I. Simulation of an Industrial Production cell

Artur Brauer

Abstract

In this work, an application of Tk/Tcl to the domain of process visualization is described. We developed a simulation of an industrial production cell for to evaluate and validate control software for this (reactive) system. This paper describes the abilities of the simulation and how it is used.

1.1 Problem

In the context of the German “Korso” Project (Correct Software) a number of formal software construction methodologies are developed. The Forschungszentrum Informatik (FZI) proposed the case study “production cell” to evaluate and compare these approaches. It bases on a realistic functional model of a small industrial production cell, which is completely described in [1].

The task of the case study, which is treated by several research groups in Germany, is the development of control software with various approaches, in order to compare the approaches and to show the usefulness of formal methods enforcing safety requirements in industrial applications. As the partners are distributed all over Germany it was necessary to provide them with a facility to evaluate their developments. We decided to construct a graphical software simulation of the production cell that should be easy to build and easy to connect with any control software.

1.2 Simulation

We developed a simulation of the production cell, which imitates the important abilities of the real production cell. The simulation runs a graphical window which displays the substantial elements of the production cell. For a better performance (which suffers anyway) all the devices are drawn simplified, see figure 1.

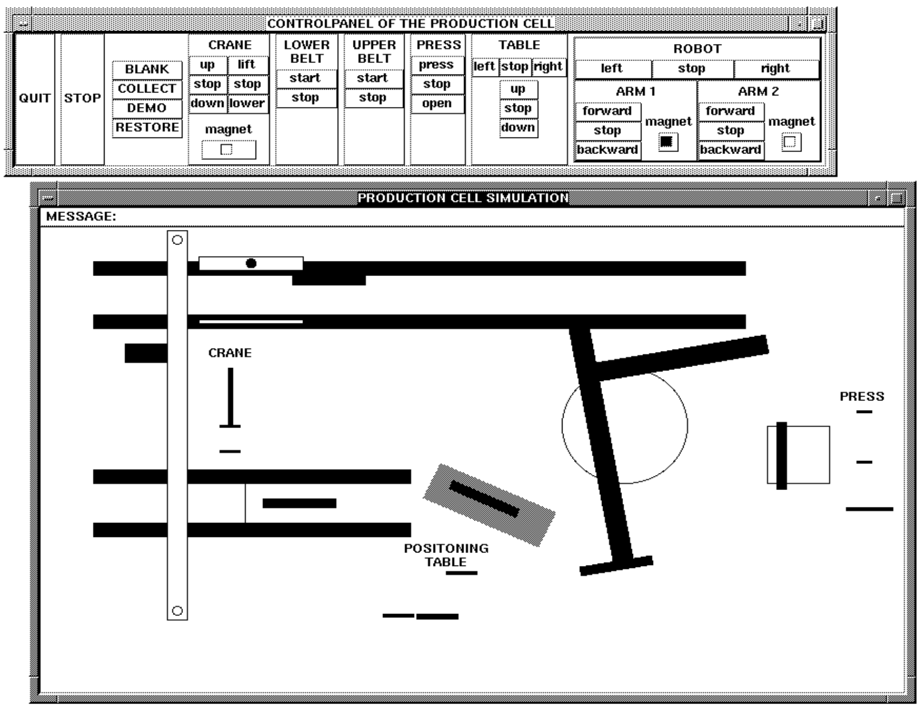

Figure 1 *Screen dump of the visualization (together with a control panel)*

The simulation is managed by transmitting commands to it and receiving sensor information from it (the commands are described in subsection 1.5). Both kind of communication is done via UNIX-pipes, so a control software just have to write commands to stdout and read the status information from stdin. It performs the movement of the devices and blanks, detects collisions and reports them. The simulation displays the production cell from the top view. The elevating rotary table, the press and the crane can change their height too. This kind of movement wouldn't be visible in the chosen view. For that reason we have introduced scales, which represents the height of a device. A scale shows the name of the corresponding device and the height indication. Additionally, it contains height marks, which represents the height of the neighbour devices. For a simple identification the marks use the same colour like the devices they stand for.

The simulation machines blanks which are stored in the left bottom corner of the window. To put a blank onto belt 1, one can use the related command. During the testing it might be useful to put a blank directly on a particular device. This can be done by selecting the blank pressing the left mouse button and dragging it over one of the following devices: both belts, elevating rotary table or press.

The simulation is distributed together with a control panel. It can be used to get familiar with the simulation by manual control. It also provides a *demo* button which causes the simulation to perform a demonstration.

The simulation can be used in two modes. There is once the *asynchronous* mode, which means that the simulation will use its own stroke. This is useful when it is used with the manual control panel. Once a command is given to the simulation it will be performed until it is stopped intern or extern. Imagine, you give the command for rotating the robot left. The simulation will rotate the robot step by step until you stop it or until it reaches its left stop (or causes a collision). The *synchronous* mode means, that you have to give the stroke for the simulation. After you have given the command for rotating the robot left in the above example, you have to give a *react* command each time you want the robot (or all the other devices) to move one more step. In this way the simulation can be forced to run synchronous to your controller.

1.3 Install the simulation

The implementation of the simulation was done using Tk/Tcl. For that reason you need a running Tk/Tcl installation on you system to start the simulation. The simulation is available as the compressed tarfile *visualization.tar.Z* on the ftp server *gate.fzi.de* [141.21.4.3] in the directory *pub/korso/fzelle/simulation.* To use the simulation you first have to uncompress and then to un-tar the file into a directory of your choice. The README file contains a list of files which all should be now in your simulation directory. Change the file *startsimu* by entering the correct path of your *wish* interpreter in the first line (note: the path must not be longer than 30 characters!) and set the variable *wishpath* in the third line also to the path for *wish*. Now you should have an executable version of the simulation which can be stated with the *startsimu* file. The next subsection describes in detail how the simulation is started.

1.4 Starting the simulation

For starting the simulation the executable file startsimu is provided together with the simulation. It takes several parameters and starts the simulation coupled together with your control program or with the control panel. If you call startsimu without any parameters, it will come up together with the manual control panel, both will use colours display mode and english language. The simulation will use the asynchronous mode. In below the parameters are listed which can be used to configure the simulation.

- **-con controllername** the simulation will be started together with the controller **controllername**; in case you want to give some parameters to the controller you have to quote them together with **controllername**
- **-snc** the simulation will use the synchronous mode; the controller have give a react command each time it want the simulation to do one more cycle
- **-bw** the simulation will use monochrome display
- **-grm** the simulation will use german language
- **-eng** the simulation will use english language

1.5 Commands protocol

To couple the control program and simulation UNIX-pipes are used. This allows the control program to steer the simulation just by writing ASCII commands to *stdout* and reading status information of the simulation from *stdin*.

The simulation provides two kinds of commands. There are once commands for goggling the devices and manipulating the whole system, which don't provide feedback. On the other hand there are some commands to get information from the simulation, which then can be read from stdin. It is of great importance that after both kinds of commands the control program flushes the pipe, since otherwise the commands will be buffered and will have no effect. This can be done e.g. in C with the fflush command. All possible commands are now described below.

1.5.1 System commands

This commands provide the facility to manipulate the simulation. They don't belong to particular device and don't produce any output from the simulation.

- **system_quit** quits the simulation
- **system_stop** stops any movement in the simulation
- **blank_add** puts a blank from the stock on the feed belt
- **blanks_collect** puts all the blanks dropped irregularly back to the stock
- **system_demo** starts a demonstration of the simulation
- **system_restore** restores the simulation
- **react** this command is only available in the synchronous mode; it is then used to force the simulation to do one more cycle; before this command is not transmitted the simulation won't perform a change

1.5.2 Feed belt commands

This commands allow to toggle the feed belt. They don't produce any output from the simulation.

- **belt1_start** starts the feed belt to move right
- **belt1_stop** stops the feed belt

1.5.3 Deposit belt commands

This commands allow to toggle the deposit belt. They don't produce any output from the simulation.

- **belt2_start** starts the deposit belt to move left
- **belt2_stop** stops the deposit belt

1.5.4 Elevating rotary table commands

This commands can be used to rotate and move up and down the elevating rotary table. They don't produce any output from the simulation.

- **table_left** starts the rotation to the left of the elevating rotary table
- **table_stop_h** stops the rotation of the elevating rotary table
- **table_right** starts the rotation to the right of the elevating rotary table
- **table_upward** starts the upward movement of the elevating rotary table

- **table_stop_v** stops the upward or downward movement of the elevating rotary table
- **table_downward** starts the downward movement of the elevating rotary table

1.5.5 Robot commands

This commands allows to rotate the robot and move the arms forward and backward. They don't produce any output from the simulation.

- **arm1_forward** starts the forward movement of arm 1
- **arm1_stop** stops the movement of arm 1
- **arm1_backward** starts the backward movement of arm 1
- **arm1_mag_on** activates the magnet of arm 1
- **arm1_mag_off** deactivates the magnet of arm 1
- **arm2_forward** starts the forward movement of arm 2
- **arm2_stop** stops the movement of arm 2
- **arm2_backward** starts the backward movement of arm 2
- **arm2_mag_on** activates the magnet of arm 2
- **arm2_mag_off** deactivates the magnet of arm 2
- **robot_left** starts the rotation to the left of the robot
- **robot_stop** stops the rotation of the robot
- **robot_right** starts the rotation to the right of the robot

1.5.6 Crane commands

This commands start and stop the movement of the crane. They don't produce any output from the simulation.

- **crane_to_belt2** starts the crane moving toward the deposit belt
- **crane_stop_h** stops the movement of the crane
- **crane_to_belt1** starts the crane moving toward the feed belt
- **crane_lift** starts the crane's magnet moving upward
- **crane_stop_v** stops the movement of the crane's magnet
- **crane_lower** starts the crane's magnet moving downward
- **crane_mag_on** activates the crane's magnet

- **crane_mag_off** deactivates the crane's magnet

1.5.7 Press commands

This commands are used to steer the press. They don't produce any output from the simulation.

- **press_upward** starts the press moving upward
- **press_stop** stops the movement of the press
- **press_down** ward starts the press moving downward

1.5.8 Commands to get status information from the simulation

The simulation provides two commands which can be used to obtain information from the production cell. Both produce a output from the simulation which can be read from stdin. The important one is sure the get_status command. It provides information of device sensors. The get_passings command was introduced to achieve some synchronization possibilities. It shouldn't be necessary when the synchronous mode is used.

- **get_status**

  writes the status information of the simulation to the standard output

  This command causes the simulation to write all the status information to stdout, as a 15 element vector. Each element of the vector is separated by a newline. The status vector contains all the information that the real production cell can provide and additionally a list of errors that occurred since the last status report. The format of the status vector is described below.

Index	Sensor corresponding to device	Description	Value/ Meaning	Notes
1	press	Press in bottom position	1 = yes 0 = no	
2	press	Press in middle position	1 = yes 0 = no	
3	press	Press in top position	1 = yes 0 = no	
4	robot	Extension of arm 1	0..1	Value 1 mens the arm is fully extended.
5	robot	Extension of arm 2	0..1	Value 1 mens the arm is fully ex tended.

Index	Sensor corresponding to device	Description	Value/ Meaning	Notes
6	robot	angle of rotation of the robot	-100..70	Value of 0 degrees means that the robot is in the starting position.
7	elevating rotary table	elevating rotary table in bottom position	1 = yes 0 = no	
8	elevating rotary table	elevating rotary table in top position	1 = yes 0 = no	
9	elevating rotary table	angle of rotation of the table	-5..90	Value of 0 degrees means that the table is in the starting position.
10	crane	is the crane over the deposit belt	1 = yes 0 = no	
11	crane	is the crane over the feed belt	1 = yes 0 = no	
12	crane	height of the crane's magnet	0..1	Value 0 means the magnet is in top position.
13	feed belt	is a blank inside the photoelectric barrier	1 = yes 0 = no	
14	deposit belt	is a blank inside the photoelectric barrier	1 = yes 0 = no	
15	all	error report		A list of errors, that occurred since the last status report. The list is enclosed in curly braces and each element is separated by a space. The errors are coded as shown in the next table.

The meaning of the error numbers

Number	**Meaning**
0	No error occurred; if this is true, no other elements are in the error list.
1	A blank dropped irregularly, during the passage from the feed belt to the elevating rotary table.
2	Collision between elevating rotary table and the feed belt.
3	elevating rotary table reached the right stop.
4	Robot reached the left stop.
5	Robot reached the right stop.
6	Arm 1 dropped a blank invalidly.
7	Collision between arm 1 and press.
8	Arm 2 dropped a blank invalidly.
9	Collision between arm 2 and press.
10	A blank dropped from the end of the deposit belt.
11	Collision between crane and the deposit belt.
12	Collision between crane and the feed belt.
13	Crane dropped a blank irregularly.
14	Crane reached the stop over the feed belt.
15	Crane reached the stop over the deposit belt.

- **get_passings**

  writes the number of passings through the main loop to *stdout*

  This command causes the simulation to write the number of passings through the main loop to stdout. The number of passings is computed modulo 10000. This can be used to synchronize the control program with the simulation, since the performance of the simulation depends a lot on the host it is running on.

1.5.9 Advanced commands

This command should only be used if the controller's speed is to low to assure that a device is stopped at the right time. Even then, the command should be unnecessary if the simulation is used in the synchronous mode. Nevertheless, the command may be useful in some situations.

- **new_guard sensor_number operator destination_value command**

  a new guard is created

  The simulation allows to create guards, which assures that a particular action is performed at the right time. A guard tests a condition until it becomes true and than call a simulation command and removes himself. The condition to be tested consists of a sensor, an operator and a destination value. The **sensor_number** describes which sensor should be compared with the **operator** to the **destination_value**. When this expression is true, **command** will be executed.

1.6 Operating the cell

To machine a blank through the production circuit of the simulation one have to know some detail information about it. There are various collisions that can occur during operating the cell and which of course should be avoided. On the other hand one have to know constants describing e.g. at which angle of the robot and of the table how far have arm 1 to extend to reach the blank. Both knowledge is provided below.

1.6.1 Collisions

There are three kinds of irregular behaviour detected by the simulation. Firstly, two devices can clash against each other. Second a device should be stopped before it reaches the end of it's possible movement, if the user misses to do so this is treated as a failure. And third, a blank that doesn't settle correct from one device to another causes a error report, too.

The following list enumerates the collisions between devices and gives conditions which assures that the production cell works to rule. Note, that the conditions below aren't the only one which assure collision avoidance, but when considered no collisions will occur.

1. The rotary elevating table can clash against the feed belt. This can be avoided, if the angle of the table keeps greater or equal to 0.
2. The rotary elevating table and the robot run the risk of collision, if both arm 1 and the table hold a blank and the table is in top position. Then the two blanks can touch each other, what is detected as a collision. To prevent this, one can keep the robot from the table by one of the following conditions:

   — the angle of the robot is less or equal to 0,

   — the extension of the robot is 0,

   — the table is in bottom position.
3. The robot can reach the press with both arms and cause a collision when the press's height is in the range oh the corresponding arm. Collisions between the arm 1 may be avoided by one of the following conditions:

   — the angle of the robot is greater or equal to -70,

   — the extension of arm1 is less or equal to 0.3708,

   — the press is in bottom or middle position.

   To prevent arm 2 from clashing with the press one the following conditions is satisfying:

   — the robot's angle is greater or equal to 55,

   — the robot's angle is less or equal to 15,

   — the extension of arm 2 is 0,

   — the press is in top or in bottom position.
4. The two last possible collisions between devices are the collisions of the crane with the both belts. If the crane moves over one of the belts and it's magnet is deeper as the belt a collision is detected. Following method is sufficient to prevent failures:

— during the crane’s movement it’s height should be less or equal to 0.6593 (remember that the height of the crane is the distance from the top to the current position).

Besides collisions between devices a controller can cause conflicts by moving some devices out of their moving range. So one have to regard some constraints when moving the devices.

1. The robot’s angle should stay between 65 and -95.
2. The elevating rotary table haven’t to rotate over 85 degrees.
3. The crane shouldn’t move outside the region which is limited by the two photoelectric barriers.

The third kind of possible conflict occurs when a device consign a blank irregularly to the next device. That is e.g. when arm 1 of the robot releases a blank not over the press or the table. There are various situations where a blank can be released faulty. We won’t describe all them here, since for a error-free operating of the simulated cell one just have to know some constraints. If the controller pay attention to them when a blank changes from one device to the next. This information is provided in the section Constants below.

1.6.2 Constants

During operating the cell blanks moves from one device to the next. It is obvious that for such a transition the concerned devices should satisfy some conditions. They are mow described below.

1. When a blank changes from the feed belt to elevating rotary table, the table have to be in bottom position and it’s angle have to be 0.
2. When arm 1 wants to pick up a blank from the elevating rotary table, the following conditions should be satisfied:
   — robot’s angle: 50
   — arm 1 extension: 0.5208
   — table is in top position
   — table’s angle: 50.
3. To put the blank from arm 1 into the press, one have to ensure the following conditions:
   — robot’s angle: -90
   — arm 1 extension: 0.6458

— press is in middle position.

4. To get a blank from the press with arm 2 one have to make sure:
   — robot’s angle: 35
   — arm 2 extension: 0.7971
   — press in bottom position.
5. Arm 2 can release the blank on the deposit belt when:
   — robot’s angle between -90 and -45
   — arm 2 extension: 0.5707
6. The crane can pick up the blank from the deposit belt when:
   — the crane moves over the deposit belt, the crane should be stopped when the corresponding photoelectric barrier becomes active
   — crane’s height: 0.9450.
7. The crane can deliver the blank on the feed belt when:
   — the crane moves over the feed belt, the crane should be stopped when the corresponding photoelectric barrier becomes active
   — crane’s height: 0.6593.

References

[1] Thomas Lindner, Case Study Production Cell: Task Definition, Forschungszentrum Informatik, Haid-und-Neu-Straße 10-14, D-76131 Karlsruhe.