Malaysian Journal of Computer Science, Vol. 14 No. 2, December 2001, pp. 81-94

SPECIFYING CONCURRENT CONTROLLER OF PRODUCTION CELL USING THE NOTATION OF SHARED STATE AND EVENTS OF DURATION CALCULUS

Rusdi Md. Aminuddin
School of Information Technology
Northern University of Malaysia
06010 Sintok, Kedah, Malaysia
email: rusdi@cs.usm.my

He Jifeng
International Institute for Software Technology
United Nation University (UNU/iist)
P.O Box 3058, Macau
email: jifeng@iist.unu.edu

Rosni Abdullah
School of Computer Science
Malaysia University of Science
Minden 11800, Penang, Malaysia
email: rosni@cs.usm.my

ABSTRACT

This report presents a method to specify a set of controllers for a Robotics production cell using the Duration Calculus. The case study is adopted from a report by Claus Leverentz on specifying a real metal processing plant in Karlsruhe using other formal and semi-formal methods. Our contribution to this case study aims at illustrating the methodology associated with the concept of shared state model and events for describing and specifying synchronised controllers. We use the notion of state to describe and model the sensors and actuators. Next, we apply the leads-to operator to list the assumptions which are aspects of the behaviour of the plant that cannot be controlled by the controllers. We then show how one can specify each controller using leads-to and state notation as shared variables for synchronising the interaction of the controllers. Specification are structured modularly according to the physical structure of the system.

Keyword: ***Formal Method, Duration Calculus, Concurrent Controllers, and Production Cell***

1.0 INTRODUCTION

Duration Calculus (abbreviated by DC) is one of the formal specification languages used to specify a real-time or a reactive system such as production cell. The DC represents a logical approach for formal design of real-time systems, where real numbers are used to model time, and Boolean-valued functions over time are used to model states and events of the real-time system [1]. Since its introduction, DC has been applied to many case studies and it has been extended in several directions.

The case study, "Control Software for an Industrial Production Cell", is taken from [4] which reports a study by Claus Leverentz. The study was done as one of the two major case studies of the KorSo project [6]. Both case studies have common primary objectives to show the usefulness of formal methods for critical software systems and to prove their applicability to the real-world examples. The "Production Cell" case study focuses on comparing different approaches of formal and semi-formal software construction methods developed inside and outside the KorSo project, and checking their suitability for the class of problems represented by the production cell. This case study has been specified using different approaches namely ESTEREL, LUSTRE, StateCharts, SDL, KIV, Tatzelvum, RAISE, Deductive Synthesis, FOCUS, SPECTRUM, TROLL light, Eiffel and Modula-3.

We take another approach using the notion of state and leads-to operator of Duration Calculus to specify this system which consists of many components or controllers. We also adopt a shared state in support of synchronisation.

We proceed as follows: in section two, we briefly present the basic formalism of Duration Calculus. Section three gives the overall description of Industrial Production Cell. Section four shows the description of each component of the plant and how we use the notion of state to model the input (i.e. the sensor) and the output (i.e. the actuators) of the components. The aspect of the behaviour of the plant that cannot be controlled by the controllers are listed as assumptions in section five. The specification of each controller appears in section six and we finish the paper with some conclusions in section seven.

2.0 BASIC FORMALISM OF DURATION CALCULUS

The Duration Calculus (abbreviated DC) is based on Interval Temporal Logic [1]. The notion *state* is used to model the behaviour of real-time systems. A *Boolean State Model* of a real-time system is a set of Boolean valued functions over time: *Time* {0,1}, where *Time* is the set of the real numbers. Each Boolean valued function, is also called a *Boolean state* (or simply a *state*) of the system, is a *characteristic* function of a *specific aspect* of the system behaviour, and the whole set of Boolean valued functions characterise all the concerned aspects of the behaviour.

The notion of *state duration* is an essential measurement of the behaviour of real-time systems. The duration of a Boolean state over a time interval is the accumulated time in which the state is present in the interval. Let *P* å *Time* {0,1} be a Boolean state and [*b*,*e*] an interval, i.e. *b*,*e* å *Time* and e b. Mathematically, the duration of state *P* over [*b*,*e*] equals the integral $\int_b^e P(t)\,dt$, where *P* is a function from time to Booleans (represented by {0,1}). *P* gives the duration of the interval over which *P* holds. Let true denotes the state function which maps any time point to 1. The length of an interval can be defined as $\ell \mathrel{\hat{=}} \int true$.

A Boolean state *P* holds almost everywhere over a non-empty interval can be defined as:

$$\lceil P \rceil \mathrel{\hat{=}} \left(\int P = \ell\right) \wedge \ell > 0 \tag{1}$$

The concept of an event [2], derived two operators *followed-by* and *leads-to* which are useful in specifications. The event is a discontinuity of a state. The reader may refer to [2] for more details. We only present the leads-to notation since we will be using it throughout our paper.

Leads-to : *For a given formula D and state assertion P and positive real number t, the construct* $D \xrightarrow{t} \lceil P \rceil$ *(D leads to P within t) is defined by*

$$D \xrightarrow{t} \lceil P \rceil \mathrel{\hat{=}} \neg\Diamond((D \wedge \ell = t) \frown \lceil P \rceil) \tag{2}$$

This property states that it is not the case that the observation starts with *D* on holding for *t* time unit followed by $\neg P$ for a non-point subinterval.

3.0 OVERALL DESCRIPTION OF INDUSTRIAL PRODUCTION CELL

This section is devoted to the development of an automation software to control a typical industrial production cell used in a metal processing factory. The example was taken from a real metal processing plant in Karlsruhe [4].

The production cell consists of two conveyor belts, a positioning table (or elevation-table), a two-armed robot and a press. Fig. 1 shows a conceptual view of the production cell.

The production cell serves to process metal blanks which are conveyed to a press by a feed-belt. A robot takes each blank from the feed-belt and places it into the press. The robot arm then withdraws from the press, while the press processes the metal blank and opens again. Finally, the robot takes the forged metal plate out of the press and puts it on a deposit-belt.

This procedure is made more complicated by adding further mechanisms. To optimise the utilisation of the press, the robot is fitted with two arms - thus making it possible for the first arm to pick up a blank while the press is forging another plate. The robot arms are placed on different horizontal planes, and they are not vertically mobile. This explains why an elevating rotary table has to be intercalated between the feed-belt and the robot. Another consequence of the fact that the two robot arms are at different levels is that the press has not only two but three states, i.e. open for unloading by the lower arm, open for loading by the upper arm, and closed for pressing.

The general sequences (from the perspectives of a metal plate) are as follows:

1. The feed-belt conveys the metal plate to the elevating rotary table.
2. The elevating rotary table is moved to a position adequate for the unloading by the first robot arm.
3. The first robot arm picks up the metal plate.
4. The robot rotates counter clockwise so that the arm 1 points to the open press, places the metal plate into it and then withdraws from the press.
5. The press forges the metal blank and opens again.
6. The robot retrieves the metal plate with its second arm, rotate further and unloads the plate on the deposit-belt.
7. The deposit-belt transports the plate to the packing operator.

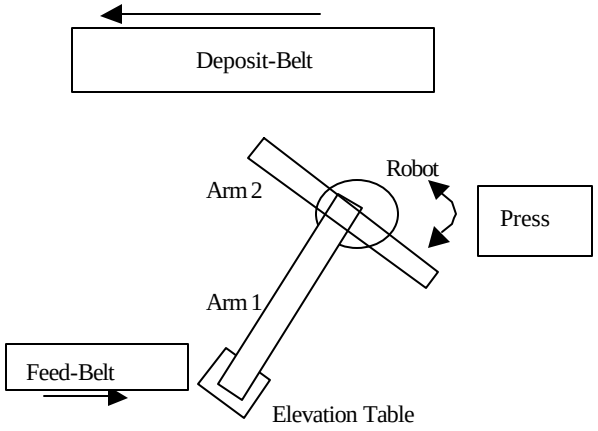


Fig. 1: Conceptual View of the Production Cell

To tackle this problem, we apply the divide-and-conquer method by dealing with each component of the cell individually, i.e. the Feed-Belt (FB), the elevation-table (ET), the two-arms-robot (R), the press (P) and the Deposit-Belt (DP).

Each component is controlled by its own controller. All controllers run concurrently and they are synchronised using shared state sensors. There are numbers of sensors and actuators involved in controlling the cell which we will mention in subsequent sections.

4.0 MODELING THE PLANT AND ENVIRONMENT

This phase focuses on a description of the plant. In this phase, we will produce a state model for the input and output.

4.1 Feed-Belt

The task of the feed-belt is to transport metal blanks to the elevating rotary table. The belt is powered by an electric motor, which can be started up or stopped off by the control program. A photo-electric cell is installed at the end of the belt; which reports whether a blank has entered or left the final part of the belt. We can specify the sensor signal using the notation of duration calculus as follows:

$$MendFB : Time \rightarrow \{0,1\}, \text{ where} \tag{3}$$

$$MendFB(t) = \begin{cases} 0 - \text{No metal at the end of Feed - Belt} \\ 1 - \text{Metal at the end of Feed - Belt} \end{cases} \tag{4}$$

This state variable is set to 1 if the metal is at the end of the belt and to 0 otherwise. Whereas the actuator signal which is used to control the movement of the belt can be modelled as follows:

$$moveFB : Time \rightarrow \{0,1\}, \text{ where} \tag{5}$$

$$moveFB(t) = \begin{cases} 0 - \text{Stop the feed - belt motor} \\ 1 - \text{Start the feed - belt motor} \end{cases} \tag{6}$$

In the following section, the notation $statename: Time \rightarrow \{0,1\}$ representing the Boolean function will be omitted to save spaces.

4.2 Elevation-Table

The task of the elevating rotary table is to rotate the blanks by about 45 degrees and to lift them to a level where they can be picked up by the first robot arm. The vertical movement is necessary because the robot arm is located at a different level than the feed-belt and because it cannot perform vertical translations. The rotation of the table is also required because the arm's gripper is not rotary, and is therefore, unable to place the metal plates into the press in a straight position by itself. Fig. 2 shows the elevating rotary table.

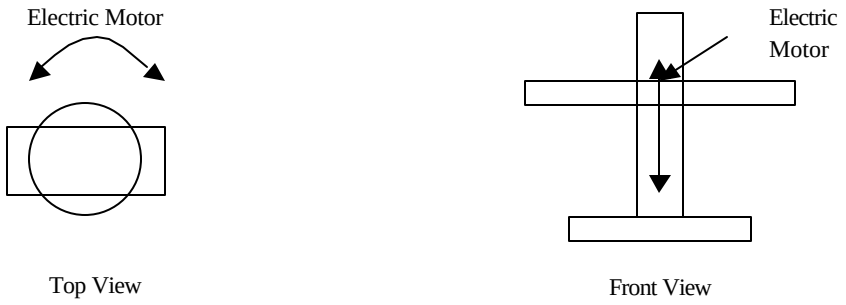


Fig. 2: Elevation Rotary Table

There are two possible positions for the elevating rotary table. One is the rotation position and the second is the vertical position. How far the table rotates can be detected by the potentiometer as a continuous value. We can model this as

$$ETrotate: Time \rightarrow R \tag{7}$$

where R is the rotation in degrees from the feed-belt.

The vertical position can be detected by two switches. One informs whether it is in lower position and the second informs whether it is in upper position. It can be modelled by two Boolean state variables $ETlowPos$ and $ETupPos$ defined as follows

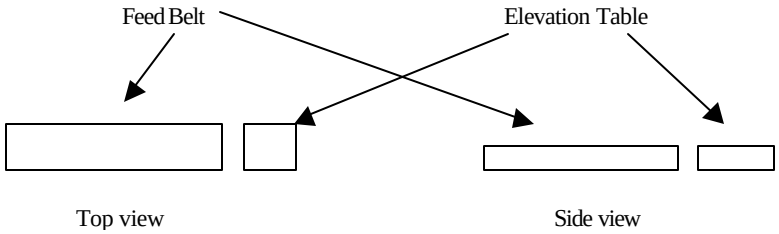
$$ETlowPos(t) = \begin{cases} 0 - \text{Not in the lower position} \\ 1 - \text{In the lower position} \end{cases} \tag{8}$$

$$ETupPos(t) = \begin{cases} 0 - \text{Not in the upper position} \\ 1 - \text{In the upper position} \end{cases} \tag{9}$$

Our only concern at this stage is the two positions. One is when the Elevation-Table is at 0^o with the feed-belt (i.e. in line with the feed-belt) and it is at the low position. The second position is when the Elevation-Table is at 45^o with the feed-belt and it is at the upper position. Therefore, we can have a composite Boolean state as follows:

$$ETPos(t) = \begin{cases} 0 - \text{ETrotate} = 45^o \text{ and ETupPos} \\ 1 - \text{ETrotate} = 0^o \text{ and ETlowPos} \end{cases} \quad (10)$$

Fig. 3 shows these two positions. Fig. 3 (a) shows side view and top view when the *ETPos* value is 1. The table is at the lower position and in-line with the feed-belt. Fig. 3 (b) shows when the elevation-table is at the position from which the robot can unload the metal, i.e. *ETPos* = 0.



a) ETpos = 1

b) ETpos = 0

Fig. 3: Elevation-Table Position

Another sensor indicates whether there is any metal on the elevation-table. This can be described by the state model:

$$ETempty(t) = \begin{cases} 0 - \text{Metal is on the elevation table} \\ 1 - \text{Metal is not on the elevation table} \end{cases} \quad (11)$$

The elevation-table can be moved from position 0 to 1 or vice versa using the actuator signal namely *moveET*. The actuator can be specified as:

$$moveET(t) = \begin{cases} 0 - \text{Move ET to } 45^o \text{ rotation and up position} \\ 1 - \text{Move ET in line with Feed - Belt and lower position} \end{cases} \quad (12)$$

4.3 Robot

The robot comprises two orthogonal arms. For technical reasons, the arms are set at two different levels. Each arm can retract or extend horizontally. Both arms rotate jointly. Mobility on the horizontal plane is necessary, since elevating rotary table, press, and deposit belt are all placed at different distances from the robot's turning centre.

The end of each robot arm is fitted with an electromagnet that allows the arm to pick up metal plates. The robot's tasks consist of taking metal blanks from the elevating rotary table to the press and transporting forged plates from the press to the deposit-belt. See Fig. 4.

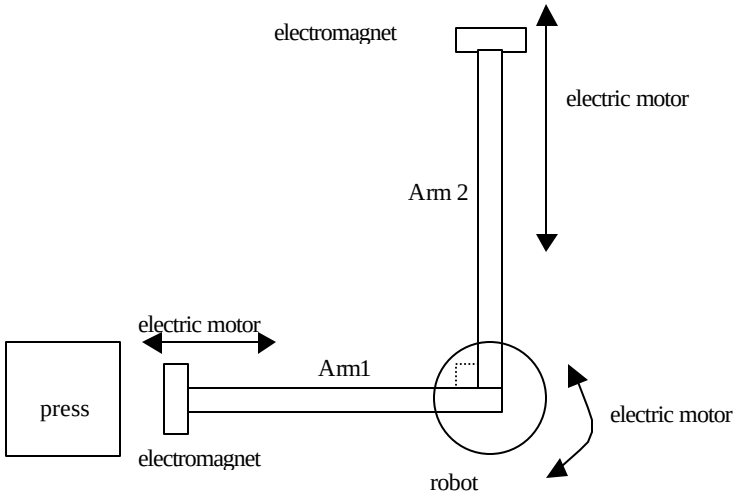


Fig. 4: Robot and Press (top view)

We can specify the electromagnetic actuators for the two arms as follows:

$$ActMagnetA\ 1(t) = \begin{cases} 0 - \text{Deactivate magnet on Arm1} \\ 1 - \text{Activate magnet on Arm 1} \end{cases} \tag{13}$$

$$ActMagnetA\ 2(t) = \begin{cases} 0 - \text{Deactivate magnet on Arm 2} \\ 1 - \text{Activate magnet on Arm 2} \end{cases} \tag{14}$$

The true signal will cause the magnet to activate and false to deactivate the magnet. There are two sensors for each arm to detect whether the metal is loaded or not. This can be specified by:

$$A1load(t) = \begin{cases} 0 - \text{Arm 1 is not loaded} \\ 1 - \text{Arm 1 is loaded with metal} \end{cases} \tag{15}$$

$$A2load(t) = \begin{cases} 0 - \text{Arm 2 is not loaded} \\ 1 - \text{Arm 2 is loaded with metal} \end{cases} \tag{16}$$

The robot is fitted with two arms so that the press can be used for maximum capacity. Below, we describe the order of the rotation operations the robot arm has to perform (Fig. 5); supposing the feed-belt delivers blanks frequently enough. We presuppose that initially the robot is rotated such that Arm 1 points towards the elevating rotary table, and assume that all arms are retracted to allow safe rotation.

1. Arm 1 extends and picks up a metal blank from the elevating rotary table.
2. The robot rotates counterclockwise until Arm 2 points towards the press. Arm 2 is extended until it reaches the press. Arm 2 picks up a forged work piece and retracts.
3. The robot rotates counterclockwise until Arm 2 points towards the deposit-belt. Arm 2 extends and places the forged metal plate on the deposit-belt.
4. The robot rotates counterclockwise until Arm 1 can reach the press. Arm 1 extends, deposits the blank in the press, and retracts again.

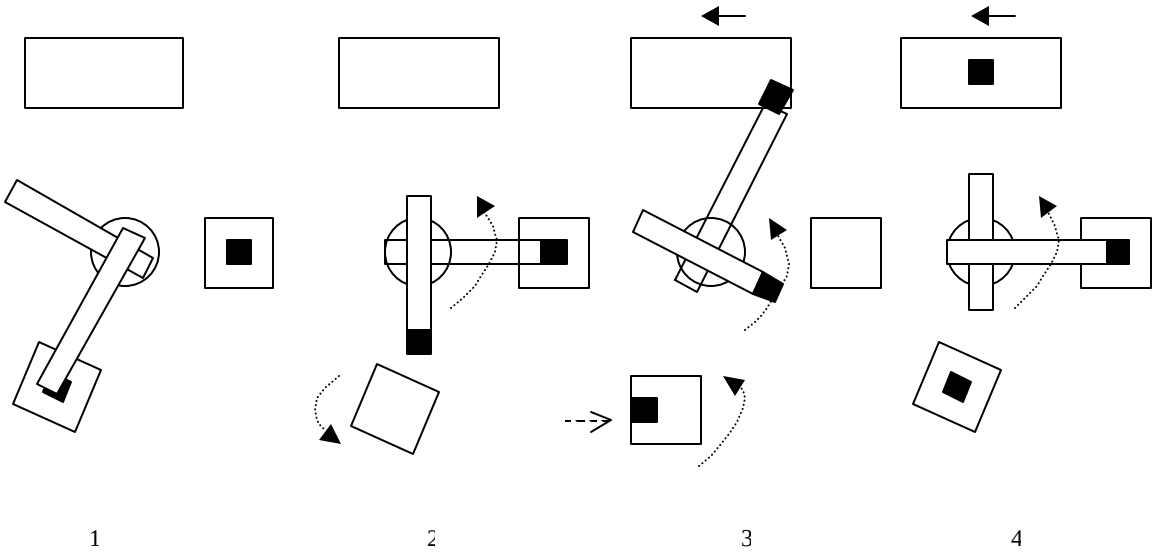


Fig. 5: Order of Robots's actions

Finally, the robot rotates clockwise towards its original position, and the cycle starts again with 1.

There are four robot positions that can be detected by 3 sensors. The three sensors respectively indicate how far the arm 1 extends, how far the Arm 2 extends and the angle of the rotation of the robot. This can be represented in DC as follows:

$$A1Extend: Time \rightarrow R \tag{17}$$

$$A2Extend: Time \rightarrow R \tag{18}$$

$$Rrotate: Time \rightarrow R \tag{19}$$

At this level of abstraction, we can combine this three sensor signals to four positions in concerned as follows:

$$RobotPos: Time \rightarrow Position, \text{ where} \tag{20}$$

$$RobotPos(t) = \begin{cases} 0 - \text{Arm 1 over the Elevation - Table} \\ 1 - \text{Arm 2 in the Press} \\ 2 - \text{Arm 2 over the Deposit - Belt} \\ 3 - \text{Arm 1 in the Press} \end{cases} \tag{21}$$

There are also three actuators that can be used to rotate and extend/retract both arms. However, at this stage of abstraction, we will use one signal actuator called *next* which will activate the rotation motor and the both arms so that it will move to the next position. This actuator can be represented by DC as follows:

$$next(t) = \begin{cases} 0 - \text{No signal send} \\ 1 - \text{Signal send to the robot's motor} \end{cases} \tag{22}$$

By using one composite sensor for robot position and one actuator signal, we can simplify the specification.

4.4 Press

The task of the press is to forge metal blanks. The press consists of two horizontal plates. The lower plate is movable along a vertical axis. The press operates by pressing the lower plate against the upper plate. Because the robot arms are placed on different horizontal planes, the press has three positions. In the lower position, the press is

unloaded by Arm 2, while in the middle position it is loaded by Arm 1. The operation of the press is coordinated with the robot arms as follows (see Fig. 6):

1. Open the press in its lower position and wait until Arm 2 has retrieved the metal plate and left the press.
2. Move the lower plate to the middle position and wait until Arm 1 has loaded and left the press.
3. Close the press, i.e. forge the metal plate, and then repeat step 1 cyclically.

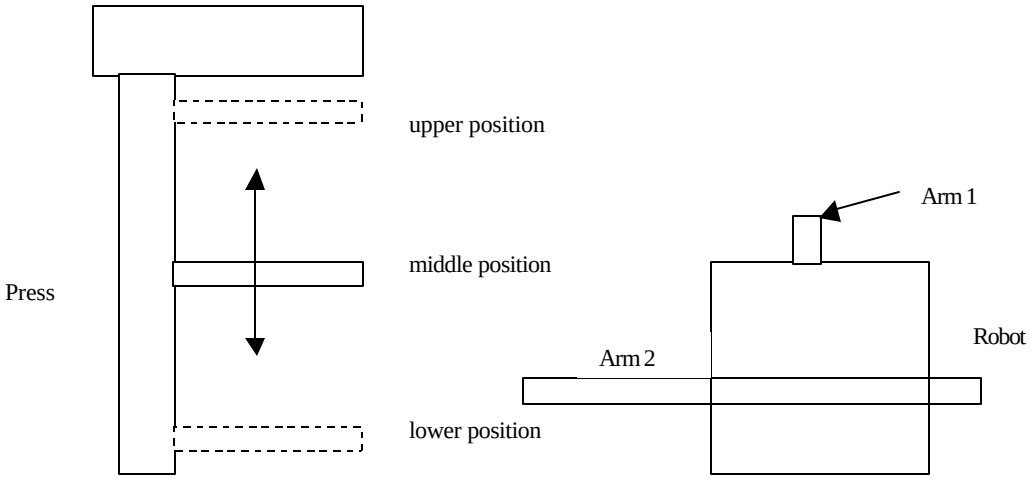


Fig. 6: Robot and Press (side view)

We have three sensors to detect the position of the moving level whether it is at the lower, middle or upper position as follows:

$$PlowPos(t) = \begin{cases} 0 - \text{Not in lower position} \\ 1 - \text{In the lower position} \end{cases} \tag{23}$$

$$PmidPos(t) = \begin{cases} 0 - \text{Not in middle position} \\ 1 - \text{In the middle position} \end{cases} \tag{24}$$

$$PupPos(t) = \begin{cases} 0 - \text{Not in upper position} \\ 1 - \text{In the upper position} \end{cases} \tag{25}$$

Another sensor is to detect whether the presser is empty or not. The state to model this sensor can be as follows:

$$Pempty(t) = \begin{cases} 0 - \text{Metal is on the movable level of the Press} \\ 1 - \text{Metal is not on the movable level of the Press} \end{cases} \tag{26}$$

We will use one actuator signal that will make the motor move the level to the next position. This signal can be specified as

$$moveP(t) = \begin{cases} 0 - \text{No signal send} \\ 1 - \text{Signal send to the robot's motor} \end{cases} \tag{27}$$

4.5 Deposit-Belt

The task of the deposit-belt is to transport the work pieces unloaded by the second robot arm to the packing operator. A photo-electric cell is installed at the end of the belt. It reports whether or not a work piece reaches the

end section of the belt. The control program then has to stop the belt. The belt can restart as soon as the operator picks up the work piece.

This sensor can be represented by DC as follows:

$$MendDB(t) = \begin{cases} 0 \text{ - No Metal at the end of Deposit - Belt} \\ 1 \text{ - Metal at the end of Deposit - Belt} \end{cases} \tag{28}$$

To avoid the Arm 2 unloading a metal on another metal that is previously loaded onto the belt, another sensor call *DBempty* is installed at the position where Arm 2 unloads. This can be represented by:

$$DBempty(t) = \begin{cases} 0 \text{ - Metal is on the dropping position of Deposit - Belt} \\ 1 \text{ - Metal is not on the dropping position of Deposit - Belt} \end{cases} \tag{29}$$

Only one actuator signal involve to move and stopping the Deposit belt. This can be specified in DC as follows:

$$moveDB(t) = \begin{cases} 0 \text{ - Stop the Deposit - Belt motor} \\ 1 \text{ - Start the Deposit - Belt motor} \end{cases} \tag{30}$$

5.0 ASSUMPTIONS

In this phase we list the assumptions which are the aspects of the behaviour of the plant that cannot be controlled by the controller. This behaviour must be provided by the vendor of the plant.

1. If the Feed-Belt is moving, eventually the metal will reach at the end of the table. This will set on the sensor $MendFB$.

$$\left(\lceil moveFB \rceil \wedge \lceil \neg MendFB \rceil \right) \xrightarrow{e_1} \lceil MendFB \rceil \tag{31}$$

2. Once the metal crosses the the $MendFB$ sensor while the Elevation Table is in-line with the Feed-Belt, this will switch off the sensor $ETempty$ and also $MendFB$.

$$\left(\lceil ETPos \rceil \wedge \lceil MendFB \rceil \wedge \lceil moveFB \rceil \right) \xrightarrow{e_2} \lceil \neg ETempty \wedge \neg MendFB \rceil \tag{32}$$

3. A true signal sent to actuator $moveET$ will lead to the movement of the Elevation-Table to the position in-line with the feed-belt and set the sensor $ETPos$ to 1. On the other hand, if the signal 0 is sent, then the elevation-table will move to 45° and the upper position.

$$\lceil moveET \rceil \xrightarrow{e_3} \lceil ETPos \rceil \tag{33}$$

$$\lceil \neg moveET \rceil \xrightarrow{e_4} \lceil \neg ETPos \rceil \tag{34}$$

4. If the magnet of the Arm 1 is activated while it is over the elevation-table, it will set the sensor $A1load$ to true. And at the same time the changes of $A1load$ from false to true, will lead the setting of $ETempty$ to true.

$$\left(\begin{array}{c} \lceil ActMagnetA1 \rceil \wedge \lceil RobotPos = 0 \rceil \\ \wedge \\ \lceil \neg A1load \rceil \wedge \lceil \neg ETempty \rceil \end{array} \right) \xrightarrow{e_5} \lceil A1load \wedge ETempty \rceil \tag{35}$$

5. On the receipt of the *next* signal, the robot will extract the arm and turn, and extend the arm to the next position of the four possible positions 0 to 3.

$$(\lceil RobotPos = i \rceil \wedge \lceil next \rceil) \xrightarrow{e_6} \lceil RobotPos = i \oplus 1 \rceil \tag{36}$$

where

$$i \oplus 1 \hat{=} i + 1 \bmod 4 \tag{37}$$

6. If the magnet of the Arm 2 is activated while it is in the press, it will set the sensor *A2load* to true. And at the same time, the changes of *A2load* from false to true, will change *Pempty* to true.

$$\left(\begin{array}{c} \lceil ActMagnetA\ 2 \rceil \wedge \lceil RobotPos = 1 \rceil \\ \wedge \\ \lceil \neg A2load \rceil \wedge \lceil \neg Pempty \rceil \end{array} \right) \xrightarrow{e_7} \lceil A2load \wedge Pempty \rceil \tag{38}$$

7. If the magnet of the Arm 2 is deactivated while it is over the deposit-belt, it will set the sensor *A2load* to false. The change of *A2load* from true to false will falsify *DBempty*.

$$\left(\begin{array}{c} \lceil \neg ActMagnetA\ 2 \rceil \wedge \lceil RobotPos = 2 \rceil \\ \wedge \\ \lceil A2load \rceil \wedge \lceil DBempty \rceil \end{array} \right) \xrightarrow{e_8} \lceil \neg A2load \wedge \neg DBempty \rceil \tag{39}$$

8. If the magnet of the Arm 1 is deactivated while it is in the press, it will set the sensor *A1load* to false. The change of *A1load* from true to false will falsify *Pempty*.

$$\left(\begin{array}{c} \lceil \neg ActMagnetA\ 1 \rceil \wedge \lceil RobotPos = 3 \rceil \\ \wedge \\ \lceil A1load \rceil \wedge \lceil Pempty \rceil \end{array} \right) \xrightarrow{e_9} \lceil \neg A1load \wedge \neg Pempty \rceil \tag{40}$$

9. A signal *MoveP* will trigger the move level of the press from one position to another in cyclic order. It moves from lower position to middle position, and then to the upper position and back to lower position.

$$(\lceil PlowPos \rceil \wedge \lceil moveP \rceil) \xrightarrow{e_{10}} \lceil PmidPos \rceil \tag{41}$$

$$(\lceil PmidPos \rceil \wedge \lceil moveP \rceil) \xrightarrow{e_{11}} \lceil PupPos \rceil \tag{42}$$

$$(\lceil PupPos \rceil \wedge \lceil moveP \rceil) \xrightarrow{e_{12}} \lceil PlowPos \rceil \tag{43}$$

10. If the deposit-belt is moving when the sensor *MendDB* is off, eventually the metal will reach the end of the belt and cause the *MendDB* to be set to on. At the same time, the sensor *DBempty* is set to on.

$$(\lceil moveDB \rceil \wedge \lceil \neg MendDB \rceil) \xrightarrow{e_{13}} \lceil DBempty \wedge MendDB \rceil \tag{44}$$

6.0 THE SPECIFICATIONS OF THE CONTROLLERS

6.1 The Specification of the Feed-Belt Controller

1. The feed-belt will be moved either when there is no metal at the end of the belt or the empty elevation-table is in-line with the feed-belt.

$$(\lceil \neg MendFB \rceil \vee (\lceil ETPos \rceil \wedge \lceil ETempty \rceil)) \xrightarrow{d_1} \lceil moveFB \rceil \tag{45}$$

2. The feed-belt will be stopped when the elevation-table is filled and also the other metal is at the end of the belt. For safety, the belt must also be stopped when the metal arrives at the end but the elevation-table is not in-line with the feed-belt.

$$\left(\begin{array}{c} (\lceil MendFB \rceil \wedge \lceil \neg ETempty \rceil) \\ \vee \\ (\lceil MendFB \rceil \wedge \lceil \neg ETPos \rceil) \end{array} \right) \xrightarrow{d_2} \lceil \neg moveFB \rceil \tag{46}$$

6.2 The Specification of the Elevation-Table Controller

1. The filled elevation-table which is currently in-line with the feed-belt will be moved to the unloading position.

$$(\lceil ETPos \rceil \wedge \lceil \neg ETempty \rceil) \xrightarrow{d_3} \lceil \neg moveET \rceil \tag{47}$$

2. On the other hand, when the empty elevation-table is at the unloading position, it will be moved back to the position in-line with the feed-belt.

$$(\lceil \neg ETPos \rceil \wedge \lceil ETempty \rceil) \xrightarrow{d_4} \lceil moveET \rceil \tag{48}$$

6.3 The Specification of the Robot Controller

One of the most important requirements in the robot controller is, it must proceed in the order shown in Fig. 5. The robot starts by loading Arm 1, then follows by loading Arm 2, unloading Arm 2 and unloading Arm 1. The cycle is repeated again from the very beginning. We will specify each operation and take into account the proper ordering. Please take note that the robot position is numbered from 0 to 3.

Loading Arm 1 consists of two step. The first is to bring the Arm 1 to the first position (i.e. $RobotPos = 0$) and the second is to activate the magnet to extract the metal from the elevation-table. However, the arm must be at the last position and also the operation at that position must be completed. The last operation is unloading the metal from Arm 1 into the press. Therefore, we can use the state of $A1load$ sensor to make sure that the previous operation is completed.

The first DC formula states that, if the $RobotPos$ is at the last position (i.e. 3) and the sensor $A1load$ is 0, then we can send a signal to move the robot to the next position.

$$(\lceil RobotPos = 3 \rceil \wedge \lceil \neg A1load \rceil) \xrightarrow{g_1} \lceil next \rceil \tag{49}$$

Once the Robot is at the position of loading Arm 1, we must check whether the elevation-table is ready at the unloading position and also if there is a metal to be loaded. The DC formula below specifies the condition.

$$(\lceil RobotPos = 0 \rceil \wedge \lceil \neg ETPos \rceil \wedge \lceil \neg ETempty \rceil) \xrightarrow{g_2} \lceil ActMagnetA1 \rceil \tag{50}$$

Loading Arm 2 consists of two steps. One is to move the robot and arm to the position near to the press and then to activate the magnet. To move the second arm into the press safely, we have to ensure that the robot must be at the first position; the operation of loading the first arm is already finished; the second arm is not yet loaded; the press is at the low position so that the Arm 2 will not collide with the press; and the press is not empty.

The specification for this can be written using DC as follows:

$$\left(\begin{array}{c} \lceil RobotPos = 0 \rceil \wedge \lceil A1load \rceil \wedge \lceil \neg A2load \rceil \\ \wedge \\ \lceil PlowPos \rceil \wedge \lceil \neg Pempty \rceil \end{array} \right) \xrightarrow{g_3} \lceil next \rceil \tag{51}$$

Once the robot is at the second position, we can activate the magnet provided that the second arm is not yet loaded and the press is not empty. The specification can be given as follows:

$$\left(\lceil RobotPos = 1 \rceil \wedge \lceil \neg A2load \rceil \wedge \lceil \neg Pempty \rceil \right) \xrightarrow{g_4} \lceil ActMagnetA\ \ 2 \rceil \tag{52}$$

Unloading Arm 2 To unload the second arm, we bring the robot and arm from the second position to the third provided the operation at the second position is completed. This can be done by checking the $A2load$ sensor. Once the robot is at the third position, we can deactivate the magnet provided that there is no metal on the deposit-belt over the arm.

The first formula specifies the movement and the second specifies the deactivation of the magnet.

$$\left(\lceil RobotPos = 1 \rceil \wedge \lceil A2load \rceil \right) \xrightarrow{g_5} \lceil next \rceil \tag{53}$$

$$\left(\lceil RobotPos = 2 \rceil \wedge \lceil DBempty \rceil \right) \xrightarrow{g_6} \lceil \neg ActMagnetA\ \ 2 \rceil \tag{54}$$

Unloading Arm 1. Unloading the metal from the first arm into the press also needs two steps as the other operations. To move the arm into the press, we have to establish the following conditions: the robot is currently at third position; the second arm has been unloaded; the first arm is empty; the press is at the middle in order to avoid the collision with the press; the press is empty.

The specification for the process of moving the Arm 1 is shown below:

$$\left(\begin{array}{c} \lceil RobotPos = 2 \rceil \wedge \lceil A1load \rceil \wedge \lceil \neg A2load \rceil \\ \wedge \\ \lceil PmidPos \rceil \wedge \lceil Pempty \rceil \end{array} \right) \xrightarrow{g_7} \lceil next \rceil \tag{55}$$

Once the arm is in the press, and the first arm is loaded and the press is empty, we can deactivate the first arm. The specification can be written as:

$$\left(\lceil RobotPos = 3 \rceil \wedge \lceil A1load \rceil \wedge \lceil Pempty \rceil \right) \xrightarrow{g_8} \lceil \neg ActMagnetA\ \ 1 \rceil \tag{56}$$

6.4 The Specification of the Press Controller

The Press's controller needs to control the moving's level of the press. There are three positions: low, middle and upper position. The order of the movement is low, and then middle and followed by the upper position. From the upper position, the press should be moved to the lower position and the cycle starts again. At any time during the movement, no arm should be in the press. This can be done by making sure that the robot is not at the second and fourth position. The second position is when the second arm is in the press and the fourth is when the first arm is in the press. We can specify this by:

$$ArmNotIn\ \Pr ess \hat{=} \neg\left(RobotPos = 1 \wedge RobotPos = 3\right) \tag{57}$$

With this we can specify the movement as follows. Please remember that the *moveP* actuator will cause the level to move to the next position as stated earlier as part of the assumptions.

1. At any time t when the level is at the lower position and it is empty and also there is no arm in the press, the level can be moved by sending the *moveP*.

$$\left(\lceil PlowPos \rceil \wedge \lceil Pempty \rceil \wedge \lceil ArmNotIn\ \Pr ess \rceil\right) \xrightarrow{u_1} \lceil moveP \rceil \tag{58}$$

   This condition is satisfied when the second arm of the robot has unloaded the metal from the press and has left the press.

2. Whenever the level at the middle position and there is a metal as well as no arms in the press, the controller needs to move the level to the upper position.

$$\left(\lceil PmidPos \rceil \wedge \lceil \neg Pempty \rceil \wedge \lceil ArmNotIn\ \Pr ess \rceil\right) \xrightarrow{u_2} \lceil moveP \rceil \tag{59}$$

3. Once the press has forged the metal, it will return to lower position for Arm 2 to unload. However, the level can only be moved if there is no arm in the press.

$$\left(\lceil PupPos \rceil \wedge \lceil \neg Pempty \rceil \wedge \lceil ArmNotIn\ \Pr ess \rceil\right) \xrightarrow{u_3} \lceil moveP \rceil \tag{60}$$

6.5 Specification of the Deposit-Belt Controller

The controller is responsible for moving the belt when there is no metal at the end of the belt or stop otherwise. The specification can be given as follows:

$$\lceil \neg MendDB \rceil \xrightarrow{u_4} \lceil moveDB \rceil \tag{61}$$

$$\lceil MendDB \rceil \xrightarrow{u_5} \lceil \neg moveDB \rceil \tag{62}$$

7.0 CONCLUSION

We have shown that the notation of state in Duration Calculus is very useful in describing and modelling the real-time or reactive system. We used state notation to model the input or the sensor and also the output or the actuator of the system. We have also shown that the leads-to notation is very handy in specifying the system. In the stage of producing the assumptions, we use the state of the sensors as conditions and actuators as the actions which lead to new sensors state.

Our main contribution is how we use the sensors state as shared variables to synchronise the controllers. We use the combination of state of the sensors as a condition that must be fulfilled before any signal is sent to any actuator. With this specification, one can implement a system on a shared-memory multiprocessor.

The research can be extended further such as implementing the system on Distributed Shared Memory (DSM), which is the current and future trends of Distributed Real-Time computing. Other area of research is the formal proving technique for verification of a specification.

REFERENCES

[1] M. R. Hansen, Z. Chouchen, "Duration Calculus: Logical Foundation". *Formal Aspects of Computing*. 3:1-1000, 1997.

[2] A. P. Ravn, "Design of Embedded Real-Time Computing System". *Thesis Denmark Technical University*, Denmark. 1995.

[3] The Raise Language Group, *The Raise Specification Language*. Prentice Hall. 1992.

[4] C. Leverentz, T. Linder, "Case Study Production Cell". *A Comparative Study in Formal Software Development*, FZI-Publication. 1/94.

[5] S. Austin, F. I. Parkin, "Formal Methods: A Survey". *Technical Report*, National Physical Laboratory, Great Britain. 1993.

[6] M. Broy, S. Jähnichen, editors: Korrekte Software Durch Formale Methoden, Technische Universität Berlin, Franklinstarabe 28-29, D-10587 Berlin. March 1993.

[7] Z. Chouchen, A. P. Ravn, M. R. Hansen, "An Extended Duration Calculus for Hybrid Real-Time Systems", *UNU/iist Report No. 9*. 1993.

BIOGRAPHY

Rusdi Md. Aminuddin obtained his Master of Computer Science from Western Michigan University in 1989. He is a lecturer at the School of Information Technology, University Utara Malaysia. Currently, he is on study leave pursuing his PhD at Universiti Sains Malaysia. His research areas include distributed and parallel computing and Fault-Tolerance on cluster environment. His teaching areas include Data Communication and Networking, and Concept of Programming Languages.

He Jifeng is a Senior Research Fellow of UNU/IIST, on leave from East China Normal University and Shanghai Jiao Tong University of China, where he is a professor. His previous position is as Senior Research Fellow at Oxford University Computing Laboratory, Programming Research Group. His research interest lies in the sound methods of specification of computer systems, communications, application and standards, and the techniques for designing and implementing those specifications in software and /or hardware, with high reliability and at low cost.

Rosni Abdullah obtained her PhD in Computer Science from Loughborough University, UK in 1997. Currently, she is an Associate Professor and Deputy Dean for Postgraduate Studies and Research at the School of Computer Science, Universiti Sains Malaysia. She also leads the Parallel and Distributed Research Group at the department. Her research interest includes parallel and distributed computing Design and Analysis of parallel algorithms and Distributed Shared Memory (DSM).