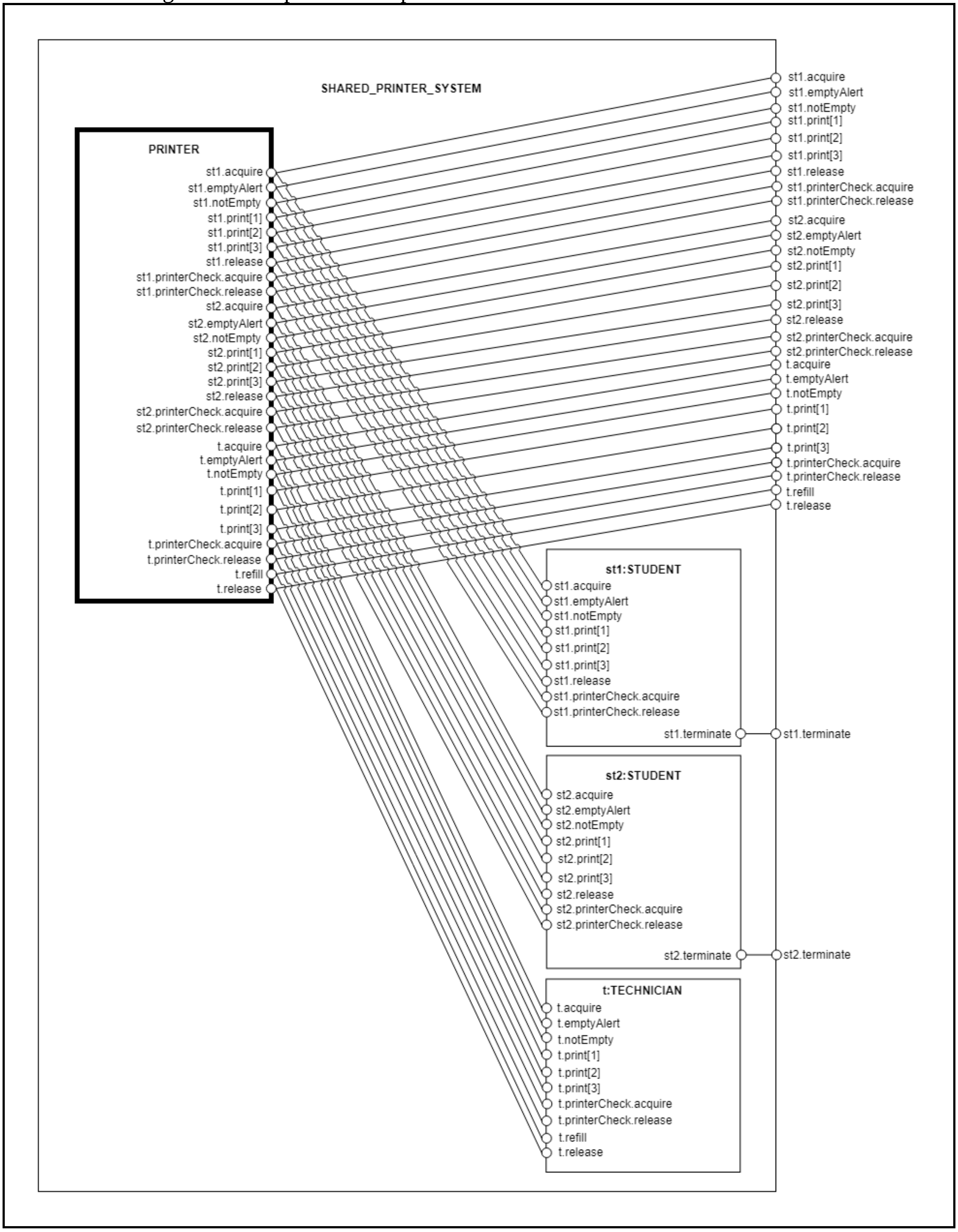
Synchronous Actions	**Synchronised by Sub-Processes (List)**
st1.acquire, st1.print[1..3], st1.release	STUDENT(3), PRINTER
st2.acquire, st2.print[1..3], st2.release	STUDENT(2), PRINTER
t.printerCheck.acquire, t.notEmpty, t.printerCheck.release, t.emptyAlert, t.acquire, t.refill, t.release	TECHNICIAN, PRINTER

Blocked Synchronous Actions	**Synchronising Sub-Processes**	**Blocking Sub-Processes**
t.print[1..3]	TECHNICIAN, PRINTER	TECHNICIAN
st1.printerCheck.acquire, st1.emptyAlert, st1.notEmpty st1.printerCheck.release	STUDENT(3), PRINTER	STUDENT(3)
st2.printerCheck.acquire, st2.emptyAlert, st2.notEmpty st2.printerCheck.release	STUDENT(2), PRINTER	STUDENT(2)

Sub-Process	**Asynchronous Actions (List)**
STUDENT(3)	st1.terminate
STUDENT(2)	st2.terminate

5. Parallel Composition Structure Diagram

The structure diagram for the parallel composition.



SHARED_PRINTER_SYSTEM
PRINTER
st1.acquire
st1.emptyAlert
st1.notEmpty
st1.print[1]
st1.print[2]
st1.print[3]
st1.release
st1.printerCheck.acquire
st1.printerCheck.release
st2.acquire
st2.emptyAlert
st2.notEmpty
st2.print[1]
st2.print[2]
st2.print[3]
st2.release
st2.printerCheck.acquire
st2.printerCheck.release
t.acquire
t.emptyAlert
t.notEmpty
t.print[1]
t.print[2]
t.print[3]
t.printerCheck.acquire
t.printerCheck.release
t.refill
t.release
st1:STUDENT
st1.acquire
st1.emptyAlert
st1.notEmpty
st1.print[1]
st1.print[2]
st1.print[3]
st1.release
st1.printerCheck.acquire
st1.printerCheck.release
st1.terminate
st1.terminate
st2:STUDENT
st2.acquire
st2.emptyAlert
st2.notEmpty
st2.print[1]
st2.print[2]
st2.print[3]
st2.release
st2.printerCheck.acquire
st2.printerCheck.release
st2.terminate
st2.terminate
t:TECHNICIAN
t.acquire
t.emptyAlert
t.notEmpty
t.print[1]
t.print[2]
t.print[3]
t.printerCheck.acquire
t.printerCheck.release
t.refill
t.release