Production Cell Case Study

An ASM Solution

Egon Börger

Dipartimento di Informatica, Universita di Pisa
http://www.di.unipi.it/~boerger
Production Cell Case Study Goals

- **Ground Model** construction for inspection by application domain expert
- **Stepwise Refinement** to executable code whose module structure reflects the decomposition of the ground model into components
- **Verification** of required controller properties
- **Validation** of code by experimentation with the FZI production cell simulator
- **Documentation** of the design at each level
For details see Chapter 5 (Synchronous Multi-Agent ASMs) of:

E. Börger, R. Stärk

Abstract State Machines
A Method for High-Level System Design and Analysis

Springer-Verlag 2003

For update info see AsmBook web page:

http://www.di.unipi.it/AsmBook
Reflecting the Component Structure

• …the production cell is composed of two conveyor belts, a positioning table, a two-armed robot, a press, and a travelling crane. Metal plates inserted in the cell via the feed belt are moved to the press. There, they are forged and then brought out of the cell via the other belt and the crane.

• Therefore one basic ASM is defined for each device
  – with precise interfaces for device interaction and explicitly stated physical assumptions

• Wlog the entire system is viewed as synch ASM
  – given that each device interacts in a sequential manner with exactly one predecessor and one successor device
Feed Belt Description

• ...The task of the feed belt consists in transporting metal blanks to the elevating rotary table. The belt is powered by an electric motor, which can be started up or stopped by the control program. A photoelectric cell is installed at the end of the belt; it indicates whether a blank has entered or left the final part of the belt. ... the photoelectric cells switch on when a plate intercepts the light ray. Just after the plate has completely passed through it, the light barrier switches off. At this precise moment, the plate ... has just left the belt to land on the elevating rotary table---provided of course that the latter machine is correctly positioned ... the feed belt may only convey a blank through its light barrier, if the table is in loading position ... do not put blanks on the table, if it is already loaded ...
Deposit Belt Description

• … The task of the deposit belt is to transport the work pieces unloaded by the second robot arm to the traveling crane. A photoelectric cell is installed at the end of the belt; it reports when 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 traveling crane has picked up the work piece. ... photoelectric cells switch on when a plate intercepts the light ray. Just after the plate has completely passed through it, the light barrier switches off. At this precise moment, the plate is in the correct position to be picked up by the traveling crane ...
Modelling Transport Belts for Reuse

- Abstracting from motors and peculiar timing of reaction to sensor changes

  \[ \text{PieceAtLightBarrier} \equiv \text{PhotoelectricCell=on} \]

  For DepBelt: when light barrier switches off

  \[ \text{PieceDeliverable} \equiv \text{TableReadyForLoading} \]

  \[ \text{DeliverPiece} \equiv \begin{cases} \text{FeedBeltFree:=true} \\ \text{TableLoaded:=true} \\ \text{Critical Run} \end{cases} \]

  For DepBelt: DepBeltLoadable:=true
Elevating Rotary Table Description

• ...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.
Modelling Elevating Rotary Table

• Abstracting from motors, rotation and lifting

- StoppedIn
- LoadPos
- Table
- Loaded
- MoveTo
- Unload Pos
- StoppedIn
- UnloadPos
- MoveTo
- Load Pos
- not Table
- Loaded

• Making Movement durative:
  MoveToActionPos ≡

MovingTo
ActionPos
ActionPos
Reached

© Egon Börger: Production Cell ASM
Robot Description

• 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 center. The end of each robot arm is fitted with an electromagnet that allows the arm to pick up metal plates. The robot's task consists in: taking metal blanks from the elevating rotary table to the press; transporting forged plates from the press to the deposit belt.
Modelling Robot: Device Action Rules

- Abstracting from motors, movement details, magnet

- Defining $\text{DevAction} \rightarrow \text{Next(DevAction)}$ (the suggested order):
  TableUnload $\rightarrow$ PressUnload $\rightarrow$ DepBeltLoad $\rightarrow$ PressLoad

- Yield: $\text{TableLoaded} := \text{false}$  $\text{PressLoaded} := \text{false}$
  $\text{DepBeltLoadable} := \text{false}$  $\text{PressLoaded} := \text{true}$
## Robot Action Macros

<table>
<thead>
<tr>
<th>DevAction</th>
<th>Next (DevAction)</th>
<th>Yield (DevAction)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableUnload</td>
<td>PressUnload</td>
<td>TableLoaded:=false</td>
</tr>
<tr>
<td>PressUnload</td>
<td>DepBeltLoad</td>
<td>PressLoaded:=false</td>
</tr>
<tr>
<td>DepBeltLoad</td>
<td>PressLoad</td>
<td>DepBeltLoadable:=false</td>
</tr>
<tr>
<td>PressLoad</td>
<td>TableUnload</td>
<td>PressLoaded:=true</td>
</tr>
</tbody>
</table>
Robot Interface Predicates

• DevReadyForUnload \equiv \begin{cases} \text{forDev} = \text{Table, Press} \\ \text{DevInUnloadPos and DevLoaded} \end{cases}

  – DevInUnloadPos and DevLoaded
    • TableInUnloadPos \equiv \text{ERT.ctl_state} = \text{StoppedInUnloadPos}
    • PressInUnloadPos \equiv \text{Press.ctl_state} = \text{OpenForUnload}

• PressReadyForLoad \equiv \begin{cases} \text{PressInLoadPos and not PressLoaded} \end{cases}

  – PressInLoadPos and not PressLoaded
    • TableInUnloadPos \equiv \text{ERT.ctl_state} = \text{StoppedInUnloadPos}
    • PressInLoadPos \equiv \text{Press.ctl_state} = \text{OpenForLoad}

• DepBeltReadyForLoad = (\text{DepBeltLoadable} = \text{true})
Press Description

• …The task for the press is to forge metal blanks. The press consists of two horizontal plates, with the lower plate being 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: 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. This processing sequence is carried out cyclically.
Modelling Press

• Abstracting from movement details

OpenFor Unload \rightarrow \text{not Press Loaded} \rightarrow \text{MoveTo MiddlePos} \rightarrow \text{OpenFor Load} \rightarrow \text{Press Loaded}

\text{MoveTo BottomPos} \rightarrow \text{Forging Completed} \rightarrow \text{ClosedFor Forging} \rightarrow \text{MoveTo TopPos}

• Making movement durative:
  \text{MoveToAnyPos} \equiv \text{MovingTo AnyPos} \rightarrow \text{AnyPos Reached}
Verification of Ground Model Properties

- **Safety Properties** proved from assumptions on device placements, movements, actions
  - no movement outside its bounds, no crash between devices, no action unless safe, etc.

- **Liveness** proved: every blank put on the FeedBelt will eventually be forged and picked up by the crane at the end of the DepBelt
  - exploiting the sequential order of travel of blanks

- **Maximal Performance** (maximal throughput in minimal time) proved from assumptions on the time of device actions, movements and insertion of blanks to the FeedBelt
Data Refinement of Feed Belt Movement as Motor Driven

• Definition:
  – ctl_state = Run ≡ FeedBeltMot = on & not Delivering
  – ctl_state = CriticalRun ≡ FeedBeltMot = on & Delivering
  – ctl_state = Stop ≡ FeedBeltMot = off

• Correspondingly the ctl-state updates become (non-optimized):
  – ctl_state := Run ≡ FeedBeltMot := on & Delivering:=false
  – ctl_state := CriticalRun ≡ FeedBeltMot := on & Delivering := true
  – ctl_state := Stop ≡ FeedBeltMot := off
(1,2) - Refinement of Elevating Rotary Table by Vertical and Horizontal, Motor Driven Movement

• $\text{ctl\_state} = \text{StoppedInLoadPos} \equiv$
  - $\text{BottomPosition} \& \text{MinRotation} \& \text{TableElevationMot} = \text{TableRotationMot} = \text{Idle}$

• $\text{ctl\_state} = \text{StoppedInUnloadPos} \equiv$
  - $\text{TopPosition} \& \text{MaxRotation} \& \text{TableElevationMot} = \text{TableRotationMot} = \text{Idle}$

• $\text{ctl\_state} = \text{MovingToLoadPos} \equiv$
  - $\text{TableElevationMot} = \text{Up} \text{ or } \text{TableRotationMot} = \text{clockwise}$

• $\text{ctl\_state} = \text{MovingToUnloadPos} \equiv$
  - $\text{TableElevationMot} = \text{Down} \text{ or } \text{TableRotationMot} = \text{counterClockwise}$
Signature for Robot Refinement by Rotation & Extension/Retraction of Magnetic Arms

- **RobotRotationMot** with **Angle** measuring rotation
  - Angle values where rotation stops for arm movement:
    - Arm1ToTable, Arm2ToPress, Arm2ToDepBelt, Arm1ToPress (in the suggested rotation order)

- **Arms which can be extended/retracted**
  - Arm1Mot to reach Table/Press with Arm1
  - Arm2Mot to reach Press/DepBelt with Arm2
  - Sensor **ArmiExt** measuring extension of Armi (i =1,2) with 4 values where extension/retraction stops
    - Arm1AtTable, Arm2AtPress, Arm2AtDepBelt, Arm1AtPress (in the suggested rotation order)
  - Magnets **ArmiMagnet** ∈ {on, off} to pick up or release pieces (i =1,2)
Refinement of Robot ctl states WaitingFor/MovingTo

<table>
<thead>
<tr>
<th>WaitingForDevAction</th>
<th>Angle</th>
<th>Arm1Magnet</th>
<th>Arm2Magnet</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableUnload</td>
<td>Arm1ToTable</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>PressUnload</td>
<td>Arm2ToPress</td>
<td>on</td>
<td>off</td>
</tr>
<tr>
<td>DepBeltLoad</td>
<td>Arm2ToDepBelt</td>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>PressLoad</td>
<td>Arm1ToPress</td>
<td>on</td>
<td>off</td>
</tr>
</tbody>
</table>

and ArmsRetracted and RobotIdle

<table>
<thead>
<tr>
<th>MovingToDevAction</th>
<th>RobotRotationMot</th>
<th>Arm1Magnet</th>
<th>Arm2Magnet</th>
<th>Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>PressUnload</td>
<td>counterClock</td>
<td>on</td>
<td>off</td>
<td>[Arm1ToTable,Arm2ToPress]</td>
</tr>
<tr>
<td>DepBeltLoad</td>
<td>counterClock</td>
<td>on</td>
<td>on</td>
<td>[Arm2ToPress,Arm2ToDepBelt]</td>
</tr>
<tr>
<td>PressLoad</td>
<td>counterClock</td>
<td>on</td>
<td>off</td>
<td>[Arm2ToDepBelt,Arm1ToPress]</td>
</tr>
<tr>
<td>TableUnload</td>
<td>clockwise</td>
<td>off</td>
<td>off</td>
<td>[Arm1ToTable,Arm1ToPress]</td>
</tr>
</tbody>
</table>

and ArmsRetracted and Arm1Mot = Arm2Mot = Idle

<table>
<thead>
<tr>
<th>ReachedPosForDevAction</th>
<th>PressUnload</th>
<th>DepBeltLoad</th>
<th>PressLoad</th>
<th>TableUnload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angle</td>
<td>Arm2ToPress</td>
<td>Arm2ToDepBelt</td>
<td>Arm1ToPress</td>
<td>Arm1ToTable</td>
</tr>
</tbody>
</table>

• RobotIdle ≡ RobotRotationMot = Arm1Mot = Arm2Mot = idle
• ArmsRetracted ≡ Arm1Ext = Arm2Ext = retracted
Data Refinement of Robot Waiting/Moving Rule

• **Waiting** ≡
  If WaitingForDevAction & DevReadyForAction then Arm(DevAction)Mot:= extend

• **Moving** ≡

• If MovingToDevAction & ReachedPosForDevAction then RobotRotationMot:= idle
(1,3) - Refinement of Robot Actions: arm extension

• **Action.Extension** \(\equiv\)
  If extending \textit{ArmToDev} & \textit{ArmExt} = \textit{ArmAtDev}
  Then \textit{ArmMot} := idle

• where
  moving \textit{ArmToDev} \(\equiv\)
  \textit{Angle} = \textit{ArmToDev} & \textit{ArmMot} = \textit{mov}
  for \textit{mov} = extend,retract
  
  \textit{ArmToDev} = \textit{Arm1ToTable}, \textit{Arm2ToPress},
  \textit{Arm2ToDepBelt}, \textit{Arm1ToPress}

  \textit{ArmAtDev} = \textit{Arm1AtTable}, \textit{Arm2AtPress},
  \textit{Arm2AtDepBelt}, \textit{Arm1AtPress}
(1,3) - Refinement of Robot Actions: action proper

• **Action.Proper** \(\equiv\)

  If extended ArmAtDev Then
  
  ArmMagnet := on for ArmAtDev=Arm1AtTable, Arm2AtPress
  
  off for ArmAtDev=Arm2AtDepBelt, Arm1AtPress
  
  ArmMot := retract
  
  Yield Act(ArmAtDev) \(\text{NB. Corrects a mistake found by model checking (Plonka 2000)}\)

• where **extended ArmAtDev** \(\equiv\)

  Angle = ArmToDev & ArmExt = ArmAtDev
  
  & ArmMot = idle

  Act(ArmAtDev) \(\equiv\) TableUnload, PressUnload, DepBeltLoad, PressLoad

  for ArmAtDev = Arm1AtTable, Arm2AtPress,
  
  Arm2AtDepBelt, Arm1AtPress
(1,3) - Refinement of Robot Actions: retraction

- **Action.Retraction**
  
  If retracting ArmToDev & ArmExt = retracted
  
  Then

  RobotRotationMot :=
  
  clockwise if Arm=Arm1 & Dev=Press
  
  counterClock else

  ArmMot := idle

- for ArmToDev = Arm1ToTable, Arm2ToPress,
  
  Arm2ToDepBelt, Arm1ToPress
Exercise: Press Data Refinement by Detailing Motor Driven Movement and Being Open/Closed

- $\text{ctl\_state} = \text{OpenForAction} \equiv$
  - $\text{PressMot} = \text{Idle} \ & \ \text{ActionPos}$
    - where $\text{UnloadPos} = \text{BottomPos}$, $\text{LoadPos} = \text{MiddlePos}$
- $\text{ctl\_state} = \text{ClosedForForging} \equiv$
  - $\text{PressMot} = \text{Idle} \ & \ \text{TopPos}$
- $\text{ctl\_state} = \text{MovingToMiddlePos} \equiv$
  - $\text{PressMot} = \text{Up} \ & \ \text{not} \ \text{PressLoaded}$
- $\text{ctl\_state} = \text{MovingToTopPos} \equiv$
  - $\text{PressMot} = \text{Up} \ & \ \text{PressLoaded}$
- $\text{ctl\_state} = \text{MovingToBottomPos} \equiv \ \text{PressMot} = \text{Down}$
Implementation of the Refined Model

• **Structure Mapped from Machine To Code**: One C++ module per submachine, plus a module for handling errors reported by the simulation

• **Sequentialization**: decide upon the order in which the controller asks the simulation env for new status information and forwards it to the seven modules to execute them:
  – since the component dependencies are completely described in the interfaces, wlog we can follow the order used by the simulator
    • each machine module gets its updated sensor readings from the standard input, in the order specified by the status vector in the simulator
#include "control.h"

cElist Errors;
cFeedBelt FeedBelt;
cElevRotTable ElevatingRotaryTable;
cRobot Robot;
cPress Press;
cDepositBelt DepositBelt;
cTravCrane TravelingCrane;

int main()
{
    loop
    {
        AskNewStatus();
        cin>> Press
           >> Robot
           >> ElevatingRotaryTable
           >> TravelingCrane
           >> FeedBelt
           >> DepositBelt
           >> Errors;
        Oper();
    }
}
Code Validation

• **Correctness** The code was extensively tested by Luca Mearelli, correctly controlling the simulator at FZI in Karlsruhe during night long experiments

• **Safety Properties** All test runs worked conforming to all the safety requirements
  – **Maximal Performance**: the maximum throughput of 7 pieces circulating in the system was achieved

• **Inspection** The code was inspected at the Dagstuhl Seminar 9720 (12.5.-16.5.1997) on “Practical Methods for Code Documentation and Inspection” (See Report 178, edited by E. Börger, P. Joannou, D. Parnas)
References

• C. Lewerentz and T. Lindner (eds.), Formal Development of Reactive Systems.

• E. Börger and L. Mearelli, Integrating ASMs into the Software Development Life Cycle

• L. Mearelli, Refining an ASM Specification of the Production Cell to C++ Code

• FZI simulator for the production cell
  http://www.fzi.de/divisions/prost/projects/production_cell/ProductionCell.html

• C++ code for Mearelli’s program which controls the FZI simulator
  http://www.fzi.de/prost/projects/production_cell/contributions/ASM.html
References


• K. Winter, Model Checking for Abstract State Machines
  – J.of Universal Computer Science 3(5) 1997, 689—701

• C. Plonka, Model Checking for the Design with Abstract State Machines
  – Diplomarbeit, University of Ulm, January 2000