Modelling the Production Cell Case Study using the Fusion Method

Stéphane Barbey, Didier Buchs and Cécile Péraire

Swiss Federal Institute of Technology
Software Engineering Laboratory
1015 Lausanne Ecublens
Switzerland

email: {stephane.barbey, didier.buchs, cecile.peraire} @ epfl.ch

Abstract. *The purpose of this work is to evaluate, by means of a case study, our formal testing method for object-oriented software. For this purpose we propose the development of an object-oriented application of realistic size, addressing all the phases of the software life-cycle: requirements, analysis, design, formal description, implementation and testing. Analysis and design are performed with the Fusion method [4], formal description with the CO-OPN language [1], implementation with Ada 95 and test selection with our testing method. The case study chosen for this experiment is the production cell, originally defined in [6]. This report presents the description of the production cell case study and Fusion models. The formal description and testing phase are presented in [4].*

Keywords. *object-orientation, formal testing, analysis, design, formal specification, implementation, production cell.*

1. Introduction

The purpose of this work is to evaluate, by means of a case study, our formal testing method for object-oriented software. For this purpose we propose the development of an object-oriented application of realistic size, addressing all the phases of the software life-cycle: requirements, analysis, design, formal description, implementation and testing. Analysis and design are performed with the Fusion method [2], formal description with the *CO-OPN* language [1], implementation with Ada 95 and test selection with our testing method. The case study chosen for this experiment is the production cell, originally defined in [3].

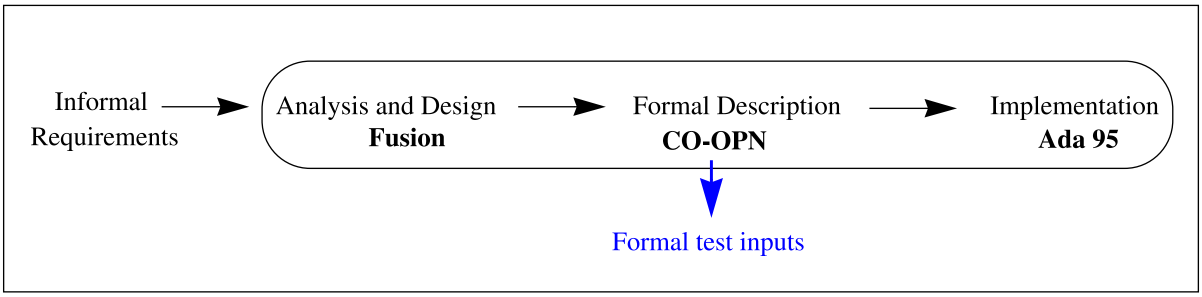

Fig. 1. Case study development life-cycle

We have chosen Ada 95 as the implementation language, but it could be any other object-oriented language. However this choice will influence the analysis phase.

The structure of this paper is the following. Section 2 describes the production cell case study. Section 3 gives the Fusion analysis of the production cell controller, section 4 gives the Fusion design and section 5 gives the Fusion data dictionary. The formal description and testing phase are presented in [4].

2. Requirements

This section is a summary of [3].

2.1 Description of the Cell

The production cell is composed of six machines: two conveyors belts (feed belt and deposit belt), a travelling crane having an extendable arm equipped with electromagnet, an elevating rotary table, a press and a rotary robot having two orthogonal extendable arms equipped with electromagnet (see figure 2). The aim of the cell is the transformation of metal blanks into forged plates (by means of a press) and their transportation from the feed belt into a container.

The production cycle of each blank is the following (see figure 3):

- the feed belt conveys the blank to the table,
- the table rotates and lifts to put the blank in the position where the robot is able to magnetize it,
- the first robot arm magnetizes the blank and places it into the press,
- the press forges the blank,
- the second robot arm places the resulting plate on the deposit belt,
- the crane magnetizes the plate and bring it from the deposit belt into a container.

Note that in the original case study proposed by the FZI, the crane magnetizes the plate and bring it from the deposit belt back to the feed belt; this in order to perform demonstration without operator. In the real cell, the crane is not between the two belts, but links the cell with a further manufacturing unit (modeled in our case by a container).

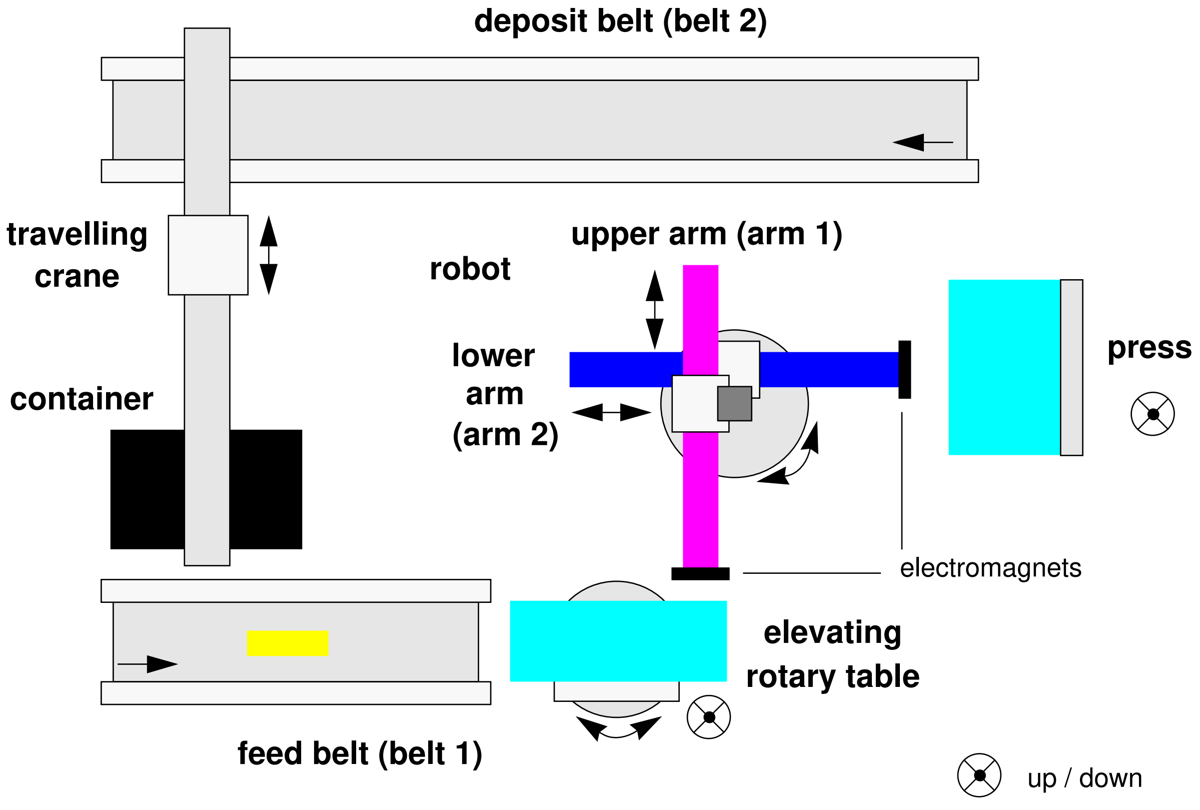

Fig. 2. Top view of the production cell

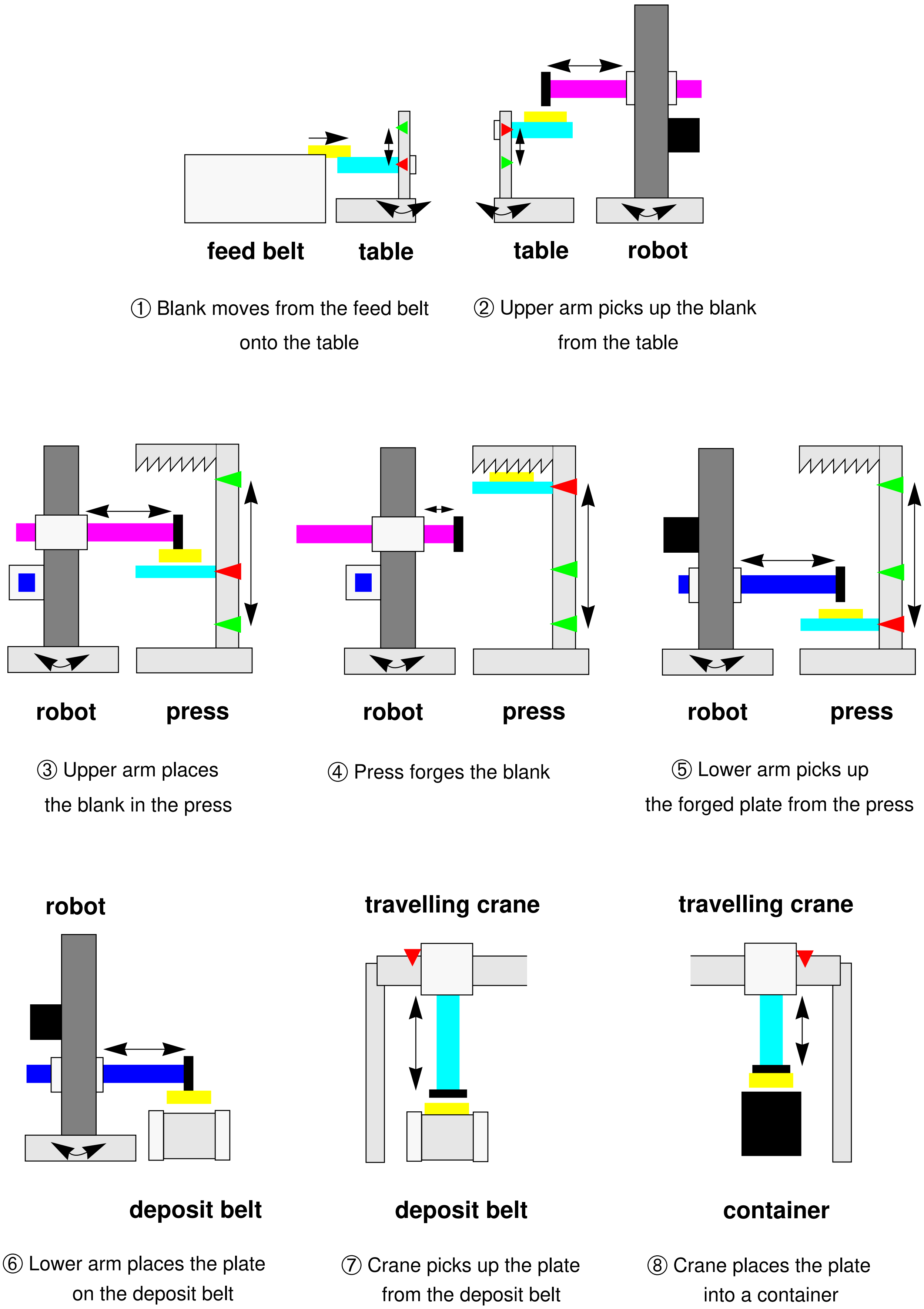

Fig. 3. Production cycle of a blank (side view)

2.1.1 Description of the Devices

- **Table**

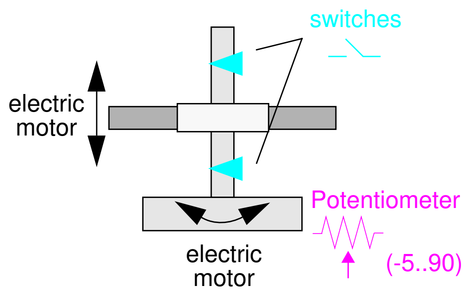

Fig. 4. Elevating rotary table (side view)

The table (see figure 4) is powered by two bidirectional electric motors which allow the vertical translation and the rotation of the table. The motors can be started and stopped by the control program. The vertical translation is necessary because the feed belt and the first robot arm are located at different levels and they are not vertically mobile (see figure 3 steps 1 and 2). The rotation of the table is required to make sure that the blank will have the good position into the press. The vertical position of the table is given by two switches and its angle of rotation by a potentiometer.

- **Robot**

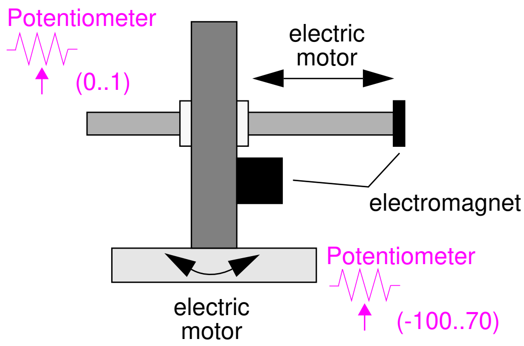

Fig. 5. Robot (side view)

The rotary robot (see figure 5) consists of two orthogonal extendable arms equipped with an electromagnet. The robot is powered by three bidirectional electric motors which allow the rotation of the robot and the horizontal translation of the arms (extension or retraction). The motors can be started and stopped by the control program. The angle of rotation of the robot and the value of extension of each arm is given by potentiometers. In order to meet various safety requirements, each arm has to be retracted while the robot rotates and while the other arm performs loading or unloading a blank.

- **Press**

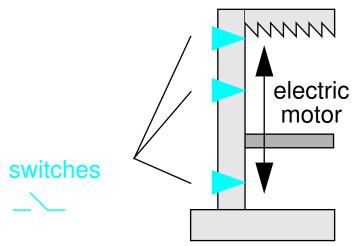

Fig. 6. Press (side view)

The press (see figure 6) consists of two horizontal plates. The press forges blanks by pressing its lower plate against the upper plate by means of a bidirectional electric motor which can be started and stopped by the control program. The press has three positions (see figure 3 steps 3-4-5): a middle position where the press is loaded by the upper arm of the robot, an upper position where the press forges a blank, and a lower position where the press is unloaded by the lower arm of the robot. The vertical position of the press is given by three switches.

- **Crane**

The crane (see figure 7) has an electric magnet which can perform horizontal and vertical translations by means of two bidirectional electric motors which can be started and stopped by the control program. Horizontal mobility serves to cover the horizontal distance between the deposit belt and the container, while vertical mobility is necessary because the deposit belt and the container are placed at different levels. The horizontal position of the crane is given by two switches and the vertical position by a potentiometer. For efficiency considerations, the crane is positioned above the deposit belt during its inactive periods.

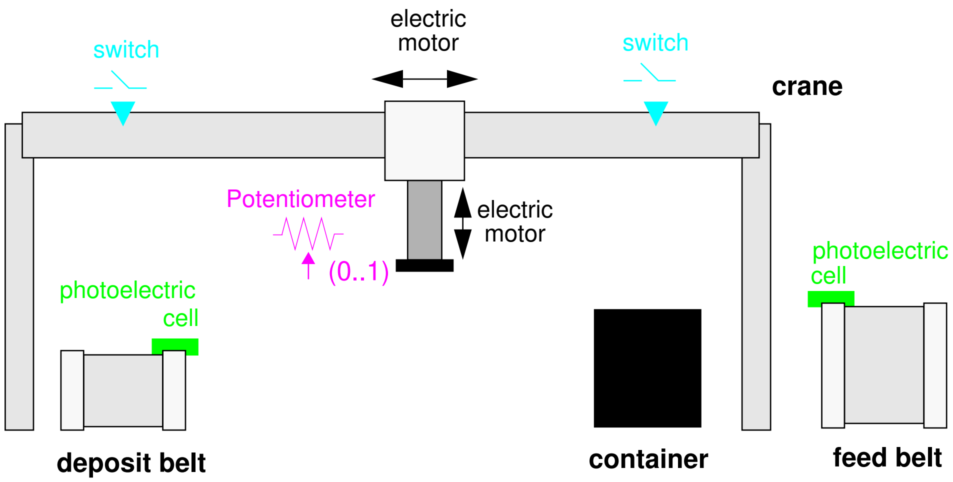

Fig. 7. Crane, Deposit belt and Feed belt (side view)

- **Feed belt**

The feed belt (see figure 7) is powered by an unidirectional electric motor which can be started and 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. For safety considerations (requirement 17), the feed belt can hold only one blank at same time.

- **Deposit belt**

The deposit belt (see figure 7) is powered by an unidirectional electric motor which can be started and stopped by the control program. A photoelectric cell is installed at the end of the belt; it indicates whether a blank has entered in the final part of the belt. For safety considerations (requirement 18), the deposit belt can hold only two plates at same time.

2.1.2 Sensors

The control program receives information from the cell by means of three kinds of sensors: switches, photoelectric cells, and potentiometers. Sensors are presented in Table 1, "Sensors," on page 5.

No	Name	Type	Device	Description	Value
1	press_bottom	switch	press	Press in lower position?	0 = no, 1 = yes
2	press_middle	switch	press	Press in middle position?	0 = no, 1 = yes
3	press_top	switch	press	Press in upper position?	0 = no, 1 = yes
4	arm1_extension	potentiometer	robot	Extension of arm 1	0 ... 1 0: the arm is fully retracted 1: the arm is fully extended
5	arm2_extension	potentiometer	robot	Extension of arm 2	0 ... 1 0: the arm is fully retracted 1: the arm is fully extended
6	robot_angle	potentiometer	robot	Angle of rotation of the robot	-100 ... 70
7	table_bottom	switch	table	Table in lower position ?	0 = no, 1 = yes

Table 1. Sensors

8	table_top	switch	table	Table in upper position ?	0 = no, 1 = yes
9	table_angle	potentiometer	table	Angle of rotation of the table	-5 ... 90
10	crane_above_belt2	switch	crane	Crane above the deposit belt ?	0 = no, 1 = yes
11	crane_above_contain	switch	crane	Crane above container ?	0 = no, 1 = yes
12	crane_mag_height	potentiometer	crane	Height of the crane's magnet	0 ... 1 0: magnet in top position 1: magnet in bottom position
13	belt1_blank_at_end	photoelectric cell	feed belt	Blank inside the photoelectric barrier?	0 = no, 1 = yes
14	belt2_blank_at_end	photoelectric cell	deposit belt	Blank inside the photoelectric barrier?	0 = no, 1 = yes

Table 1. Sensors

2.1.3 Actuators

The control program controls each machine of the cell by means of actuators (electric motors and electromagnets) using the following commands:

Device	Command name	Description
Belt	belt1_start	starts the feed belt
	belt1_stop	stops the feed belt
	belt2_start	starts the deposit belt
	belt2_stop	stops the deposit belt
Table	table_left	starts rotation to the left
	table_right	starts rotation to the right
	table_stop_h	stops the rotation
	table_upward	starts upward moving
	table_downward	starts downward moving
	table_stop_v	stops the vertical moving
Robot	robot_left	starts rotation to the left
	robot_right	starts rotation to the right
	robot_stop	stops the rotation
	arm1_forward	starts extension of arm1
	arm1_backward	starts retraction of arm1
	arm1_stop	stops arm1
	arm1_mag_on	activates magnet of arm1
	arm1_mag_off	deactivates magnet of arm1
	arm2_forward	starts extension of arm2
	arm2_backward	starts retraction of arm2
	arm2_stop	stops arm2
	arm2_mag_on	activates magnet of arm2
	arm2_mag_off	deactivates magnet of arm2
Press	press_upward	starts upward moving
	press_downward	starts downward moving
	press_stop	stops the press

Table 2. Actuators

Crane	crane_to_contain	starts horizontal movement towards the container
	crane_to_belt2	starts horizontal movement towards the deposit belt
	crane_stop_h	stops horizontal movement of the crane
	crane_lift	starts retraction of the arm
	crane_lower	starts extension of the arm
	crane_stop_v	stops vertical movement of the arm
	crane_mag_on	activates the crane's magnet
	crane_mag_off	deactivates the crane's magnet
Controller	get_status	returns a vector containing: - the values of the sensors, - an error list.

Table 2. Actuators

To allow the evaluation of the control programs of the different research groups, the FZI (Forschungszentrum Informatik) provides a simulator which imitates the important abilities of the real production cell. The FZI simulator is managed by transmitting commands to it and receiving sensor informations from it (the commands are described in table 2, "Actuators", "Actuators"). It performs the movements of the devices and blanks, detects collision and reports them by mean of the error list returned by the command *get_status*. We use a modified version of the FZI simulator in which each metal plate ends its cycle in the cell into a container.

2.2 Safety and Liveness Requirements

Mainly two types of requirements must be satisfied in a reactive system: safety and liveness requirements. Obviously, safety requirements are the most important in our setting: if a safety requirement is violated, this might result in damage of machines or injury to people. This section presents safety and liveness requirements, as well as other properties interesting in the production cell.

2.2.1 Safety Requirements

- **Restriction of machine mobility**

A machine should be stopped before the end of its possible movement, otherwise it would destroy itself. To avoid this kind of problem, the control program must make sure that the following requirements are met:

Requirement 1. *The robot must not be rotated clockwise if arm 1 points towards the table, and it must not be rotated counterclockwise if arm 1 points towards the press.*

Requirement 2. *Both robot arms must not be retracted or extended more than necessary.*

Requirement 3. *The press must not be moved downward if the sensor 1, press_bottom, is true, and it must not be moved upward if sensor 3, press_top, is true.*

Requirement 4. *The table must not be moved downward if the sensor 7, table_bottom, is true, and it must not be moved upward if sensor 8, table_top, is true.*

Requirement 5. *The table must not be rotated clockwise if it is in unloaded position (position required for transferring blanks to the robot), and it must not be rotated counterclockwise if it is in loaded position (position required to receive blanks from the feed belt).*

Requirement 6. *If the crane is positioned above the container, it may only move towards the deposit belt, and if it is positioned above the deposit belt, it may only move towards the container.*

Requirement 7. *The magnet of the crane must not be moved downward if it is in the position required for picking up a blank from the deposit belt, and it must not be moved upwards beyond a certain limit.*

- **Avoidance of machine collisions:**

Collisions are possible between the press and the robot, and between the crane and the conveyor belts. To avoid collisions, the control program must make sure that the following requirements are met:

Requirement 8. *The press may only move when no robot arm is positioned inside it.*

Requirement 9. *The robot having an arm in the proximity of the press may only rotate if this arm is retracted or if the press is in its lower or upper position.*

Requirement 10. *The magnet of the crane is not allowed to knock against the deposit belt or the container laterally.*

Requirement 11. *The magnet of the crane is not allowed to knock against the deposit belt or the container from above.*

- **Avoidance of falling metal plates**

Metal blanks can be dropped outside safe areas (belts, table, press) for two reasons:

- the electromagnets of the robot arms or of the crane are deactivated,
- a belt transports a blank too far.

To avoid this kind of problem, the control program must make sure that the following requirements are met:

Requirement 12. *The magnet of arm 1 may only be deactivated if it is inside the press.*

Requirement 13. *The magnet of arm 2 may only be deactivated if it is above the deposit belt.*

Requirement 14. *The magnet of the crane may only be deactivated if it is above the container and sufficiently closed to it.*

Requirement 15. *The feed belt may only convey a blank through its photoelectric barrier if the table is in loading position.*

Requirement 16. *The deposit belt must be stopped after a blank has passed the photoelectric barrier at its end and may only be started after the crane has picked up the blank.*

- **Avoidance of piling or overlapping plates**

Errors occur if blanks are piled on each other, overlapped, or too close for being distinguished by the photoelectric cell. To avoid theses errors, the control program must make sure that the following requirements are met:

Requirement 17. *A new blank may only be put on the feed belt if sensor 13, belt1_blank_at_end, confirms that the former one has arrived on the table.*

Requirement 18. *A plate may only be put on the deposit belt if sensor 14, belt2_blank_at_end, confirms that the former one has arrived at the end of the deposit belt.*

Requirement 19. *Blank may not be put on the table if it is already loaded.*

Requirement 20. *Blank may not be put into the press if it is already loaded.*

Requirement 21. *If the table is loaded, the robot arm 1 may not be moved above the table if it is also loaded (otherwise the two blanks collide).*

- **Predicates and constants**

To fulfil the safety restrictions, certain constants must be known. Figure 8 presents the rotation constants. It shows the angles of rotation of the table and of the robot which are required to allow the execution of the operations 1 to 5 of figure 8. Figure 9 presents the translation constants. It shows the value of translation of the arms (of the robot and of the crane) and the positions of the table, of the press and of the crane, which are required to allow the execution of the operations 1 to 8 of figure 9.

2.2.2 Liveness Requirement

The following liveness requirement must be satisfied (in case of no emergency stop):

Every plate introduced into the system via the feed belt will have been forged by the press, and will finally be dropped by the crane into the container.

There are many weaker forms of this liveness requirement.

2.2.3 Other Requirements

Efficiency. *The cell must be as efficient as possible (without violation of safety and liveness requirements).*

Flexibility. *The control program has to be as flexible as possible.*

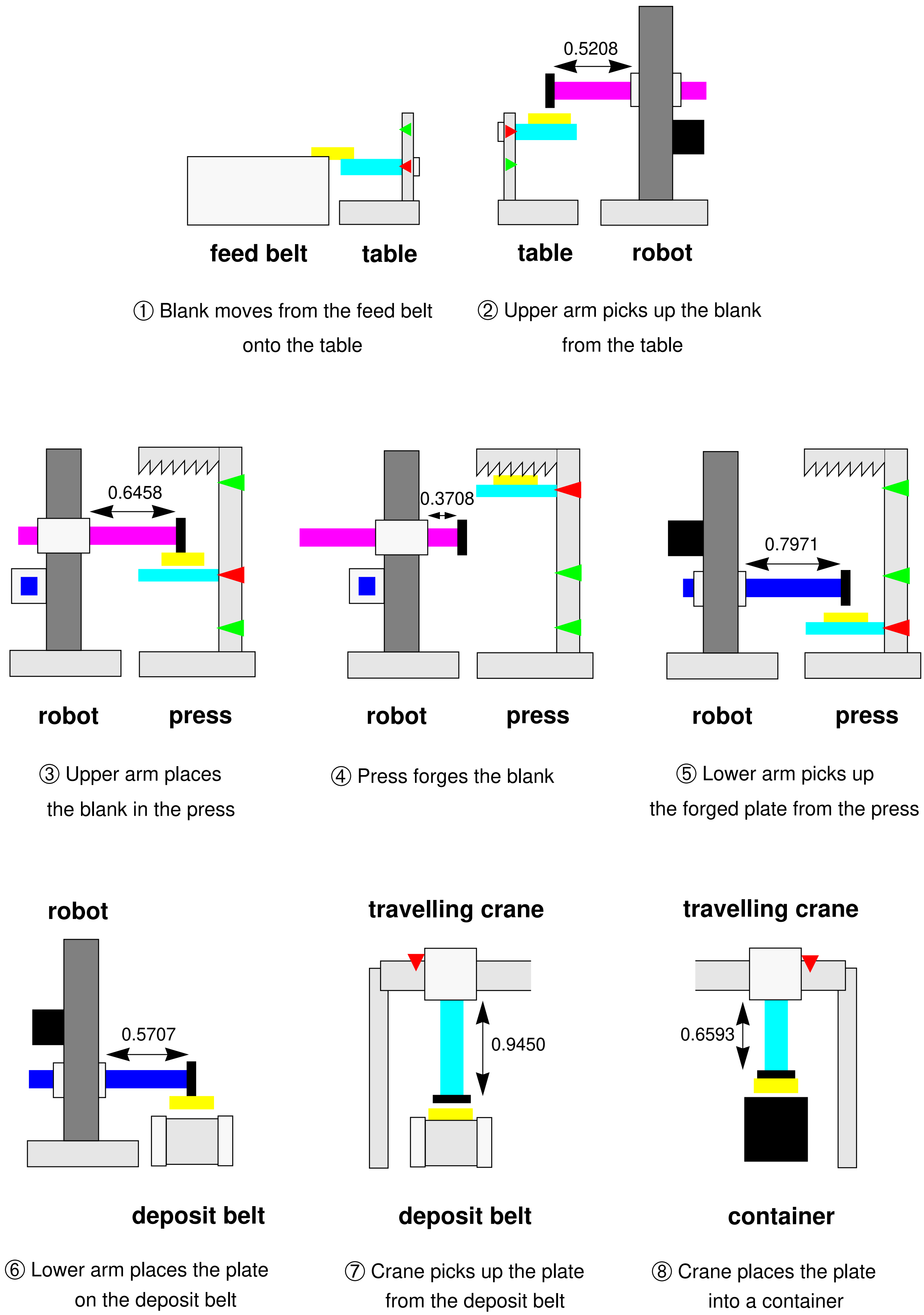

Fig. 9. Translation constants (side view)

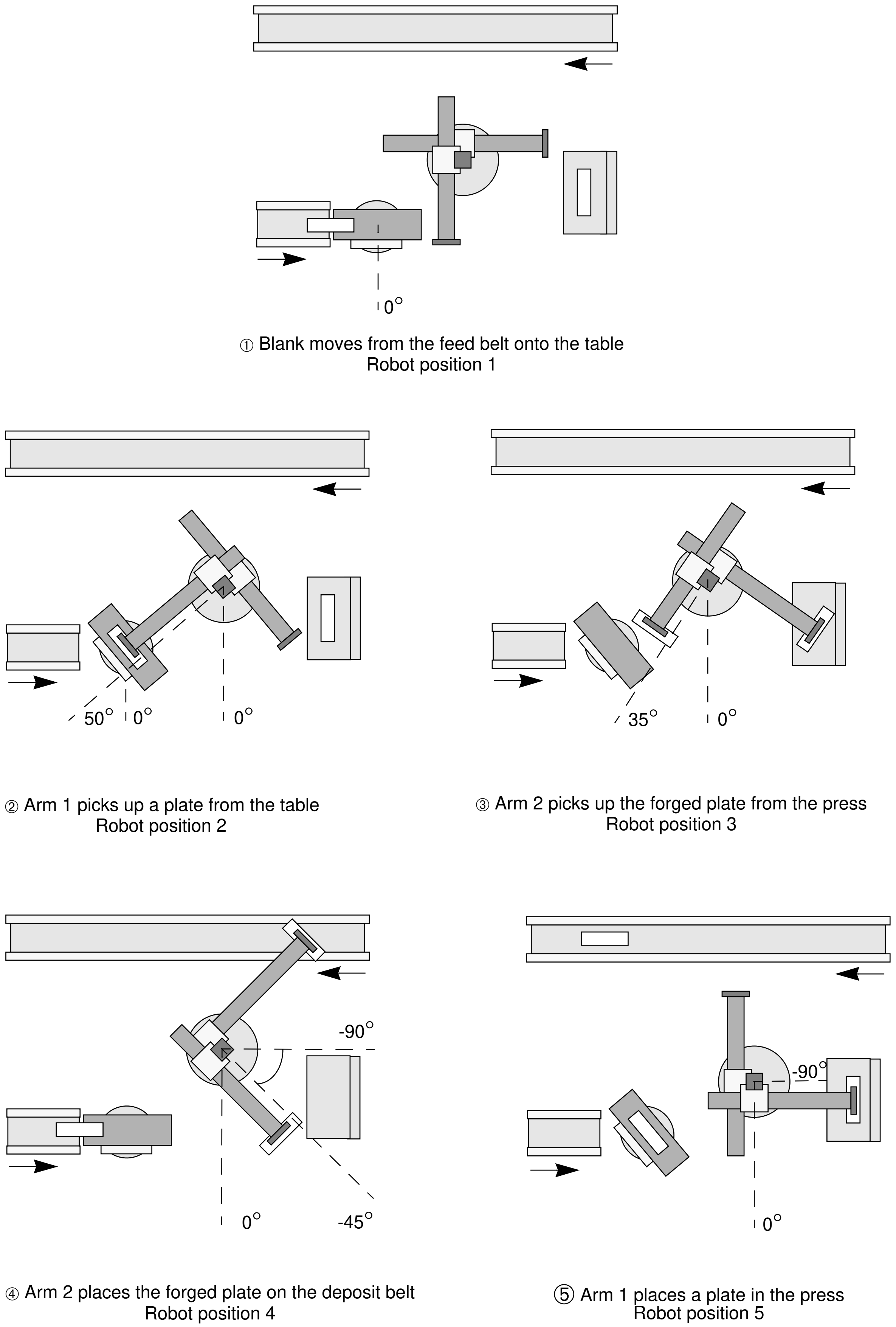

Fig. 8. Rotation constants (top view)

3. Analysis

3.1 System Context Diagram

The first system context diagram (figure 10) shows how the controller interacts with its environment.

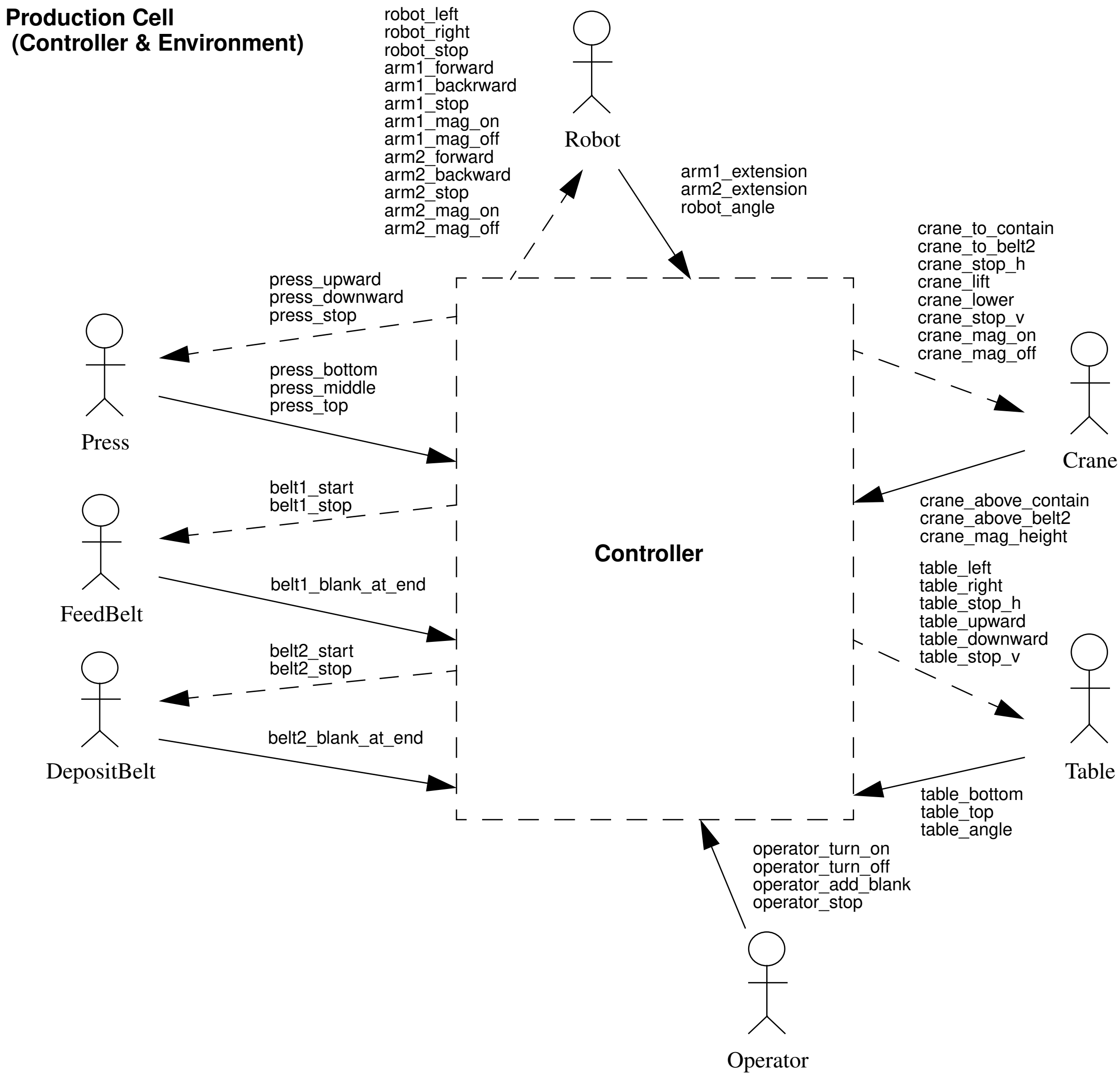

Fig. 10. System context diagram (outside view)

It shows the external agents (devices, operator) connected to the controller and what events they send to each other. The incoming events correspond to the sensors (except for the events sent by the operator), while the outgoing events correspond to commands given by the actuators.

Note that this schema does not really model the real world: the sensors are not going to send events, (not to mention know at which frequency to send one), but will wait for the controller to investigate their value. A more real approach would be to add an agent Clock (or Timer) sending tick events, to which the controller would respond by inquiring the values of the sensors.

The next schema (Figure 11) shows an inside view of the controller. Since the controller is an active concurrent system, it has been separated — as proposed in section 3.5 of the Fusion handbook [2]— to view it as a set of cooperating agents, each of which will be developed using Fusion. The inside view of the system mimic its outside view: to each device corresponds an agent, which will be responsible for interacting with the "real" device appearing on figure 10. Thus the incoming and outgoing events are not shown on figure 11. The events *TurnOn* and *TurnOff* are sent by the operator to all devices in the controller (for creating and initializing them), and are not represented either.

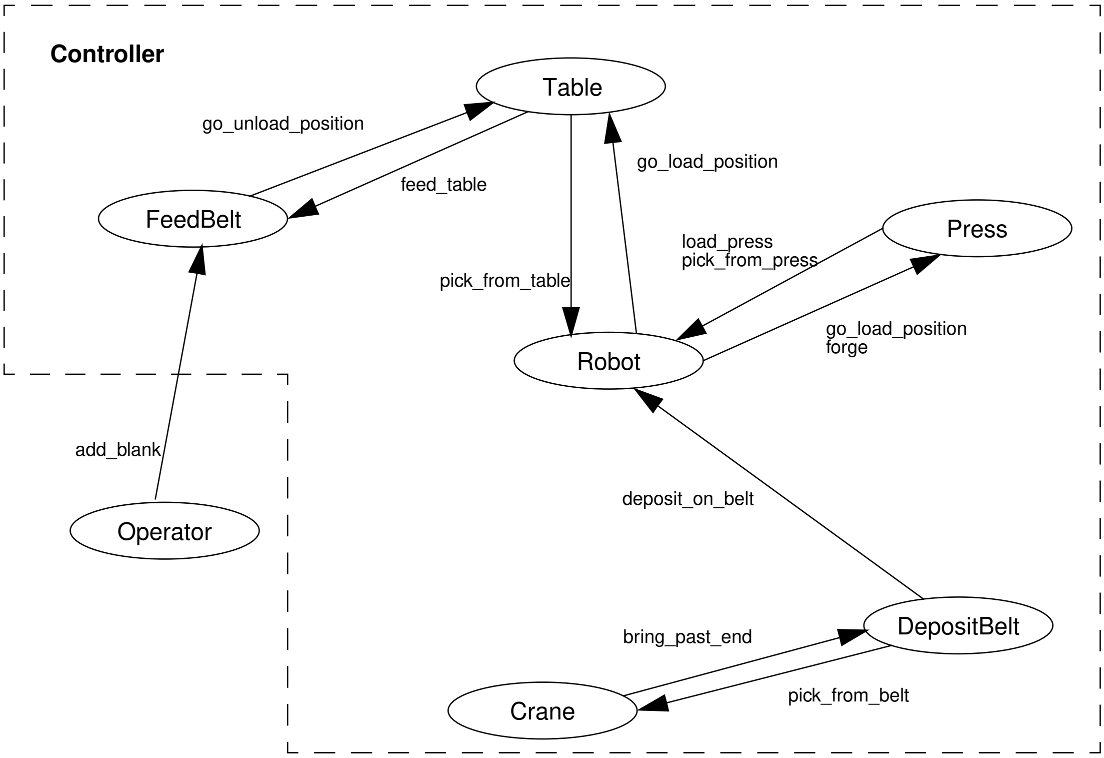

Fig. 11. System Context Diagram (Inside)

The signification of the arrows is equivalent to the signification in sequential Fusion: it is admitted that to each arrow corresponds an asynchronous event, i.e. the event will be sent even though the receiving agent is not ready to accept the event. It is admitted that events are blocking, i.e. the sending agent is blocked until the receiving agent will be able to accept the event.

The principle behind event generation is that every agent is intelligent, and will do as many actions as it can independently, and send events to connected agents as soon as it is ready to accept an interaction with them. For example, the table is initialized in load position, and, as soon as it reaches this state, informs the feed belt that it is ready to accept plates by sending it a *feed_table* event.

Possible sequences of events are shown on the time-line diagrams on figure 12, figure 13, and figure 14.

The Figure 12 shows the interactions between the feed belt, the table and the robot. After initialization, the table is in load position and an event *feed_table* is sent to the feed belt. When an event *add_blank* comes to the feed belt, a new plate "enters" the system (i.e. it is created) and it is brought to the end of the belt. Then, the feed belt can react to the *feed_table* event by moving the plate onto the table and enjoining it to go in unload position (*go_unload_position*). The feed belt is then ready to accept new plates from the operator. The feed belt keeps moving on. Once the table is in unload position, it commands the robot to pick the element from the table (*pick_from_table*). After unloading the plate from the table, the robot orders it to go in load position (*go_load_position*), after which the table asks the feed belt for another Plate (*feed_table*).

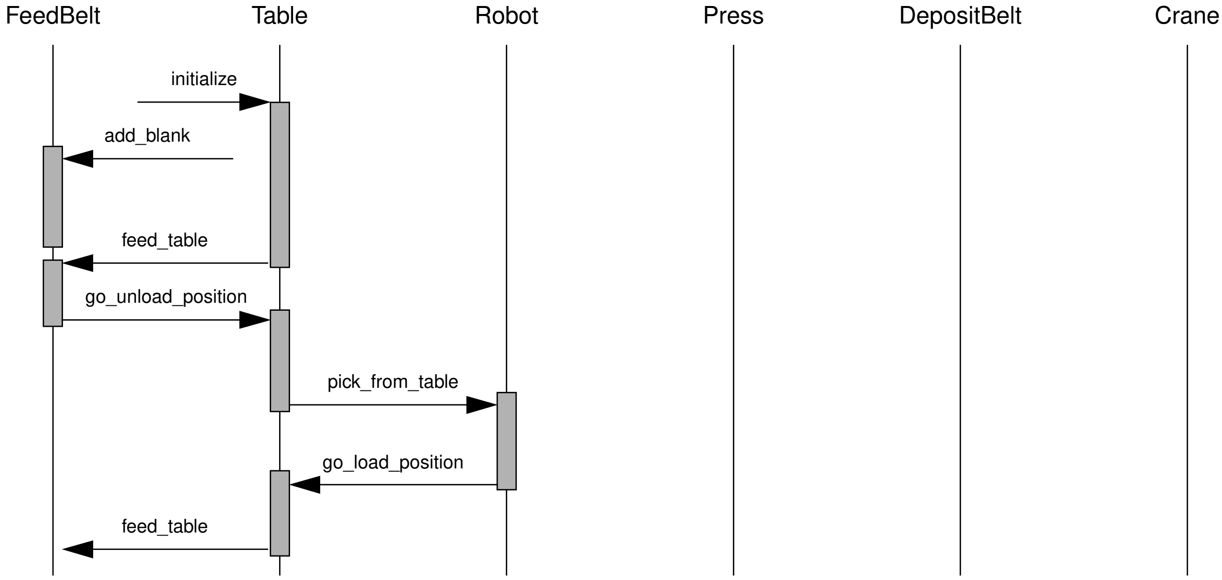

Fig. 12. FeedBelt, Robot and Table Interactions

The figure 13 shows the interactions between the robot and the press. The press is initialized in load position, and sends an event *load_press* to the robot as soon as it is ready. Once the robot has deposited a plate in the press, it sends an event *forge* to the press, and stays ready for another event (for example to load another plate from the table, or deposit a plate on the deposit belt). Once the plate is forged, the press goes by itself in unload position, and asks the robot to pick the forged plate (*pick_from_press*). Once the robot has taken the plate, it enjoins the press to go in load position (*go_load_position*), where it is ready to accept new plates to forge.

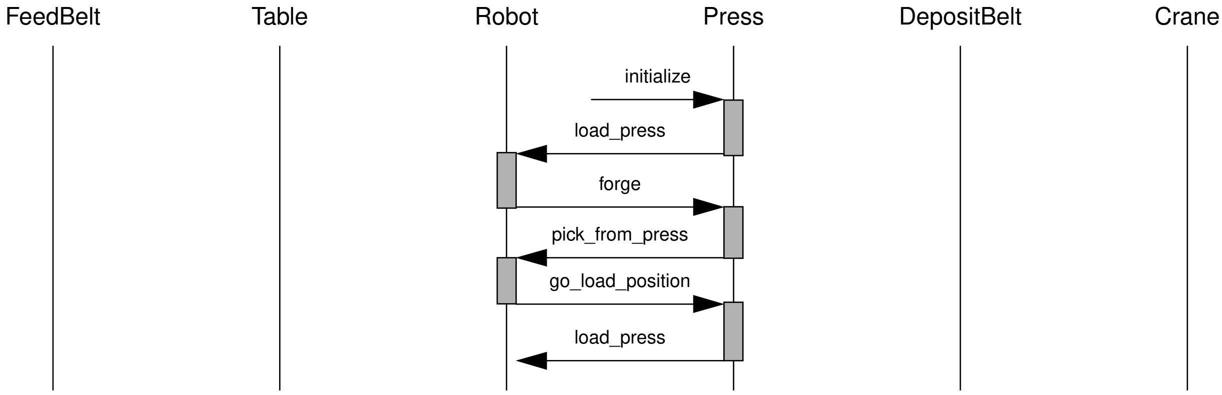

Fig. 13. Robot and Press Interactions

The last scenario, in figure 14, shows the interactions between the deposit belt, the robot, and the crane. After initialization, the deposit belt asks the robot to load it with a plate (*deposit_on_belt*), and keeps moving until a plate reaches the end of the belt. At which point, it informs the crane that a plate is ready to be picked up (*pick_from_belt*), and stops. Then the crane picks up the plate and informs the deposit belt to bring the next plate to the end (*bring_past_end*). The crane then moves onto the container, drops the plate into the container and positions itself on the deposit belt.

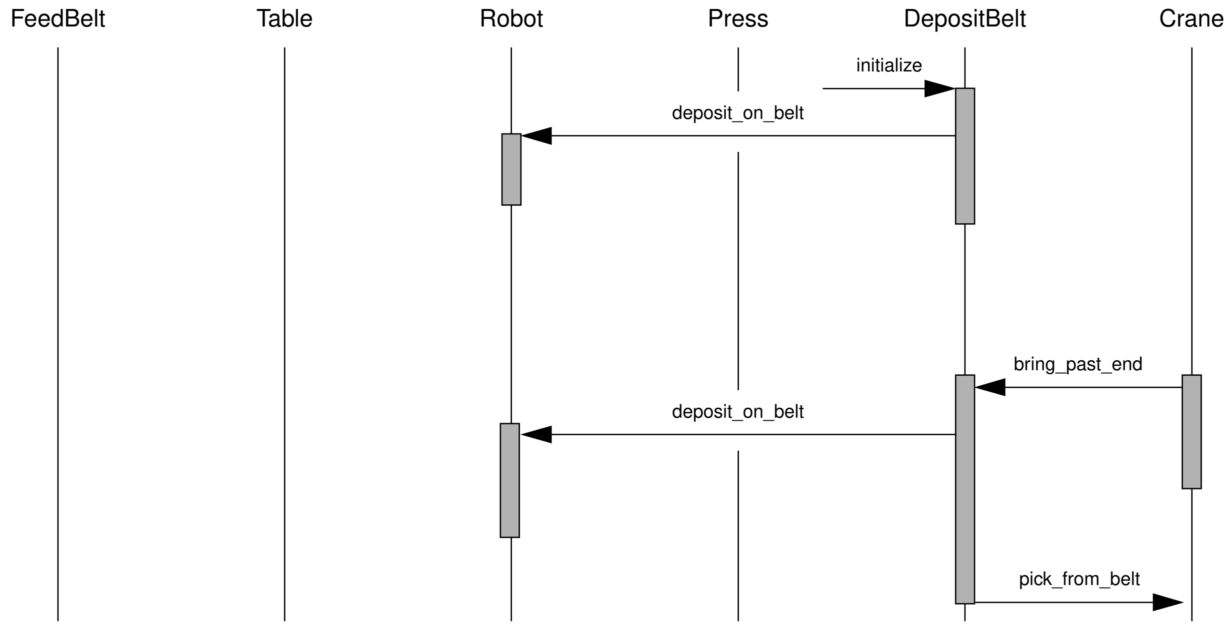

Fig. 14. Robot, DepositBelt and Crane interactions

3.2 Object Model

The Figure 15 shows the main object model: since there are no relationships between the devices, but between the devices and the plates that travel through the production cell, the corresponding object model is similar to a kind a web, at the center of which would be the plates. Each class of device is modeled as an aggregate including the concerned actuators and sensors.

The relationship between the feed belt and the plate actually indicates that a plate lays at the end of the feed belt (i.e. in front of the sensor).

There can be two plates on the deposit belt, one at the start and one at the end, this position is indicated by an attribute in the association.

A property of this schema is that no plate should belong to more than one association at the same time (i.e. the plate cannot be simultaneously on more than one device.).

The schemata in Figure 16 and Figure 17 correspond to subtyping relationships that could not be placed on the main schema.

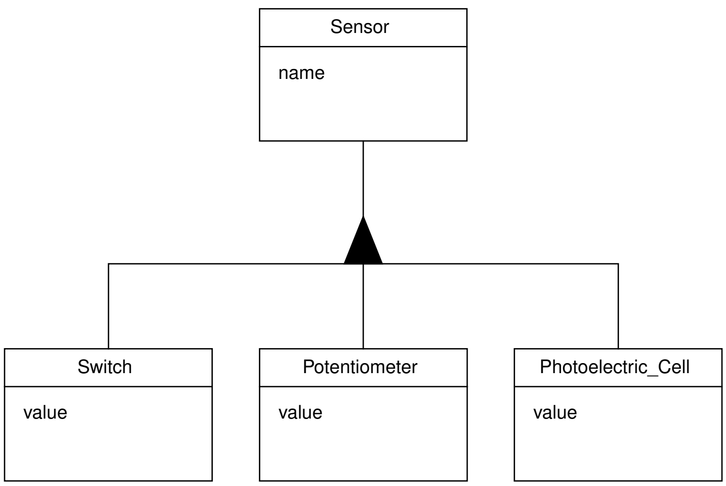

Fig. 16. Object Model (Sensors)

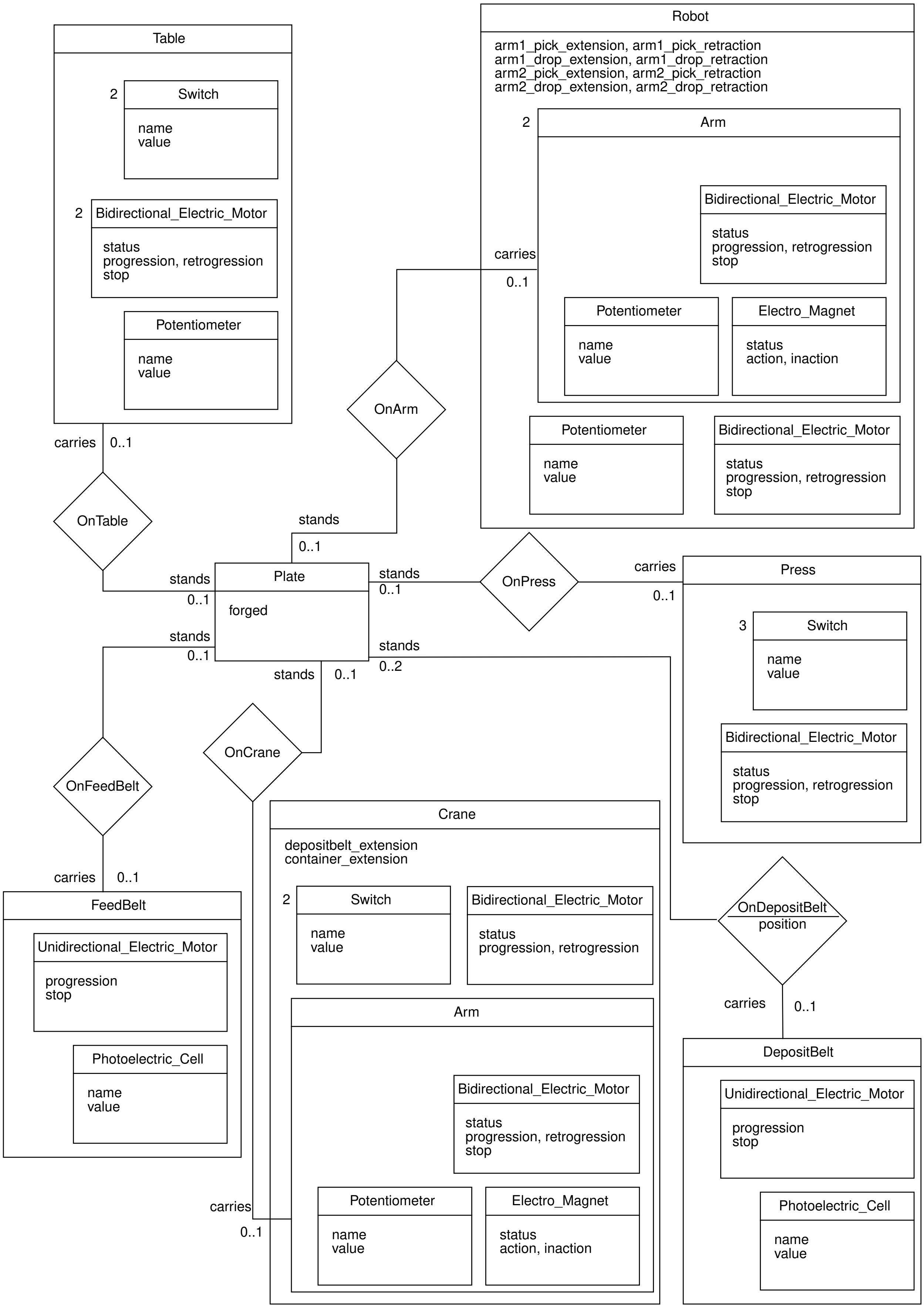

Fig. 15. Object Model (Main)

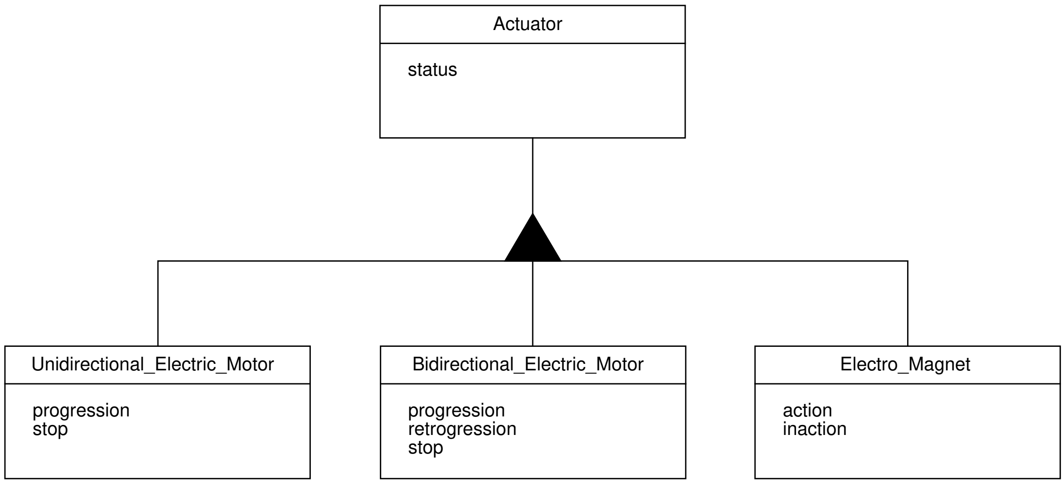

Fig. 17. Object Model (Actuators)

May we recall that the filled triangle indicates a disjoint subtyping relationship, i.e. all objects belong to one and only one class at the leaves of the subtyping hierarchy.

3.3 System Life-Cycle

Below are the life-cycle schemata[1] for the different systems in our controller:

lifecycle FeedBelt: initialize . (add_blank . feed_table . #go_unload_position) *

lifecycle Table: initialize . #feed_table . (go_unload_position . #pick_from_table . go_load_position . #feed_table)*

lifecycle Robot : initialize . EmptyRobot

lifecycle EmptyRobot : (pick_from_table . #go_load_position . Arm1
| pick_from_press . #go_load_position . Arm2)*

lifecycle Arm1 : (load_press . #forge . EmptyRobot
| pick_from_press . #go_load_position . Arm12)*

lifecycle Arm2 : (pick_from_table . #go_load_position . Arm12
| deposit_on_belt . EmptyRobot)*

lifecycle Arm12 : (load_press . #forge . Arm2 | deposit_on_belt . Arm1)*

lifecycle Press: initialize . #load_press . (forge . #pick_from_press . go_load_position . #load_press)*

lifecycle DepositBelt: initialize.#deposit_on_belt.(bring_past_end . #deposit_on_belt . #pick_from_belt)*

lifecycle Crane: initialize . #bring_past_end . (pick_from_belt . #bring_past_end) *

lifecycle Controller: TurnOn . (FeedBelt || Table || Robot || Press || DepositBelt || Crane). TurnOff

TurnOn = *create* . *initialize*

TurnOff = ε

Fig. 18. Life-cycles schemata for the active subsystems in the production cell

EmptyRobot, Arm1, Arm2 and Arm12 correspond respectively to the states of a robot carrying no plate, one plate with the first arm, one plate with the second arm and one plate in each arm.

[1] The regular expressions in the life-cycle models consist of events and the operators of concatenation “.”, alternation “| ”,repetition “*” for zero or more occurrences, “+” for one or more occurrences, interleaving “ ||”, optionality “[ ]”, and grouping “ ( )”. In decreasing order, the precedence is [ ], *, +, . , | , || . Expressions are grouped to override default precedence.

3.4 Operation Models

Below is the list of constants and predicates that will be used to describe the operations.

Table		
Predicate	**Attribute**	**Value**
load position	table.bottom.value	true
	table.rotation.value	0
unload position	table.top.value	true
	table.rotation.value	50
Robot		
Predicate	**Attribute**	**Value**
position 1	robot.rotation.value	0
position 2	robot.rotation.value	50
position 3	robot.rotation.value	35
position 4	robot.rotation.value	between -45 and -90
position 5	robot.rotation.value	-90
Robot Arm Constants		
Predicate	**Attribute**	**Value**
extension to pick a plate from the table	robot.arm1. extension.value	0.5208
retraction after picking a plate from the table	robot.arm1. extension.value	0
extension to drop a plate in the press	robot.arm1. extension.value	0.6458
extension after dropping a plate in the press	robot.arm1. extension.value	0.3708
extension to pick a plate from the press	robot.arm2. extension.value	0.7971
extension after picking a plate from the press	robot.arm2. extension.value	0
extension to drop a plate on the deposit belt	robot.arm2. extension.value	0.5707
extension after dropping the plate on the deposit belt	robot.arm2. extension.value	0
Press Constants		
Predicate	**Attribute**	**Value**
load position	press.middle.value	true
unload position	press.bottom.value	true
Crane Arm Constants		
Predicate	**Attribute**	**Value**
extension to pick a plate from the deposit belt	crane.arm. extension.value	0.9450
extension to drop a plate into the container	crane.arm. extension.value	0.6593

Table 3. Predicates and Constants

3.4.1 Feed Belt Operations

Operation:	initialize
Description:	Initialize the feed belt
Reads:	—
Changes:	
Sends:	

Assumes:	—
Result:	The feed belt is empty (**not has-**carries feedbelt). The motor is off (feedbelt.conveying_motor.status = off). The cell is off (feedbelt.photoelectric_cell.value = false).

Operation:	add_blank
Description:	Add a blank on the feed belt.
Reads:	—
Changes:	p: **new** Plate **with** plate.forged = false
Sends:	—
Assumes:	There is no blank at the end of the feed belt (**not has-**carries feedbelt, the photoelectric cell is off (feedbelt.photoelectric_cell.value = false)).
Result:	A new (unforged) plate has been created. There is a blank at the end of the feed belt (OnFeedBelt **with** (feedbelt, p), The photoelectric cell is on (feedbelt.photoelectric_cell.value = true)).

Operation:	feed_table
Description:	Bring the plate onto the table.
Reads:	—
Changes:	
Sends:	Table: {go_unload_position}
Assumes:	The table is empty (**not has-**carries table). The table is in load position. There is a blank at the end of the belt (OnFeedBelt **with** (feedbelt, p), the photoelectric cell is on (feedbelt.photoelectric_cell.value = true)).
Result:	There is no blank at the end of the feed belt (OnFeedBelt **without** (feedbelt, p), the photoelectric cell is off (feedbelt.photoelectric_cell.value = false))). An event *go_unload_position* has been sent to the table.

3.4.2 Table Operations

Operation:	initialize
Description:	Initialize the table.
Reads:	—
Changes:	
Sends:	Feedbelt: {feed_table}
Assumes:	—
Result:	The table is in load position (see table 3, "Predicates and Constants"). An event *feed_table* has been sent to the feed belt.

Operation:	go_load_position
Description:	Put the table in load position.
Reads:	—

Changes:	
Sends:	Feedbelt: {feed_table}
Assumes:	The table is empty (**not has-**carries table).
Result:	The table is in load position (see table 3, “Predicates and Constants“). An event *feed_table* has been sent to the feed belt.

Operation:	go_unload_position
Description:	Put the table in unload position.
Reads:	—
Changes:	
Sends:	Robot: {pick_from_table}
Assumes:	The table is loaded (**has-**carries table).
Result:	The table is in unload position (see table 3, “Predicates and Constants“). An event *pick_from_table* has been sent to the robot.

3.4.3 Robot Operations

Operation:	initialize
Description:	Initialize the robot.
Reads:	—
Changes:	
Sends:	—
Assumes:	—
Result:	The robot is in its initial position: (i) it is in position 2 (see table 3, “Predicates and Constants“, and figure 8), (ii) both arms are fully retracted (robot.{arm1, arm2}. extension. value = 0, see table 3, “Predicates and Constants“), (iii) both magnets are off.

Operation:	pick_from_table
Description:	Pick the plate from the table.
Reads:	—
Changes:	
Sends:	Table: {go_load_position}
Assumes:	The table is in unload position. The table is loaded (OnTable **with** (table, p)). The first arm is free (**not has-**carries Arm1, the magnet of the first arm is off).
Result:	The first arm carries a plate (OnArm1 **with** (Arm1, p), the magnet of the first arm is on). The first arm is retracted in extension after picking a plate from the press (see table 3, “Predicates and Constants“). The robot is in position 2 (see table 3, “Predicates and Constants“, and figure 8). An event *go_load_position* has been sent to the table.

Operation:	load_press
Description:	Load the plate onto the press
Reads:	—
Changes:	
Sends:	Press: {forge}
Assumes:	The press is empty (**not has-**carries press). The press is in load position (see table 3, "Predicates and Constants"). The first arm carries a plate (OnArm1 **with** (Arm1, p), the magnet of the first arm is on).
Result:	The fist arm is free (OnArm1 **without** (Arm1, p), the magnet of the first arm is off). The first arm is retracted, in extension after dropping a plate in the press (see table 3, "Predicates and Constants"). The robot is in position 5 (see table 3, "Predicates and Constants", and figure 8). An event *forge* has been sent to the press.

Operation:	pick_from_press
Description:	Pick the plate from the press.
Reads:	—
Changes:	
Sends:	Press: {go_load_position}
Assumes:	The press is loaded (OnPress **with** (press, p)). The press is in unload position (see table 3, "Predicates and Constants"). The second arm holds no plate (OnArm2 **without** (Arm2, p), the magnet of the second arm is off).
Result:	The second arm holds a plate (OnArm2 **with** (Arm2, p), the magnet of the second arm is on). The second arm is retracted from the press, in extension after picking a plate from the feedbelt (see table 3, "Predicates and Constants"). The robot is in position 3 (see table 3, "Predicates and Constants", and figure 8). An event *go_load_position* has been sent to the press.

Operation:	deposit_on_belt
Description:	Deposit a plate on the deposit belt
Reads:	—
Changes:	
Sends:	—
Assumes:	The second arm holds a plate (OnArm2 **with** (Arm2, p), the magnet of the second arm is on). There is no plate at the beginning of the deposit belt (**not has-**carries depositbelt **with** position = begin).
Result:	The second arm holds no plate (OnArm2 **without** (Arm2, p), the magnet of the second arm is off). The robot is in position 4 (see table 3, "Predicates and Constants", and figure 8). The second arm is retracted, in extension after dropping the plate on the deposit belt (see table 3, "Predicates and Constants").

3.4.4 Press Operations

Operation:	initialize
Description:	Initialize the Press
Reads:	—
Changes:	
Sends:	Robot: {load_press}
Assumes:	—
Result:	The press is in load position (see table 3, “Predicates and Constants“). An event *load_press* has been sent to the robot.

Operation:	go_load_position
Description:	Put the press in load position.
Reads:	—
Changes:	
Sends:	Robot: {load_press}
Assumes:	The press is empty (**not has-**carries press).
Result:	The press is in load position (see table 3, “Predicates and Constants“). An event *load_press* has been sent to the robot.

Operation:	forge
Description:	Forge the plate
Reads:	—
Changes:	p: Plate
Sends:	Robot: {pick_from_press}
Assumes:	There is a plate in the press (OnPress **with** (press, p)).
Result:	The press is in unload position (see table 3, “Predicates and Constants“). The plate is forged (p.forged = true). An event *pick_from_press* has been sent to the robot.

3.4.5 Deposit Belt Operations

Operation:	initialize
Description:	Initialize the DepositBelt.
Reads:	—
Changes:	
Sends:	Robot: {deposit_on_belt}

Assumes:	—
Result:	The motor of the deposit belt is off (depositbelt.conveying_motor.status = off). One event *deposit_on_belt* has been sent to the robot.

Operation:	bring_past_end
Description:	Bring the plate to the end of the deposit belt
Reads:	—
Changes:	
Sends:	Robot: {deposit_on_belt} Crane: {pick_from_belt}
Assumes:	—
Result:	A plate is at the end of the deposit belt (**has-**carries depositbelt **with** position = end). The motor of the deposit belt is off (depositbelt.conveying_motor.status = off). An event *deposit_on_belt* has been sent to the robot. An event *pick_from_belt* has been sent to the crane.

3.4.6 Crane Operations

Operation:	initialize
Description:	Initialize the Crane.
Reads:	—
Changes:	
Sends:	DepositBelt: {bring_past_end}
Assumes:	—
Result:	The crane is over the deposit belt, its arm is extended, it does not carry anything (**not has**-carries crane, the magnet of the arm is off). An event *bring_past_end* has been sent to the deposit_belt.

Operation:	pick_from_belt
Description:	Pick a plate and bring it from the deposit belt to the container. Drop it into the container and go back waiting over the deposit belt
Reads:	—
Changes:	
Sends:	DepositBelt: {bring_past_end}
Assumes:	A plate is at the end of the deposit belt (OnDepositBelt **with** (depositbelt, p) **with** position = end). The crane is over the deposit belt (switch over the deposit belt is on).
Result:	An event *bring_past_end* has been sent to the deposit_belt.

4. Design

4.1 Design Objects

Whereas the models of the analysis were not implementation-driven, the models in the design reflect our choice to use the FZI simulator.

Since the status is not directly read from the device (contrary to what is shown on figure 10), but must be read from the terminal, we add a class Sensor_Server, which will have the responsibility to manage the terminal. (Figure 19). Thus, any sensor will rely on a Sensor_Server to get their values from the simulator.

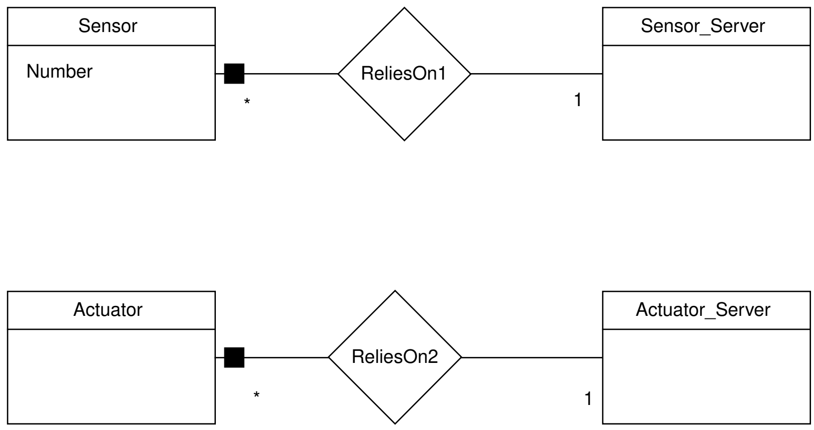

Fig. 19. Design Object Model

Similarly, we add an object Actuator_Server, to which each actuator must be connected. Thus the actuators will rely on an actuator server to make sure events are sent to the simulator in a consistent way.

4.2 Interaction Graphs

4.2.1 Feed Belt Operations

- **Operation initialize**

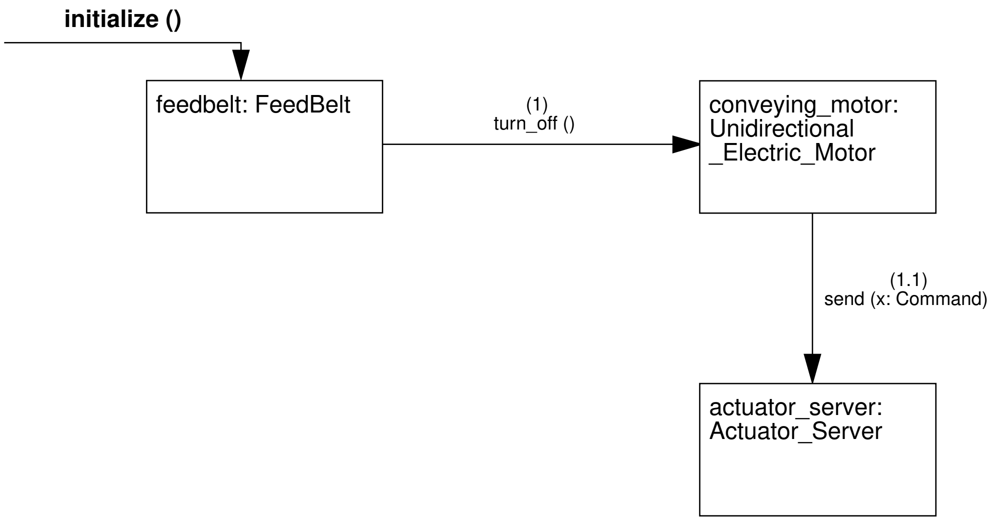

Fig. 20. Interaction graph for operation initialize

Description

Operation FeedBelt: initialize ()
turn off the motor (1)

Method Unidirectional_Electric_Motor: turn_off ()
ask the actuator_server to turn off the motor, using the command in the attribute stop (1.1)

Method Actuator_Server: send (x: Command)
send the command x to the production cell (possibly to the simulator)

- **Operation add_blank**

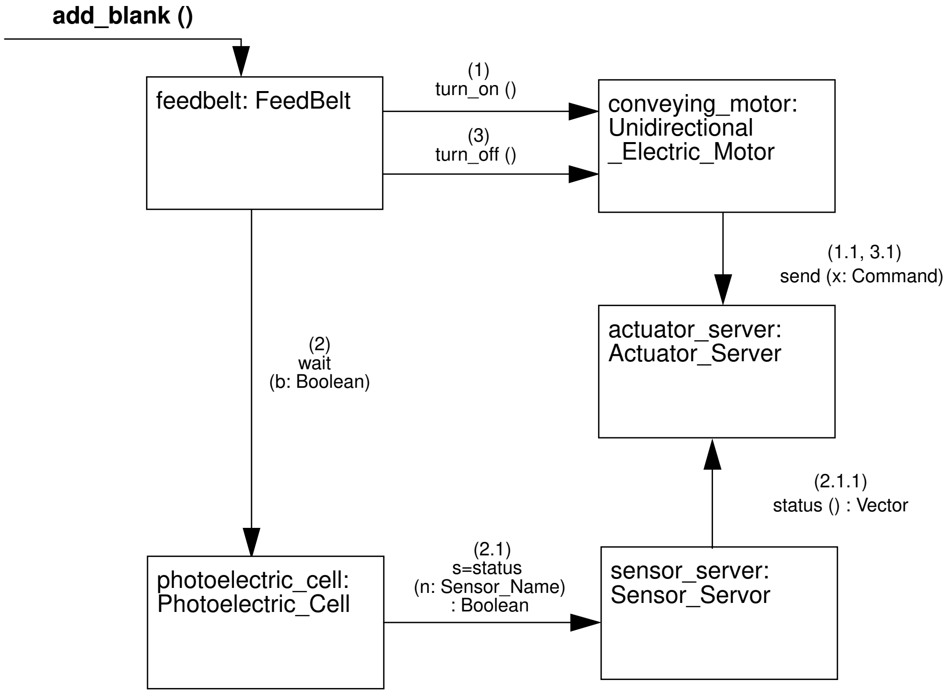

Fig. 21. Interaction graph for operation add_blank

Description

Operation FeedBelt: add_blank ()
turn on the motor (1)
wait until the photoelectric cell indicates a plate at the end of the belt (2)
turn off the motor (3)

Method Unidirectional_Electric_Motor: turn_on ()
ask the actuator_server to turn on the motor, using the command in the attribute progression (1.1)

Method Photoelectric_Cell: wait (b: Boolean)
wait until the boolean value b is reached by examining the sensor name (2.1)

Method Unidirectional_Electric_Motor: turn_off ()
(see operation FeedBelt: initialize ())

Method Sensor_Server: status (n: Sensor_Name): Boolean
returns the value of the Sensor_Name by sending the command get_status to the actuator_server and observing the resulting vector in position n (2.1.1)

Method Actuator_Server: status (): Vector
returns a vector containing the values of the sensors

Method Actuator_Server: send (x: Command)
(see operation FeedBelt: initialize ())

- **Operation feed_table**

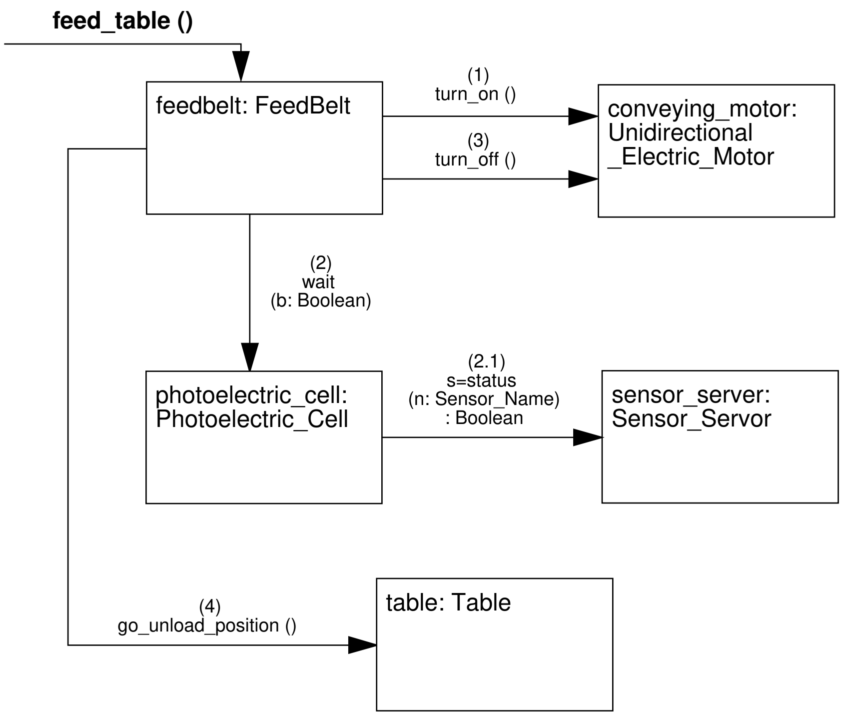

Fig. 22. Interaction graph for operation feed_table

Description

Operation FeedBelt: feed_table ()
turn on the motor (1)
wait until the photoelectric cell indicates that there is no plate at the end of the belt (2)
turn off the motor (3)
tell the table to go in the unload position (4)

Method Unidirectional_Electric_Motor: turn_on ()
(see operation FeedBelt: add_blank ())

Method Photoelectric_Cell: wait (b: Boolean)
(see operation FeedBelt: add_blank ())

Method Unidirectional_Electric_Motor: turn_off ()
(see operation FeedBelt: initialize ())

Method Sensor_Servor: status (n: Sensor_Name): Boolean
(see operation FeedBelt: add_blank ())

4.2.2 Table Operations

- **Operation initialize**

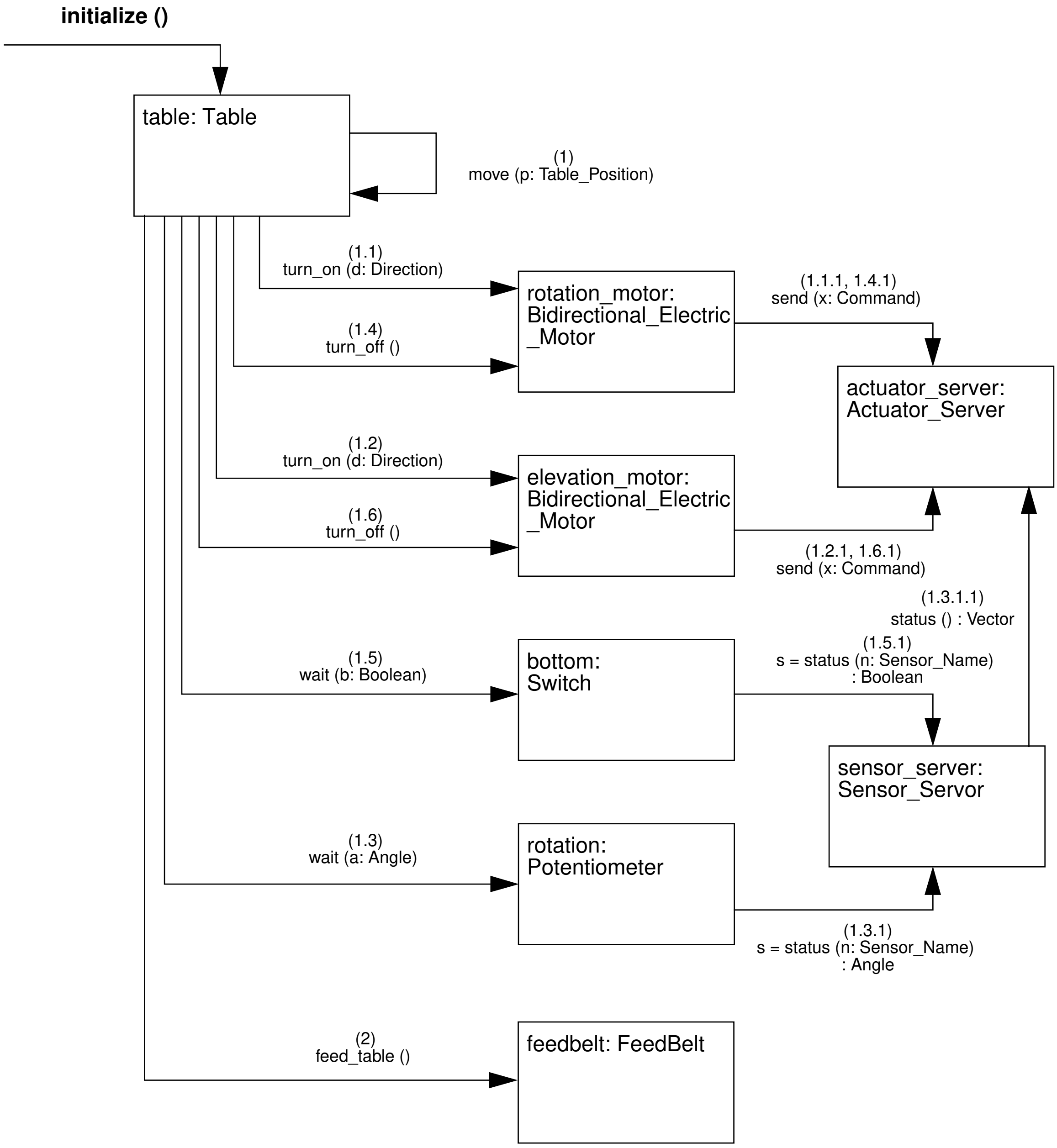

Fig. 23. Interaction graph for operation initialize

Description

Operation Table: initialize ()
move the table in load position (1)
tell the feed belt to feed the table (2)

Method Table: move (p: Table_Position)
if p is loadposition **then**
  simultaneously
    turn on the rotation motor in regressive direction (1.1)
    wait until the potentiometer rotation indicates that the load position is reached (1.3)
    turn off the rotation motor (1.4)
  and
    turn on the elevation motor in regressive direction (1.2)
    wait until the switch bottom indicates that the bottom position is reached (1.5)

turn off the elevation motor (1.6)
else p is unload position, see operation go_unload_position ()

Method Potentiometer: wait (a: Angle)
wait until the angle value a is reached by exaniming the sensor name (1.3.1)

Method Switch: wait (b: Boolean)
wait until the boolean value b is reached by exaniming the sensor name (1.5.1)

Method Bidirectional_Electric_Motor: turn_on (d: Direction)
if the direction is progressive **then**
ask the actuator_server to turn on the motor in progressive direction, using the command in the attribute progression (1.1.1, 1.2.1)
else
ask the actuator_server to turn on the motor in regressive direction, using the command in the attribute retrogression (1.1.1, 1.2.1)

Method Bidirectional_Electric_Motor: turn_off ()
ask the actuator_server to turn off the motor, using the command in the attribute stop (1.4.1, 1.6.1)

Method Sensor_Server: status (n: Sensor_Name): Angle
returns the value of the Sensor_Name by sending the command get_status to the actuator_server and observing the resulting vector in position n (1.3.1.1)

Method Sensor_Server: status (n: Sensor_Name): Boolean
(see operation FeedBelt: add_blank ())

Method Actuator_Server: send (x: Command)
(see operation FeedBelt: initialize ())

- **Operation go_load_position**

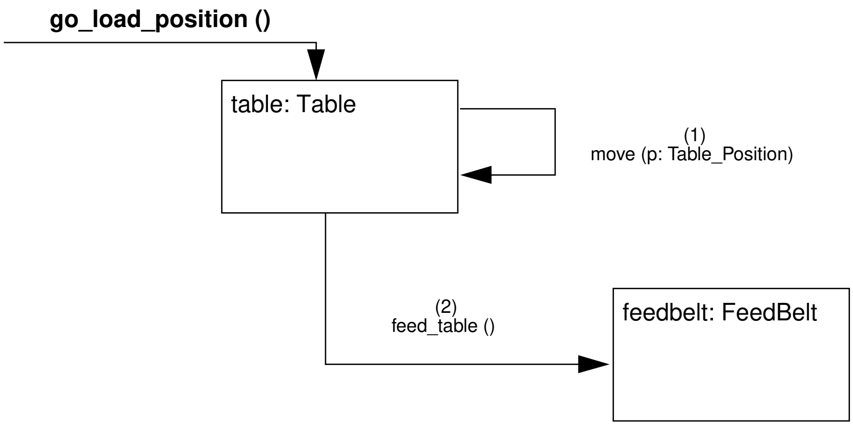

Fig. 24. Interaction graph for operation go_load_position

Description

Operation Table: go_load_position ()
move the table in load position (1)
tell the feed belt to feed the table (2)

Method Table: move (p: Table_Position)
(see operation Table: initialize ())

- **Operation go_unload_position**

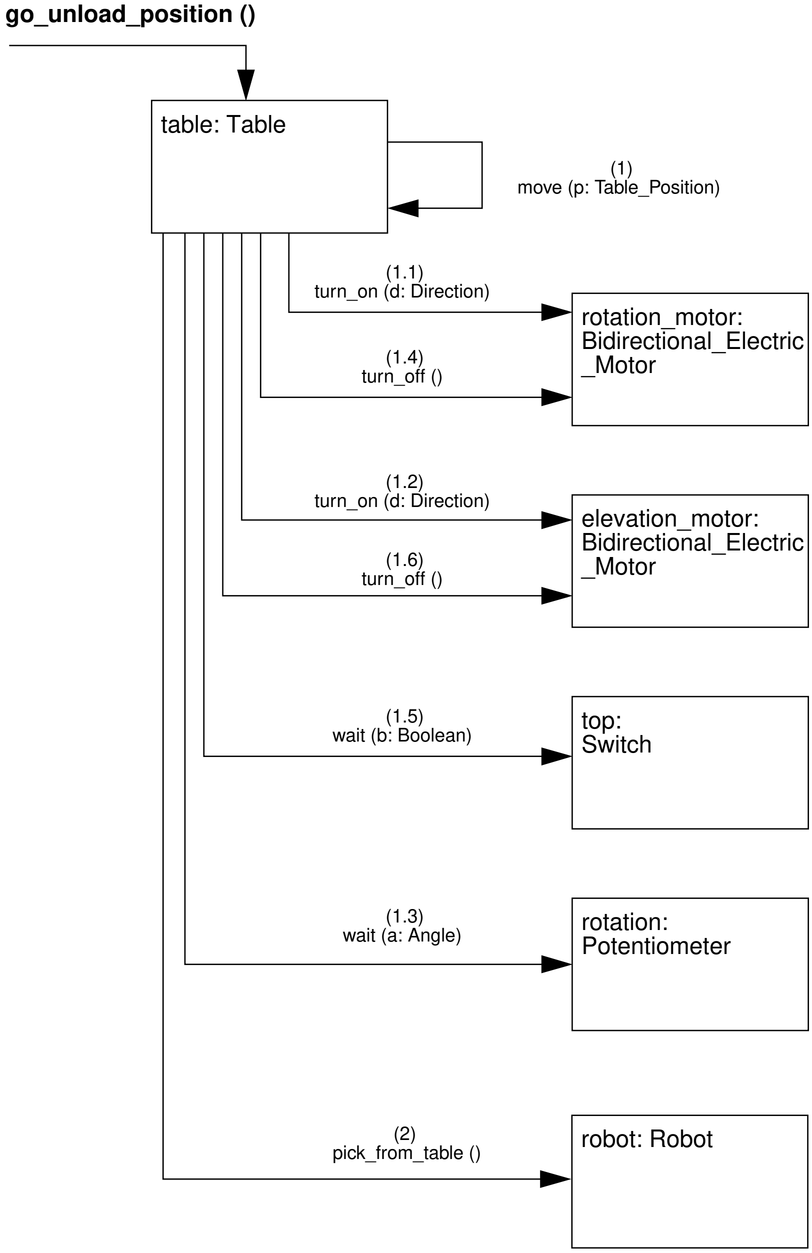

Fig. 25. Interaction graph for operation go_unload_position

Description

Operation Table: go_unload_position ()
move the table in unload position (1)
tell the robot to pick the plate (2)

Method Table: move (p: Table_Position)
if p is loadposition **then**
  see operation Table: initialize ()
else p is unloadposition
  simultaneously
    turn on the rotation motor in progressive direction (1.1)
    wait until the potentiometer rotation indicates that the unload position is reached (1.3)
    turn off the rotation motor (1.4)
  and
    turn on the elevation motor in progressive direction (1.2)
    wait until the switch top indicates that the top position is reached (1.5)

turn off the elevation motor (1.6)

Method Potentiometer: wait (a: Angle)
(see operation Table: initialize ())

Method Switch: wait (b: Boolean)
(see operation Table: initialize ())

Method Bidirectional_Electric_Motor: turn_on (d: Direction)
(see operation Table: initialize ())

Method Bidirectional_Electric_Motor: turn_off ()
(see operation Table: initialize ())

4.2.3 Robot Operations

- **Operation initialize**

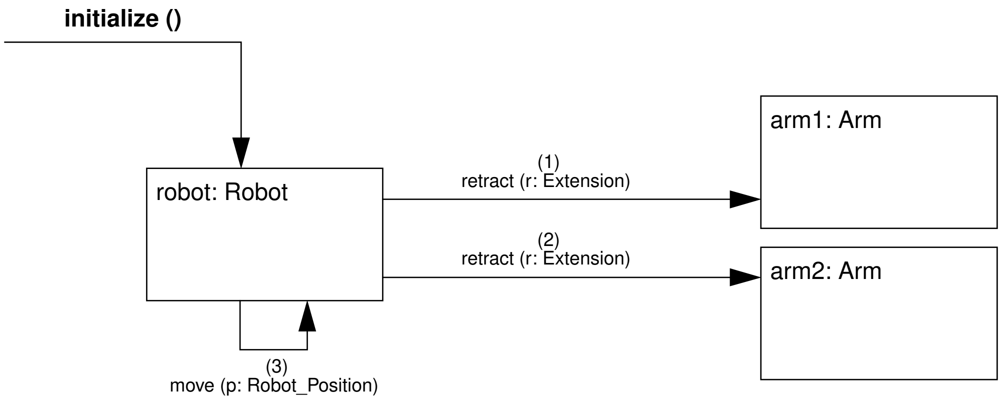

Fig. 26. Interaction graph for operation initialize

Description

Operation Robot: initialize ()
simultaneously
retract first arm (1)
and
retract second arm (2)
move the robot in robot position 2 (3)

Method Robot: move (p: Robot_Position)
(see operation Robot: pick_from_table ())

Method Arm: retract (r: Extension)
(see operation Robot: pick_from_table ())

- **Operation pick_from_table**

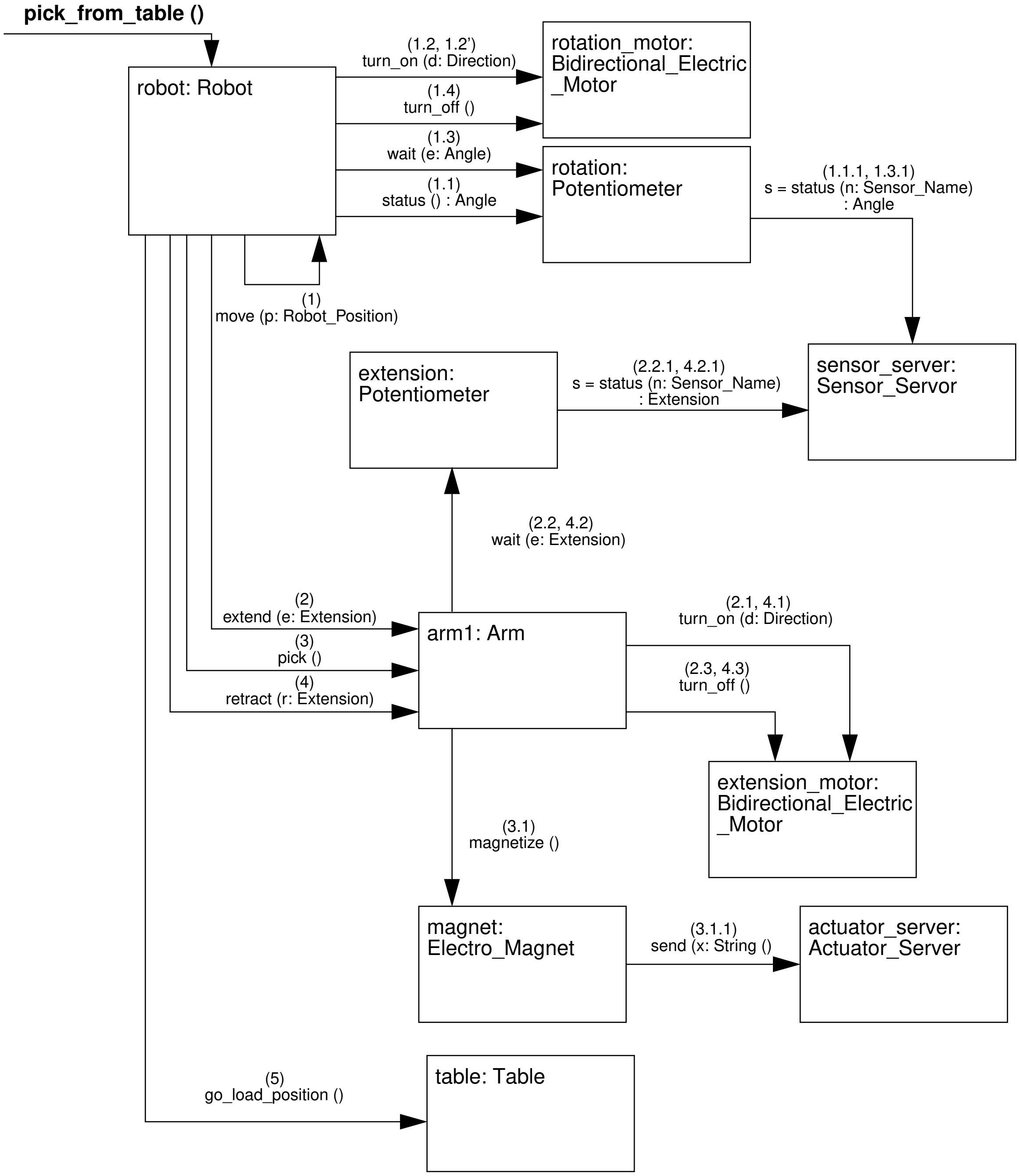

Fig. 27. Interaction graph for operation pick_from_table

Description

Operation Robot: pick_from_table ()
move the robot so that the first arm is in front of the table, i.e. in robot position 2 (1)
extend the first arm over the table, in extension given in the attribute arm1_pick_extension (2)
pick the plate (3)
retract the first arm from the table, in extension given in the attribute arm1_pick_retraction (4)
tell the table to go in load position (5)

Method Robot: move (p: Robot_Position)
if the current angle is greater than the angle to reach **then** (1.1)
    turn on the rotation motor in regressive direction (1.2)
else
    turn on the rotation motor in progressive direction (1.2')

wait until the potentiometer robot_angle indicates that the position p is reached (1.3)
turn off the motor (1.4)

Method Arm: extend (e: Extension)
turn on the extension motor in progressive direction (2.1)
wait until the potentiometer extension indicates that the extension e is reached (2.2)
turn off the extension motor (2.3)

Method Arm: pick ()
magnetize the magnet (3.1)

Method Arm: retract (r: Extension)
turn on the extension motor in regressive direction (4.1)
wait until the potentiometer extension indicates that the extension r is reached (4.2)
turn off the extension motor (4.3)

Method Electro_Magnet: magnetize ()
ask the actuator_server to turn on the magnet, using the command in the attribute action (2.1.1)

Method Potentiometer: wait (e: Extension)
wait until the extension value e is reached by examining the sensor name (2.2.1, 4.2.1)

Method Potentiometer: wait (a: Angle)
wait until the angle value a is reached by examining the sensor name (1.3.1)

Method Potentiometer: status (): Angle
return the value of the sensor name (1.1.1)

Method Bidirectional_Electric_Motor: turn_on (d: Direction)
(see operation Table: initialize ())

Method Bidirectional_Electric_Motor: turn_off ()
(see operation Table: initialize ())

Method Sensor_Server: status (n: Sensor_Name): Angle
(see operation Table: initialize ())

Method Actuator_Server: send (x: Command)
(see operation FeedBelt: initialize ())

- **Operation load_press**

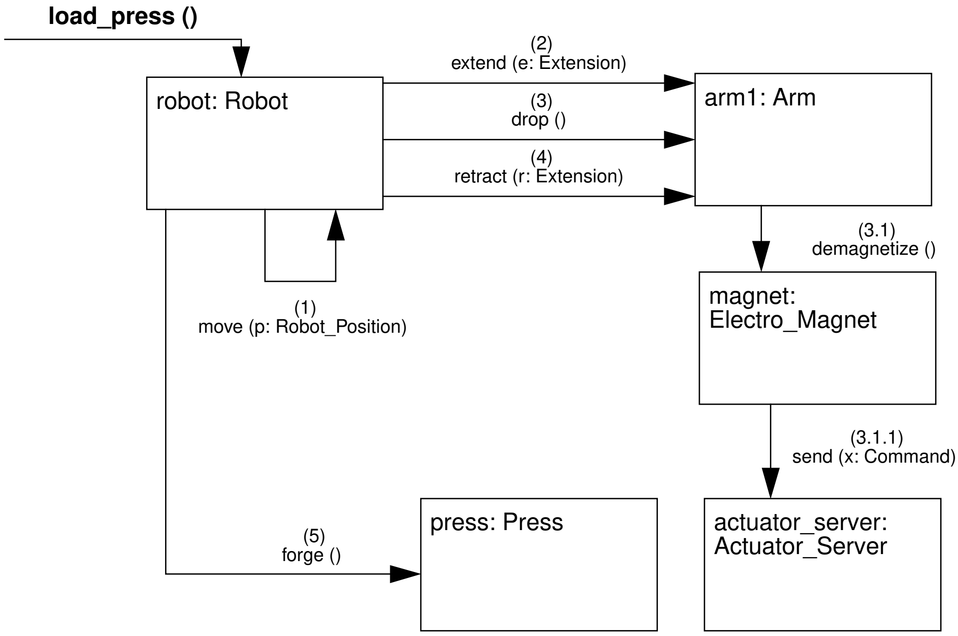

Fig. 28. Interaction graph for operation load_press

Description

Operation Robot: load_press ()
move the robot so that the first arm is in front of the press (in robot position 5) (1)
extend the arm over the press, in extension given in the attribute arm1_drop_extension (2)
drop the plate on the press (3)
retract the arm from the press, in extension given in the attribute arm1_drop_retraction (4)
tell the press to forge the plate (5)

Method Arm: drop ()
demagnetize the magnet (2.1)

Method Electro_Magnet: demagnetize ()
ask the actuator_server to turn off the magnet, using the command in the attribute inaction (3.1.1)

Method Robot: move (p: Robot_Position)
(see operation Robot: pick_from_table ())

Method Arm: extend (e: Extension)
(see operation Robot: pick_from_table ())

Method Arm: retract (r: Extension)
(see operation Robot: pick_from_table ())

Method Actuator_Server: send (x: Command)
(see operation FeedBelt: initialize ())

- **Operation pick_from_press**

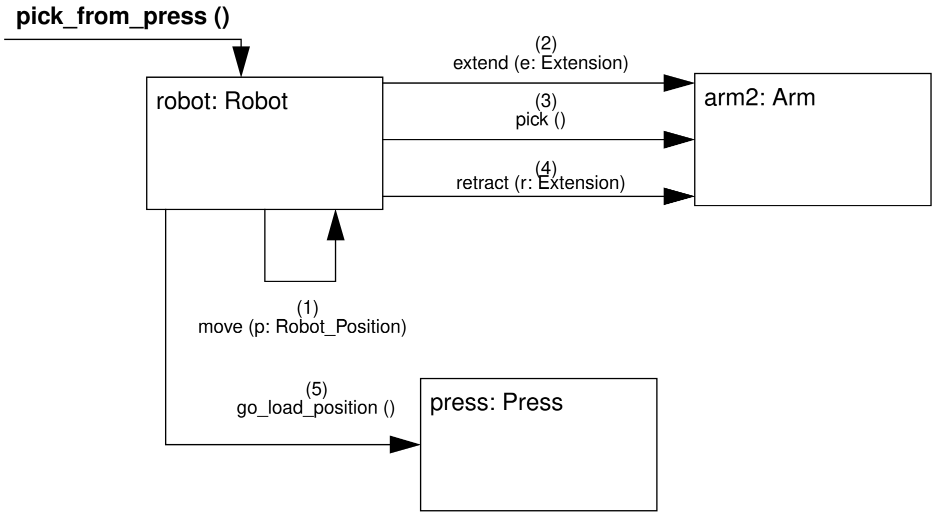

Fig. 29. Interaction graph for operation pick_from_press

Description

Operation Robot: pick_from_press ()
move the robot so that the second arm is in front of the press (in robot position 3) (1)
extend the second arm over the press, in extension given in the attribute arm2_pick_extension (2)
pick the plate from the press (3)
retract the arm from the press, in extension given in the attribute arm2_pick_retraction (4)
tell the press to go in load position (5)

Method Robot: move (p: Robot_Position)
(see operation Robot: pick_from_table ())

Method Arm: extend (e: Extension)
(see operation Robot: pick_from_table ())

Method Arm: pick ()
(see operation Robot: pick_from_table ())

Method Arm: retract (r: Extension)
(see operation Robot: pick_from_table ())

- **Operation deposit_on_belt**

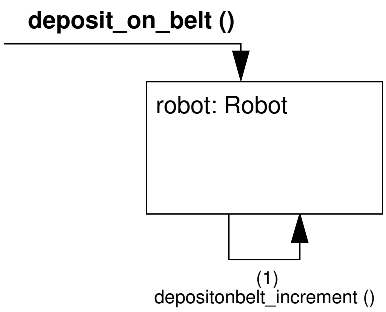

Fig. 30. Interaction graph for operation deposit_on_belt

Description

Operation Robot: deposit_on_belt ()
increment by 1 the number of blanks the robot can drop on the deposit belt (1)

Method Robot: depositonbelt_increment ()
increment by 1 depositonbelt_counter

- **Method deposit_on_belt_int**

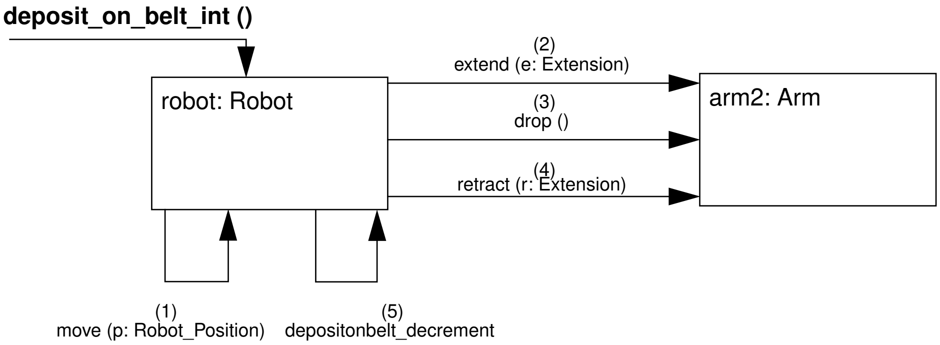

Fig. 31. Interaction graph for operation deposit_on_belt

Description

Method Robot: deposit_on_belt ()
if depositonbelt_counter > 0 **then**
move the robot so that the second arm is in front of the deposit belt (in robot position 4)(1)
extend the second arm over the deposit belt, in extension given by arm2_drop_extension (2)
drop the plate on the deposit belt (3)
retract the arm from the deposit belt, in extension given in the attribute arm2_drop_retraction (4)
decrement by 1 the number of blanks the robot can drop on the deposit belt (5)

Method Robot: move (p: Robot_Position)
(see operation Robot: pick_from_table ())

Method Arm: extend (e: Extension)
(see operation Robot: pick_from_table ())

Method Arm: drop ()
(see operation Robot: load_press ())

Method Arm: retract (r: Extension)
(see operation Robot: pick_from_table ())

Method Robot: depositonbelt_decrement ()
decrement by 1 depositonbelt_counter

4.2.4 Press Operations

- **Operation initialize**

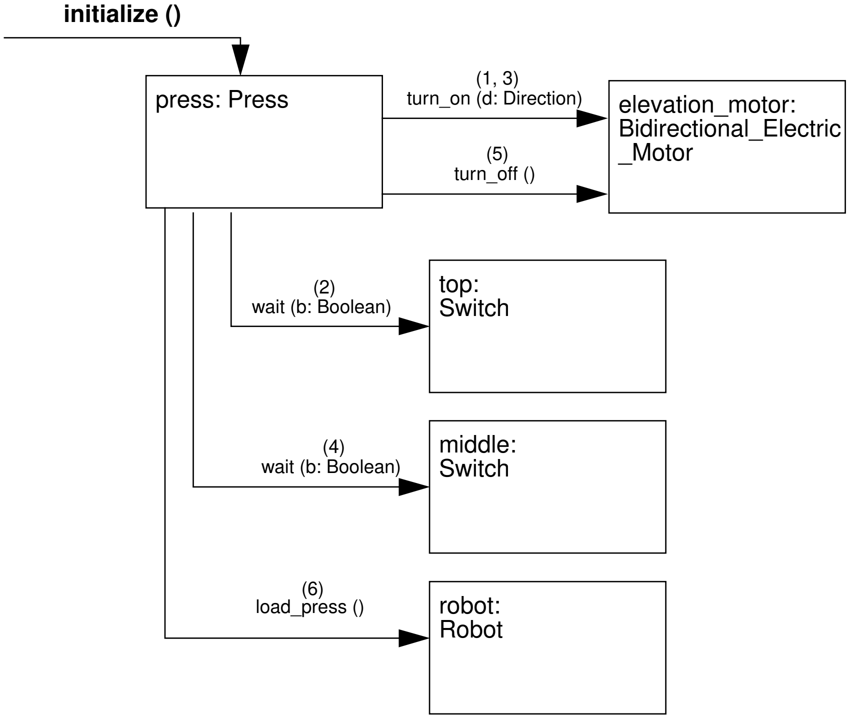

Fig. 32. Interaction graph for operation initialize

Description

Operation Press: initialize ()
turn on the elevation motor in progressive direction (1)
wait until the switch top indicates that the top position is reached (2)
turn on the elevation motor in regressive direction (3)
wait until the switch middle indicates that the middle position is reached (4)
turn off the elevation motor (5)
tell the Robot to load the press (6)

Method Bidirectional_Electric_Motor: turn_on (d: Direction)
(see operation Table: initialize ())

Method Bidirectional_Electric_Motor: turn_off ()
(see operation Table: initialize ())

Method Switch: wait (b: Boolean)
(see operation Table: initialize ())

- **Operation go_load_position**

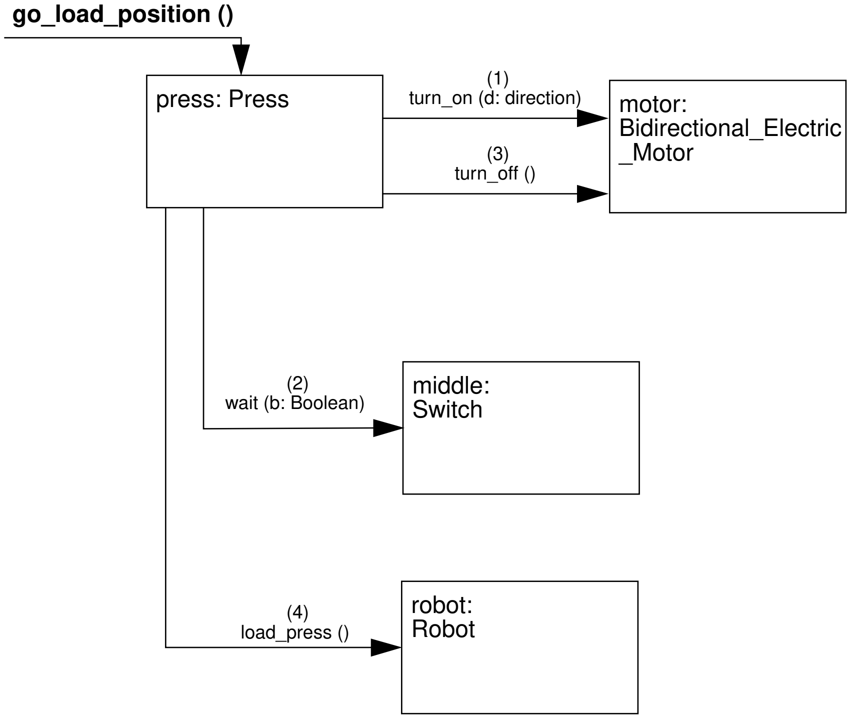

Fig. 33. Interaction graph for operation go_load_position

Description

Operation Press: go_load_position ()
turn on the motor in progressive direction (1)
wait until the switch middle indicates that the middle position is reached (2)
turn off the motor (3)
tell the robot to load the press (4)

Method Bidirectional_Electric_Motor: turn_on (d: Direction)
(see operation Table: initialize ())

Method Bidirectional_Electric_Motor: turn_off ()
(see operation Table: initialize ())

Method Switch: wait (b: Boolean)
(see operation Table: initialize ())

- **Operation forge**

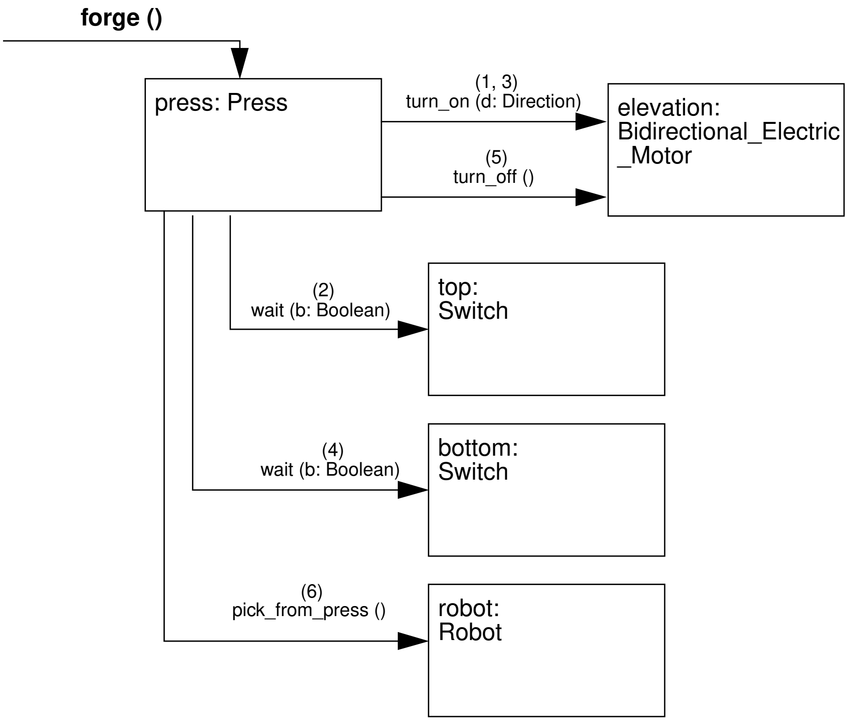

Fig. 34. Interaction graph for operation forge

Description

Operation Press: forge ()
turn on the elevation motor in progressive direction (1)
wait until the switch top indicates that the top position is reached (2)
turn on the elevation motor in regressive direction (3)
wait until the switch bottom indicates that the bottom position is reached (4)
turn off the elevation motor (5)
tell the Robot to pick the plate from the press (6)

Method Bidirectional_Electric_Motor: turn_on (d: Direction)
(see operationTable: initialize ())

Method Bidirectional_Electric_Motor: turn_off ()
(see operation Table: initialize ())

Method Switch: wait (b: Boolean)
(see operation Table: initialize ())

4.2.5 Deposit Belt Operations

- **Operation initialize**

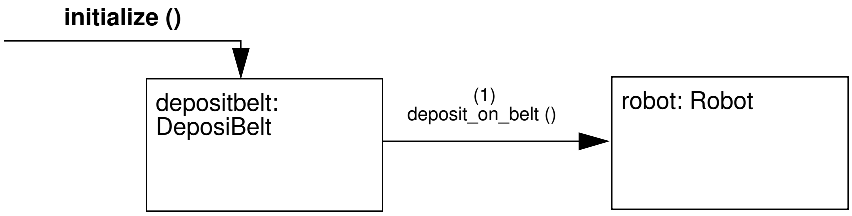

Fig. 35. Interaction graph for operation initialize

Description

Operation DepositBelt: initialize ()
tell the robot to drop a plate on the deposit belt (1)

- **Operation bring_past_end**

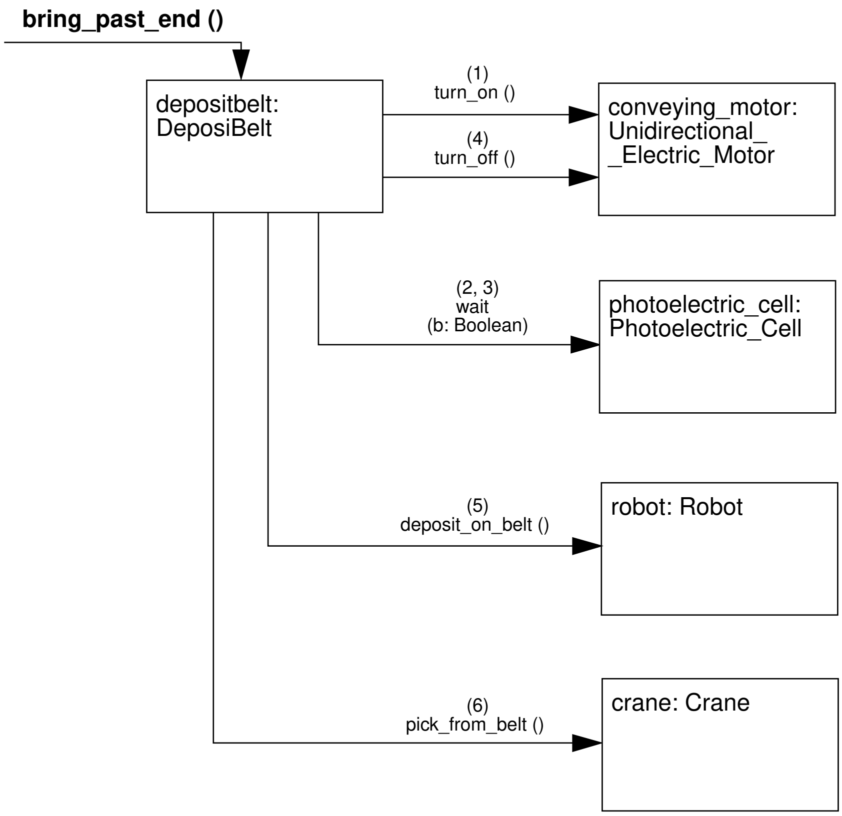

Fig. 36. Interaction graph for operation bring_past_end

Description

Operation DepositBelt: bring_past_end ()
turn on the motor (1)
wait until the photoelectric cell indicates a plate in the photoelectric barrier of the belt (2)
wait until the photoelectric cell indicates no plate in the photoelectric barrier of the belt (3)
turn off the motor (4)
tell the robot to drop a plate on the deposit belt (5)
tell the crane to pick a plate from the belt (6)

Method Unidirectional_Electric_Motor: turn_on ()
(see operation FeedBelt: add_blank ())

Method Photoelectric_Cell: wait (b: Boolean)
(see operation FeedBelt: add_blank ())

Method Unidirectional_Electric_Motor: turn_off ()
(see operation FeedBelt: initialize ())

4.2.6 Crane Operations

- **Operation initialize**

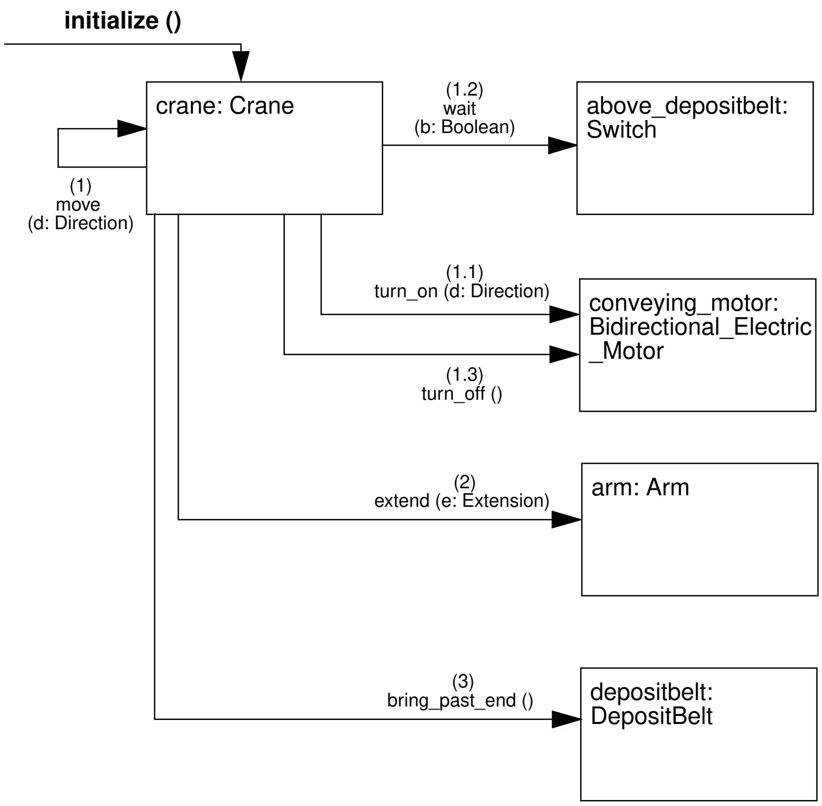

Fig. 37. Interaction graph for operation initialize

Description

Operation Crane: initialize ()
simultaneously
move horizontally the crane towards the deposit belt (1)
and
extend vertically the arm towards the deposit belt, in extension given in the attribute depositbelt_extension (2)
tell the deposit belt to bring a plate to the end of the belt (3)

Method Crane: move (d: Direction)
turn on the motor of the rail in the direction d (1.1)
if the direction is progressive **then**
(see operation Crane: pick_from_belt ())
else
wait until the switch indicates the crane above the deposit belt (1.2)
turn off the motor of the rail (1.3)

Method Arm: extend (e: Extension)
(see operation Robot: pick_from_table ())

Method Bidirectional_Electric_Motor: turn_on (d: Direction)
(see operation Table: initialize ())

Method Bidirectional_Electric_Motor: turn_off ()
(see operation Table: initialize ())

Method Switch: wait (b: Boolean)
(see operation Table: initialize ())

- **Operation pick_from_belt**

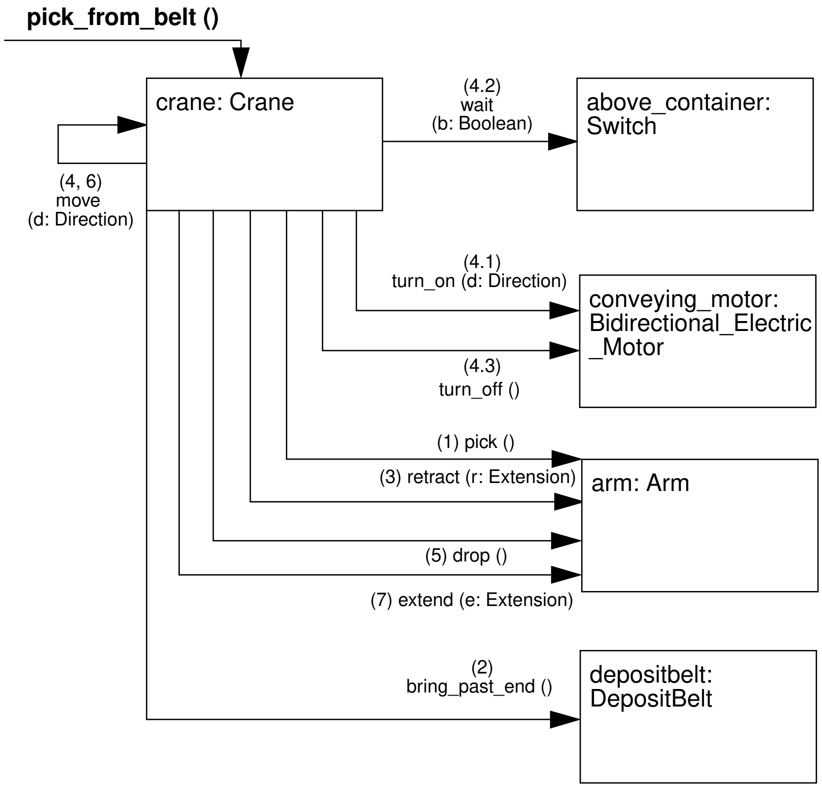

Fig. 38. Interaction graph for operation pick_from_belt

Description

Operation Crane: pick_from_belt ()
pick the plate (1)
tell the deposit belt to bring a plate to the end of the belt (2)
retract vertically the arm up to the level of the container, in extension given in the attribute container_extension (3)
move horizontally the crane towards the container (4)
drop the plate (5)
simultaneously
move horizontally the crane towards the deposit belt (6)
and
extend vertically the arm towards the deposit belt, in extension given in the attribute depositbelt_extension (7)

Method Arm: pick ()
(see operation Robot: pick_from_table ())

Method Arm: retract (r: Extension)
(see operation Robot: pick_from_table ())

Method Crane: move (d: Direction)
turn on the motor of the rail in the direction d (4.1)

if the direction is progressive **then**
wait until the switch indicates the crane above the container (4.2)
else
(see operation Crane: initialize ())
turn off the motor of the rail (4.3)

Method Bidirectional_Electric_Motor: turn_on (d: Direction)
(see operation Table: initialize ())

Method Bidirectional_Electric_Motor: turn_off ()
(see operation Table: initialize ())

Method Switch: wait (b: Boolean)
(see operation Table: initialize ())

Method Arm: drop ()
(see operation Robot: load_press ())

Method Arm: extend (e: Extension)
(see operation Robot: pick_from_table ())

4.2.7 Interaction Graphs for Creation Methods

- **FeedBelt**

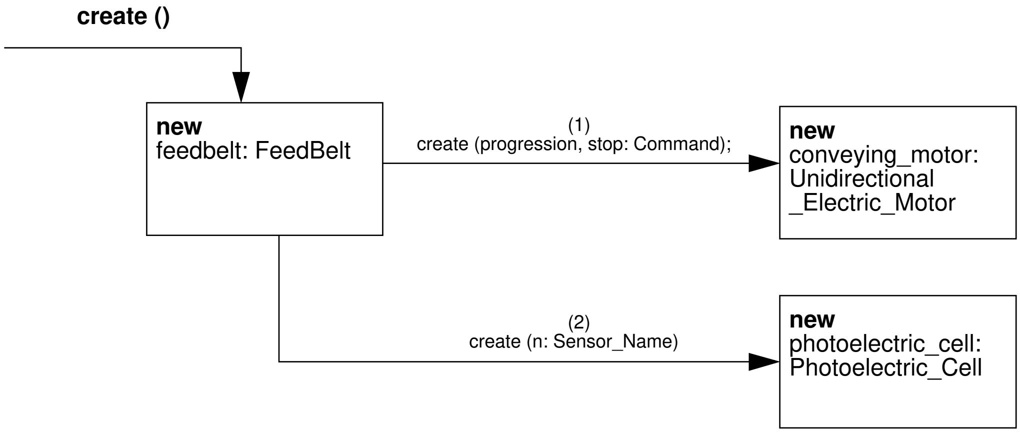

Fig. 39. Interaction graph for method FeedBelt: create ()

Description

Method FeedBelt: create ()
create the unidirectional electric motor conveying_motor with parameters:
(belt1_start, belt1_stop) (1)
create the photoelectric cell photoelectric_cell with parameter:
(belt1_blank_at_end) (2)

Method Unidirectional_Electric_Motor: create (progression, stop: Command)
assign the parameters to the attributes of the same name.

Method Photoelectric_Cell: create (n: Sensor_Name)
assign the value n to the attribute name.

- **Table**

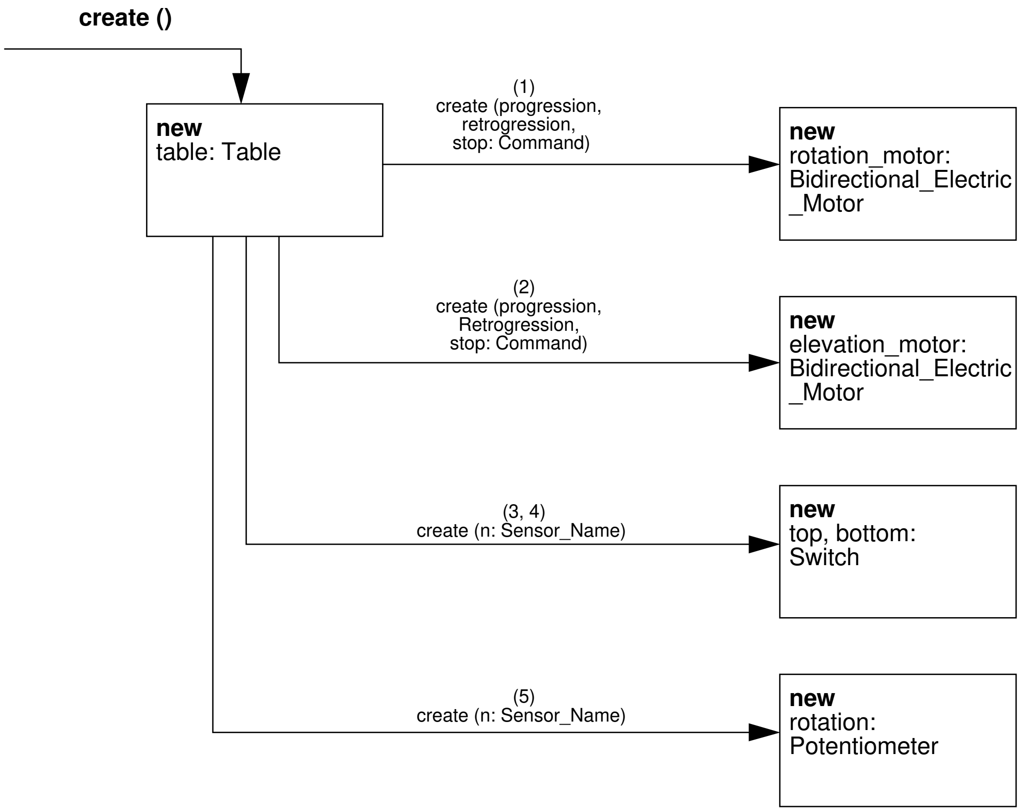

Fig. 40. Interaction graph for method Table: create ()

Description

Method Table: create ()
create the bidirectional electric motor rotation_motor with parameters: (table_left, table_right, table_stop_h) (1)
create the bidirectional electric motor elevation_motor with parameters: (table_upward, table_downward, table_stop_v) (2)
create the switch top with parameter: (table_top) (3)
create the switch bottom with parameter: (table_bottom) (4)
create the potentiometer rotation with parameter: (table_angle) (5)

Method Bidirectional_Electric_Motor: create (progression, retrogression, stop: Command)
assign the parameters to the attributes of the same name.

Method Switch: create (n: Sensor_Name)
assign the value n to the attribute name

Method Potentiometer: create (n: Sensor_Name)
assign the value n to the attribute name

- **Robot**

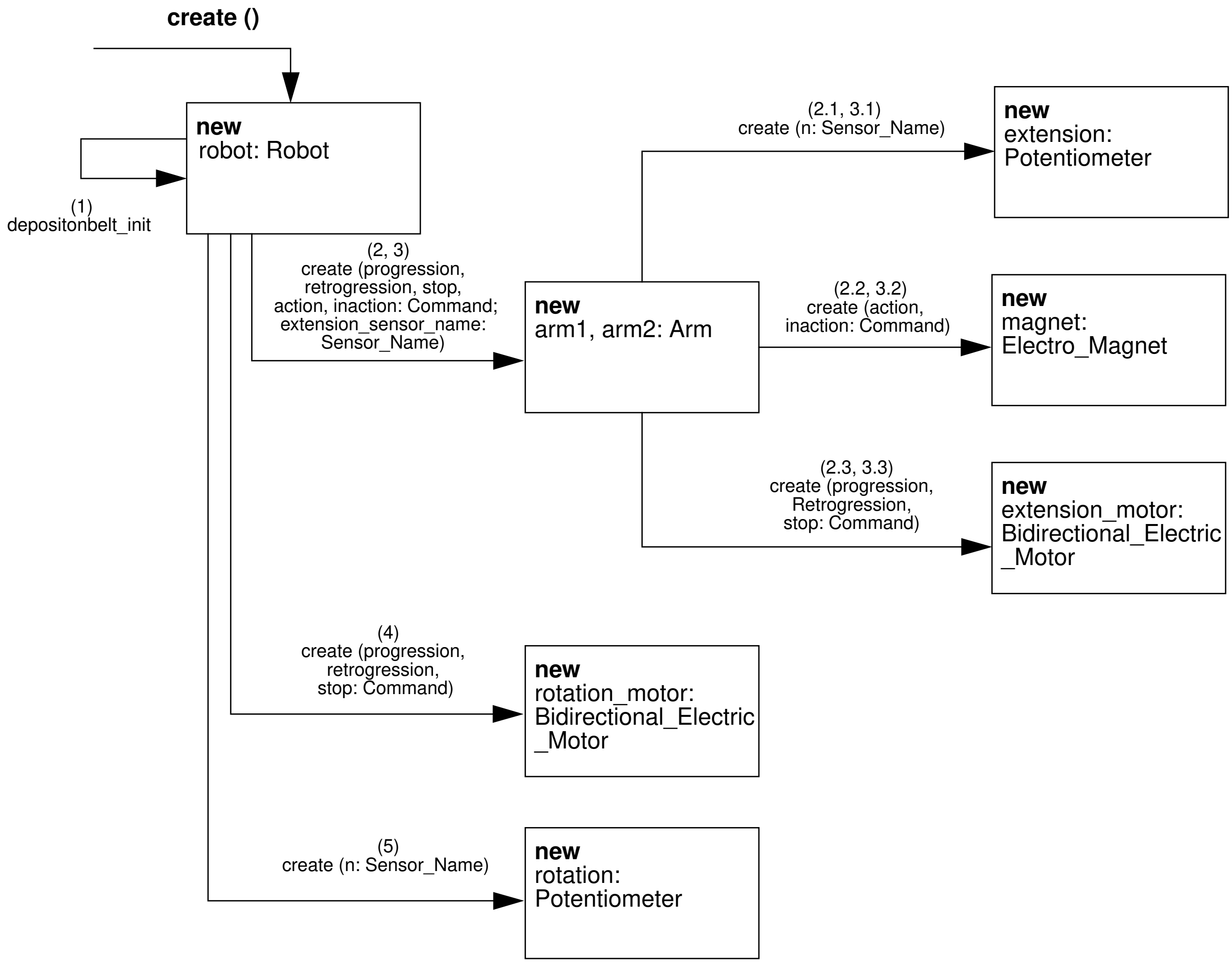

Fig. 41. Interaction graph for method Robot: create ()

Description

Method Robot: create ()
establish that the robot can not drop a blank on the deposit belt (1)
create the first arm (arm1) with parameters: (arm1_forward, arm1_backward, arm1_stop, arm1_mag_on, arm1_mag_off, arm1_extension) (2)
create the second arm (arm2) with parameters: (arm2_forward, arm2_backward, arm2_stop, arm2_mag_on, arm2_mag_off, arm2_extension) (3)
create the bidirectional electric motor rotation_motor with parameters: (robot_left, robot_right, robot_stop) (4)
create the potentiomenter rotation with parameter: (robot_angle) (5)

Method Robot: depositonbelt_init ()
assign 0 to depositonbelt_counter

Method Arm: create (progression, retrogression, stop, action, inaction: Command; extension_sensor_name: Sensor_Name)
create the potentiometer extension with parameters: (extension_sensor_name) (2.1, 3.1)
create the electromagnet magnet with parameter: (action, inaction) (2.2, 3.2)
create the bidirectional electric motor extension_motor with parameter: (progression, retrogression, stop) (2.3, 3.3)

Method Potentiometer: create (n: Sensor_Name)
(see method Table: create ())

Method Electro_Magnet: create (action, inaction: Command)
assign the parameters to the attributes of the same name.

Method Bidirectional_Electric_Motor: create (progression, retrogression, stop: Command)
(see method Table: create ())

- **Press**

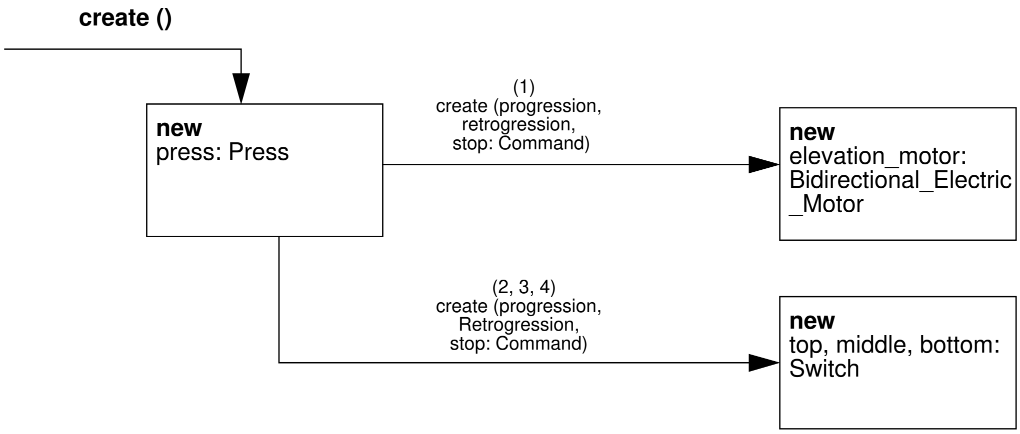

Fig. 42. Interaction graph for method Press: create ()

Description

Method Press: create ()
create the bidirectional electric motor elevation_motor with parameters: (press_upward, press_downward, press_stop) (1)
create the switch top with parameter: (press_top) (2)
create the switch middle with parameter: (press_middle) (3)
create the switch bottom with parameter: (press_bottom) (4)

Method Bidirectional_Electric_Motor: create (progression, retrogression, stop: Command)
(see method Table: create ())

Method Switch: create (n: Sensor_Name)
(see method Table: create ())

- **DepositBelt**

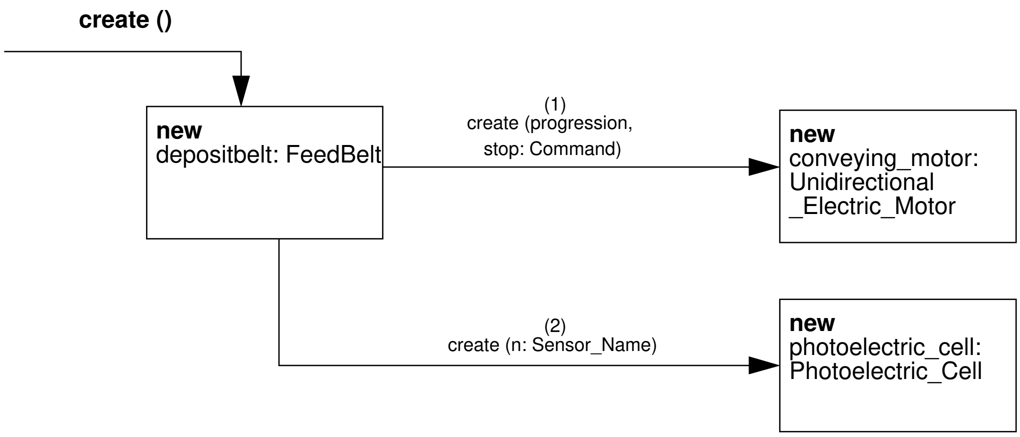

Fig. 43. Interaction graph for method DepositBelt: create ()

Description

Method DepositBelt: create ()
create the unidirectional electric motor conveying_motor with parameters:
(belt2_start, belt2_stop) (1)

create the photoelectric cell photoelectric_cell with parameter:
(belt2_blank_at_end) (2)

- **Crane**

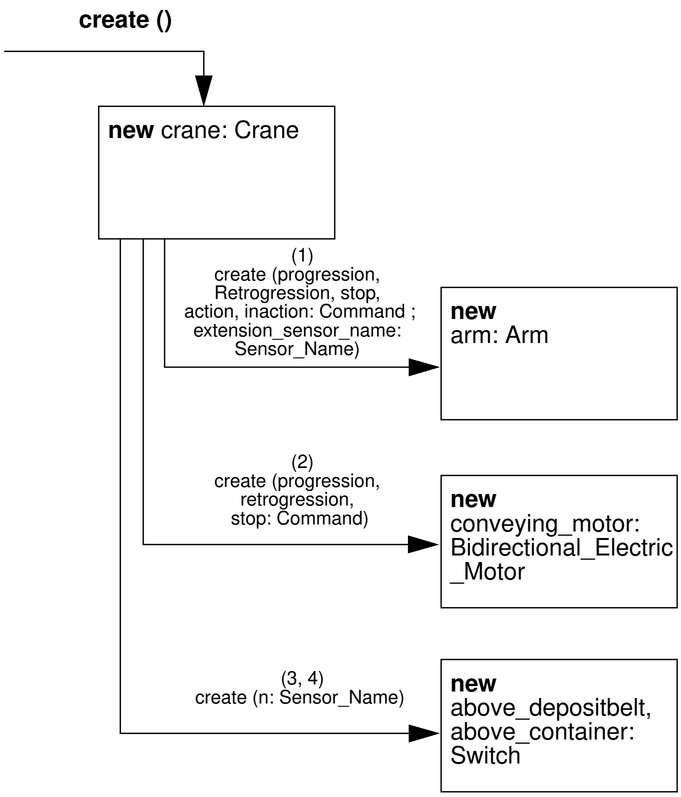

Fig. 44. Interaction graph for method Crane: create ()

Description

Method Crane: create ()
create the arm with parameters: (crane_lower, crane_lift, crane_stop_v, crane_mag_on, crane_mag_off, crane_mag_height) (1)
create the bidirectional electric motor conveying_motor with parameters:
(crane_to_contain, crane_to_belt2, crane_stop_h) (2)
create the switch above_depositbelt with parameter: (crane_above_belt2) (3)
create the switch above_container with parameter: (crane_above_contain) (4)

4.3 Visibility Graphs

In the following visibility graphs, all references are constant, and were not marked as such.

We give up using a collection for the arms in the Robot, or the switches in the Table or in the Press, because each of them has a defined identifier in the interaction graphs.

- **FeedBelt**

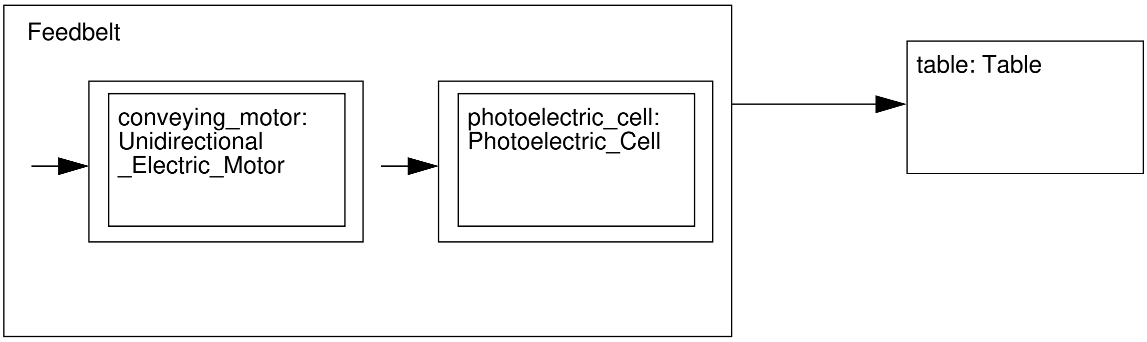

Fig. 45. Visibility graph for the class FeedBelt

- **Table**

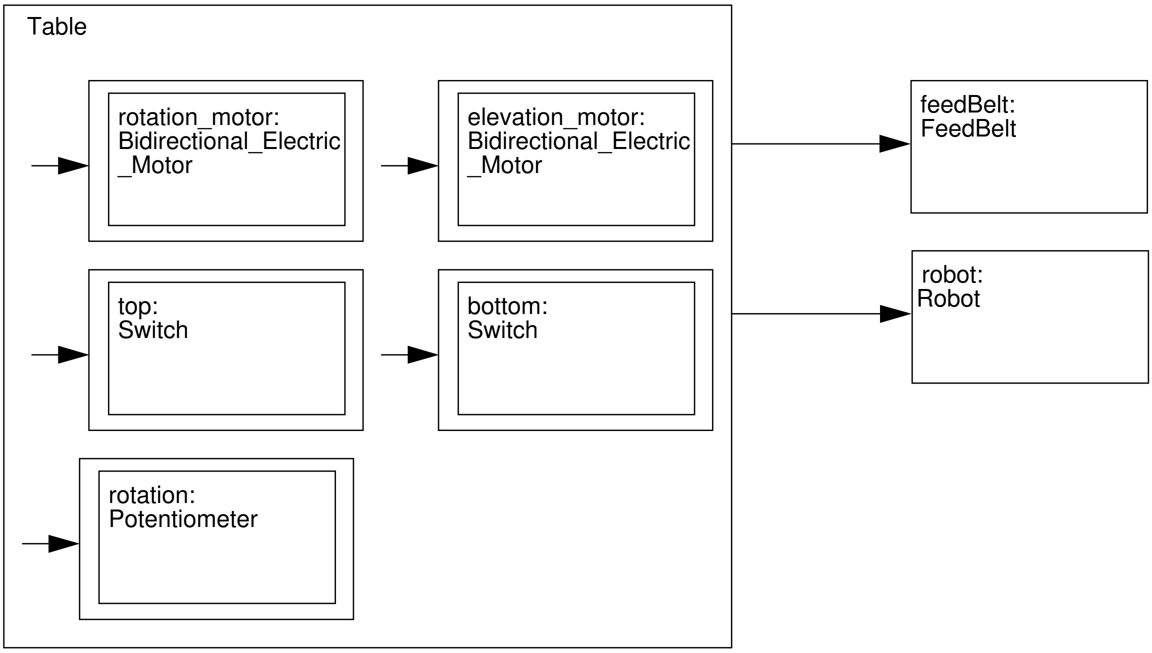

Fig. 46. Visibility graph for the class Table

- **Robot**

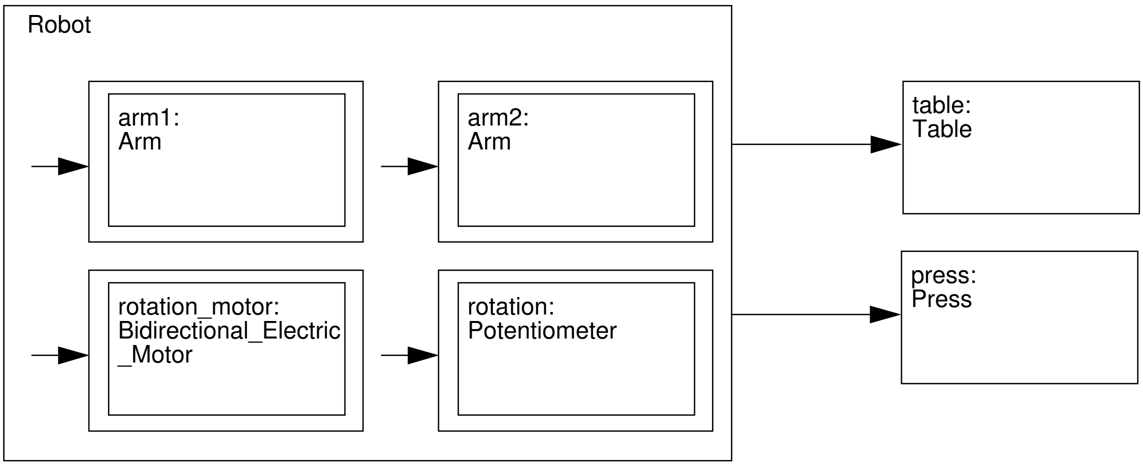

Fig. 47. Visibility graph for the class Robot

- **Press**

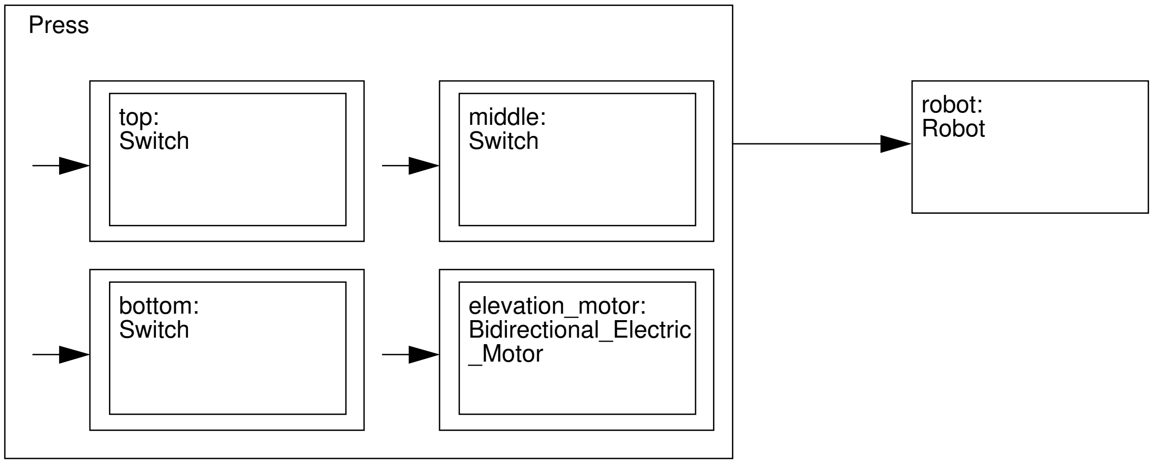

Fig. 48. Visibility graph for the class Press

- **DepositBelt**

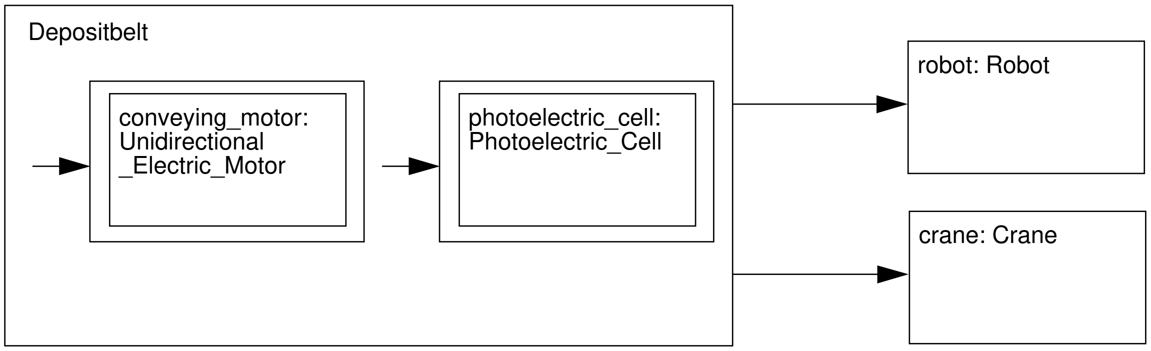

Fig. 49. Visibility graph for the class DepositBelt

- **Crane**

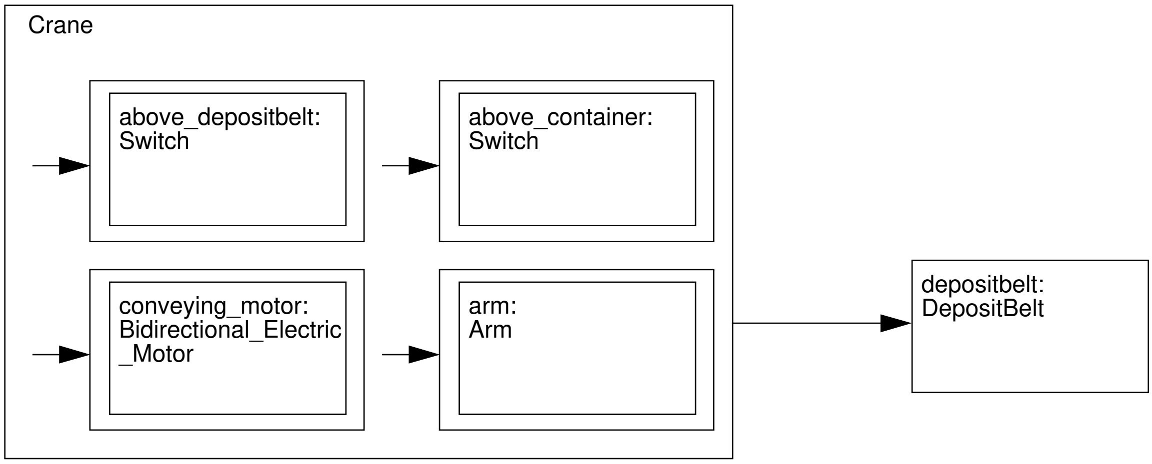

Fig. 50. Visibility graph for the class Crane

- **Arm**

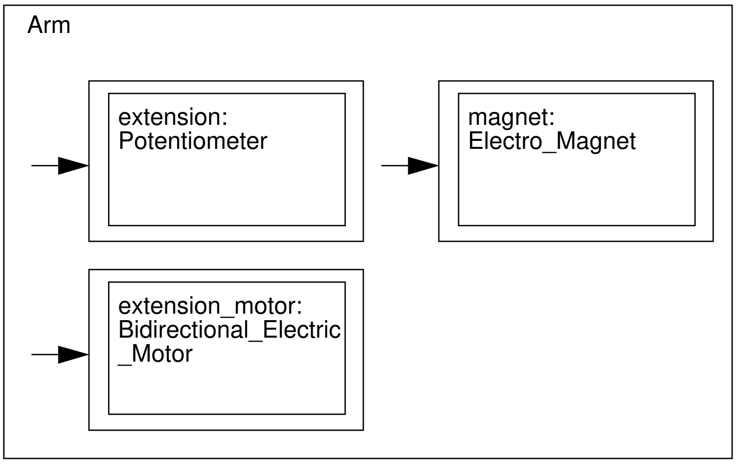

Fig. 51. Visibility graph for the class Arm

- **Actuator**

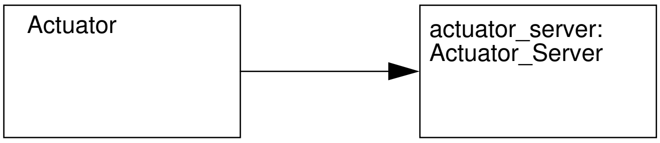

Fig. 52. Visibility graph for the classes Actuator

This visibility graph encompass the subclasses of Actuator (Unidirectional_Electric_Motor, Bidirectional_Electric_Motor, and Electro_Magnet)

- **Sensor**

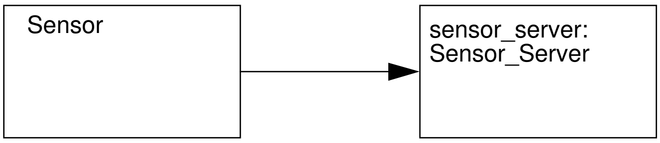

Fig. 53. Visibility graph for the classes Sensor

This visibility graph encompass the subclasses of Sensor (Potentiometer, Switch, and Photoelectric_Cell)

- **Actuator_Server**

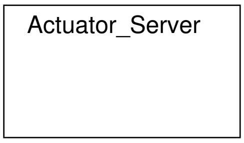

Fig. 54. Visibility graph for the class Actuator_Server

- **Sensor_Server**

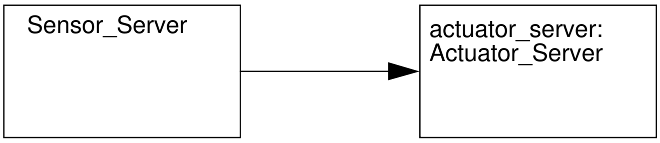

Fig. 55. Visibility graph for the class Sensor_Server

4.4 Inheritance Graphs

We keep the subtyping relationships for the sensors and the actuators found in the object model (see figure 16 and figure 17), but we refine them after analysis of a first :

- a potentiometer is a special kind of switch (see figure 56),

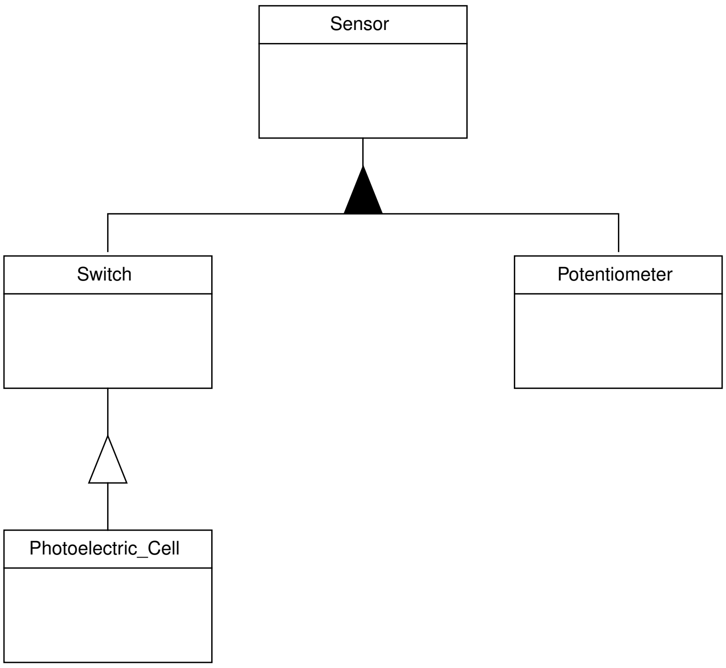

Fig. 56. Inheritance graph for sensors

- bidirectional and unidirectional electric motors are special kinds of electric motors (see figure 57).

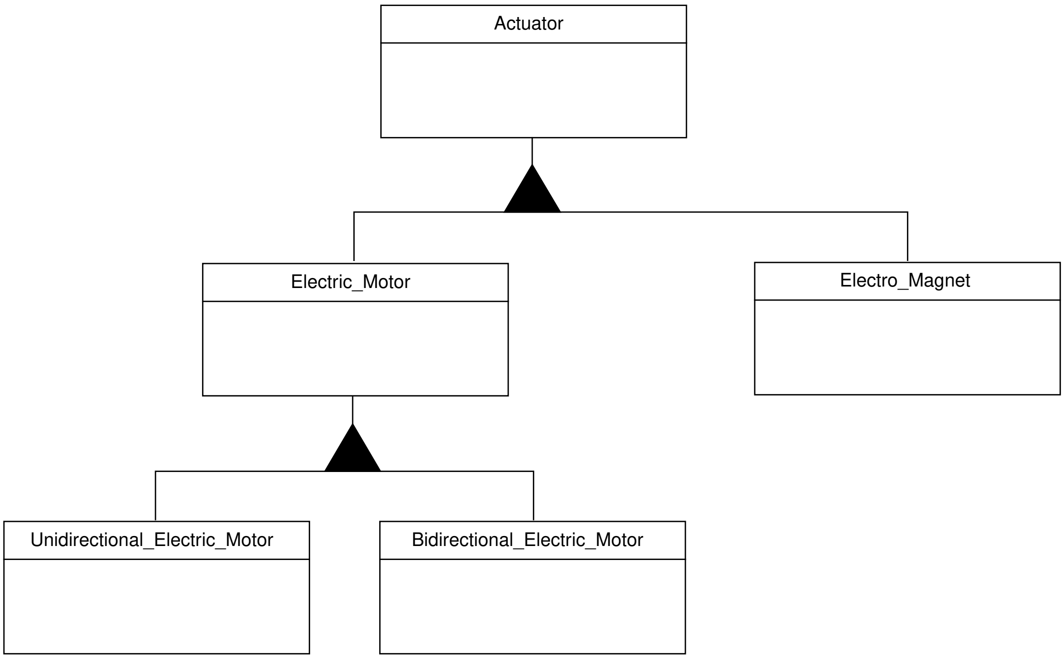

Fig. 57. Inheritance graph for actuators

We gave up making a class Belt as superclass for DepositBelt and FeedBelt, since those two classes have not much in common but for their structure (the conveying motor and the photoelectric cell).

4.5 Class Descriptions

4.5.1 Devices

- **FeedBelt**

class FeedBelt

// data attributes

// references
attribute constant conveying_motor: **exclusive bound** Unidirectional_Electric_Motor
attribute constant photoelectric_cell: **exclusive bound** Photoelectric_Cell
attribute constant table: **shared unbound** Table

// creation methods
method create ()

// public methods
method add_blank ()
method feed_table ()
method initialize ()

// private methods

endclass

- **Table**

class Table

// data attributes

// references
attribute constant rotation_motor: **exclusive bound** Bidirectional_Electric_Motor
attribute constant elevation_motor: **exclusive bound** Bidirectional_Electric_Motor
attribute constant bottom: **exclusive bound** Switch
attribute constant top: **exclusive bound** Switch
attribute constant rotation: **exclusive bound** Potentiometer
attribute constant feedbelt: **shared unbound** FeedBelt
attribute constant robot: **shared unbound** Robot

// creation methods
method create ()

// public methods
method go_load_position ()
method go_unload_position ()
method initialize ()

// private methods
method move (p: Table_Position)

endclass

- ## Robot

class Robot

// data attributes
attribute constant arm1_pick_extension: Extension := 0.5208
attribute constant arm1_pick_retraction: Extension := 0
attribute constant arm1_drop_extension: Extension := 0.6458
attribute constant arm1_drop_retraction: Extension := 0.3708
attribute constant arm2_pick_extension: Extension := 0.7971
attribute constant arm2_pick_retraction: Extension := 0
attribute constant arm2_drop_extension: Extension := 0.5707
attribute constant arm2_drop_retraction: Extension := 0
attribute constant depositonbelt_counter: Number_Blanks := 0

// references
attribute constant arm1: **exclusive bound** Arm
attribute constant arm2: **exclusive bound** Arm
attribute constant rotation_motor: **exclusive bound** Bidirectional_Electric_Motor
attribute constant rotation: **exclusive bound** Potentiometer
attribute constant table: **shared unbound** Table
attribute constant press: **shared unbound** Press
attribute constant depositbelt: **shared unbound** DepositBelt

// creation methods
method create ()

// public methods
method deposit_on_belt ()
method initialize ()
method load_press ()
method pick_from_press ()
method pick_from_table ()

// private methods
method move (p: Robot_Position)
method deposit_on_belt_int ()
method depositonbelt_init ()
method depositonbelt_increment ()
method depositonbelt_decrement ()

endclass

- **Press**

class Press

// data attributes

// references
attribute constant bottom: **exclusive bound** Switch
attribute constant middle: **exclusive bound** Switch
attribute constant top: **exclusive bound** Switch
attribute constant elevation_motor: **exclusive bound** Bidirectional_Electric_Motor
attribute constant robot: **shared unbound** Robot

// creation methods
method create ()

// public methods
method initialize ()
method forge ()
method go_load_position ()

// private methods

endclass

- **DepositBelt**

class DepositBelt

// data attributes

// references
attribute constant conveying_motor: **exclusive bound** Unidirectional_Electric_Motor
attribute constant photoelectric_cell: **exclusive bound** Photoelectric_Cell
attribute constant robot: **shared unbound** Robot
attribute constant crane: **shared unbound** Crane

// creation methods
method create ()

// public methods
method bring_past_end ()
method initialize ()

// private methods

endclass

- **Crane**

class Crane

// data attributes
attribute constant depositbelt_extension: Extension := 0.9450
attribute constant container_extension: Extension := 0.6593

// references
attribute constant conveying_motor: **exclusive bound** Bidirectional_Electric_Motor
attribute constant above_depositbelt: **exclusive bound** Switch
attribute constant above_container: **exclusive bound** Switch
attribute constant arm: **exclusive bound** Arm
attribute constant depositbelt: **shared unbound** DepositBelt

// creation methods

method create ()

// public methods
method initialize ()
method pick_from_belt ()

// private methods
method move (d: Direction)

endclass

- ## Arm

class Arm

// data attributes

// references
attribute constant extension_motor: **exclusive bound** Bidirectional_Electric_Motor
attribute constant extension: **exclusive bound** Potentiometer
attribute constant magnet: **exclusive bound** Electro_Magnet

// creation methods
method create (progression, retrogression, stop, action, inaction: Command; extension_sensor_name: Sensor_Name)

// public methods
method drop ()
method extend (e: Extension)
method pick ()
method retract (r: Extension)

// private methods

endclass

4.5.2 Sensors

- ## Sensor

class Sensor

// data attributes
attribute constant name: Sensor_Name

// references
attribute constant sensor_server: **shared unbound** Sensor_Server

// creation methods

// public methods

// private methods

endclass

- ## Switch

class Switch **isa** Sensor

// data attributes
attribute value: Boolean

// references

// creation methods

method create (n: Sensor_Name)

// public methods
method wait (b: Boolean)

// private methods

endclass

- **Potentiometer**

Potentiometer is a generic class which can take any type as attribute. In the production cell, the possible actual types for Value_Type are Angle and Extension.

class Potentiometer (Value_Type) **isa** Sensor

// data attributes
attribute value: Value_Type

// references

// creation methods
method create (n: Sensor_Name)

// public methods
method status (): Value_Type
method wait (v: Value_Type)

// private methods

endclass

- **Photoelectric_Cell**

class Photoelectric_Cell **isa** Switch

// data attributes

// references

// creation methods

// public methods

// private methods

endclass

4.5.3 Actuators

- **Actuator**

class Actuator

// data attributes
attribute status: Actuator_Status

// references
attribute constant actuator_server: **shared unbound** Actuator_Server

// creation methods

// public methods

// private methods

endclass

- ## Electric_Motor

class Electric_Motor **isa** Actuator

// data attributes
attribute progression: Command
attribute stop: Command

// references

// creation methods

// public methods
method turn_off ()

// private methods

endclass

- ## Unidirectional_Electric_Motor

class Unidirectional_Electric_Motor **isa** Electric_Motor

// data attributes

// references

// creation methods
method create (progression, stop: Command)

// public methods
method turn_on ()

// private methods

endclass

- ## Bidirectional_Electric_Motor

class Bidirectional_Electric_Motor **isa** Electric_Motor

// data attributes
attribute constant retrogression: Command

// references

// creation methods
method create (progression, retrogression, stop: Command)

// public methods
method turn_on (d: Direction)

// private methods

endclass

- ## Electro_Magnet

class Electro_Magnet **isa** Actuator

// data attributes
attribute constant action: Command
attribute constant inaction: Command

// references

// creation methods
method create (action, inaction: Command)

// public methods
method magnetize ()
method demagnetize ()

// private methods

endclass

4.5.4 Miscellaneous

- **Actuator_Server**

class Actuator_Server

// data attributes

// references

// creation methods

// public methods
method send (x: Command)
method status (): Vector

// private methods

endclass

- **Sensor_Server**

class Sensor_Server

// data attributes

// references
attribute constant actuator_server: **shared unbound** Actuator_Server

// creation methods

// public methods
method status (n: Sensor_Name): Boolean
method status (n: Sensor_Name): Angle
method status (n: Sensor_Name): Extension

// private methods

endclass

5. Data Dictionary

5.1 External View of the Controller

Name	Kind		Description	References
Operator	*agent*		operates the production cell. (The operator can turn on, turn off, and stop the plant. He also can add new blanks onto the feed belt.)	§ 3.1 (fig 10)
operator_turn_on	*event*	*in*	turns on the controller	§ 3.1 (fig 10)
operator_turn_off	*event*	*in*	turns off the controller	§ 3.1 (fig 10)
operator_add_blank	*event*	*in*	adds a new blank on the feed belt	§ 3.1 (fig 10)
operator_stop	*event*	*in*	stop the system (emergency)	§ 3.1 (fig 10)
Feedbelt	*agent*		device feed belt	§ 3.1 (fig 10)
belt1_start	*event*	*out*	starts the feed belt	§ 3.1 (fig 10)
belt1_stop	*event*	*out*	stops the feed belt	§ 3.1 (fig 10)
belt1_blank_at_end	*event*	*in*	there is a blank at the end of the feed belt	§ 3.1 (fig 10)
Table	*agent*		device table	§ 3.1 (fig 10)
table_left	*event*	*out*	starts rotation to the left	§ 3.1 (fig 10)
table_right	*event*	*out*	starts rotation to the right	§ 3.1 (fig 10)
table_stop_h	*event*	*out*	stops the rotation	§ 3.1 (fig 10)
table_upward	*event*	*out*	starts upward moving	§ 3.1 (fig 10)
table_downward	*event*	*out*	starts downward moving	§ 3.1 (fig 10)
table_stop_v	*event*	*out*	stops the vertical moving	§ 3.1 (fig 10)
table_bottom	*event*	*in*	the table is in bottom position	§ 3.1 (fig 10)
table_top	*event*	*in*	the table is in top position	§ 3.1 (fig 10)
table_angle	*event*	*in*	position of the table	§ 3.1 (fig 10)
Robot	*agent*		device robot	§ 3.1 (fig 10)
robot_left	*event*	*out*	starts rotation to the left	§ 3.1 (fig 10)
robot_right	*event*	*out*	starts rotation to the right	§ 3.1 (fig 10)
robot_stop	*event*	*out*	stops the rotation	§ 3.1 (fig 10)
arm1_forward	*event*	*out*	starts extension of arm1	§ 3.1 (fig 10)
arm1_backward	*event*	*out*	starts retraction of arm1	§ 3.1 (fig 10)
arm1_stop	*event*	*out*	stops arm1	§ 3.1 (fig 10)
arm1_mag_on	*event*	*out*	activates magnet of arm1	§ 3.1 (fig 10)
arm1_mag_off	*event*	*out*	deactivates magnet of arm1	§ 3.1 (fig 10)
arm2_froward	*event*	*out*	starts extension of arm2	§ 3.1 (fig 10)
arm2_backward	*event*	*out*	starts retraction of arm2	§ 3.1 (fig 10)
arm2_stop	*event*	*out*	stops arm2	§ 3.1 (fig 10)

Name	Kind		Description	References
arm2_mag_on	*event*	*out*	activates magnet of arm2	§ 3.1 (fig 10)
arm2_mag_off	*event*	*out*	deactivates magnet of arm2	§ 3.1 (fig 10)
arm1_extension	*event*	*in*	extension of the frist arm	§ 3.1 (fig 10)
arm2_extension	*event*	*in*	extension of the second arm	§ 3.1 (fig 10)
robot_angle	*event*	*in*	position of the robot	§ 3.1 (fig 10)
Press	*agent*		device press	§ 3.1 (fig 10)
press_upward	*event*	*out*	starts upward moving	§ 3.1 (fig 10)
press_downward	*event*	*out*	starts downward moving	§ 3.1 (fig 10)
press_stop	*event*	*out*	stops the press	§ 3.1 (fig 10)
press_bottom	*event*	*in*	the press is in bottom position	§ 3.1 (fig 10)
press_middle	*event*	*in*	the press is in middle position	§ 3.1 (fig 10)
press_top	*event*	*in*	the press is in top position (pressing)	§ 3.1 (fig 10)
DepositBelt	*agent*		device deposit belt	§ 3.1 (fig 10)
belt2_start	*event*	*out*	starts the deposit belt	§ 3.1 (fig 10)
belt2_stop	*event*	*out*	stops the deposit belt	§ 3.1 (fig 10)
belt2_blank_at_end	*event*	*in*	there is a blank at the end of the deposit belt	§ 3.1 (fig 10)
Crane	*agent*		device crane	§ 3.1 (fig 10)
crane_to_contain	*event*	*out*	starts horizontal movement towards the container	§ 3.1 (fig 10)
crane_to_belt2	*event*	*out*	starts horizontal movement towards the deposit belt	§ 3.1 (fig 10)
crane_stop_h	*event*	*out*	stops horizontal movement	§ 3.1 (fig 10)
crane_lift	*event*	*out*	starts extension of the arm	§ 3.1 (fig 10)
crane_lower	*event*	*out*	starts retraction of the arm	§ 3.1 (fig 10)
crane_strop	*event*	*out*	stops the arm	§ 3.1 (fig 10)
crane_mag_on	*event*	*out*	activates crane's magnet	§ 3.1 (fig 10)
crane_mag_off	*event*	*out*	deactivates crane's magnet	§ 3.1 (fig 10)
crane_above_belt2	*event*	*in*	the crane is above the deposit belt	§ 3.1 (fig 10)
crane_above_contain	*event*	*in*	the crane is above the container	§ 3.1 (fig 10)
crane_mag_height	*event*	*in*	the height of the crane magnet	§ 3.1 (fig 10)

5.2 Internal View of the Controller, Life-Cycle Model, and Operation Model

Name	Kind		Description	References
Operator	*agent*	*sends* add_blank	Operates the production cell. (The operator can turn on, turn off, and stop the plant. He also can add new blanks onto the feed belt.)	§ 3.1 (fig 11)
FeedBelt	*agent*	*sends* go_unload_position *receives* feed_table	Controller for the device feed belt.	§ 3.1 (fig 11)
Table	*agent*	*sends* feed_table, pick_from_table *receives* go_unload_position, go_load_position	Controller for the device table.	§ 3.1 (fig 11)
Robot	*agent*	*sends* go_load_position, forge, go_load_position *receives* pick_from_table, load_press, pick_from_press, deposit_on_belt	Controller for the device robot.	§ 3.1 (fig 11)
Press	*agent*	*sends* load_press, pick_from_press *receives* go_load_position, forge	Controller for the device press.	§ 3.1 (fig 11)
DepositBelt	*agent*	*sends* deposit_on_belt, pick_from_belt *receives* bring_past_end	Controller for the device deposit belt.	§ 3.1 (fig 11)
Crane	*agent*	*sends* bring_past_end *receives* pick_from_belt	Controller for the device crane.	§ 3.1 (fig 11)
add_blank	*event*	*out* Operator *in* FeedBelt	Add a blank on the feed belt.	§ 3.1 (fig 11) § 3.4.1
go_load_position	*event*	*out* Robot *in* Table	Put the table in load position.	§ 3.1 (fig 11) § 3.4.2
go_unload_position	*event*	*out* FeedBelt *in* Table	Put the table in unload position.	§ 3.1 (fig 11) § 3.4.2
feed_table	*event*	*in* FeedBelt *out* Table	Bring the plate onto the table.	§ 3.1 (fig 11) § 3.4.1
pick_from_table	*event*	*out* Table *in* Robot	Pick the plate from the table.	§ 3.1 (fig 11) § 3.4.3
load_press	*event*	*out* Press *in* Robot	Load the plate onto the press	§ 3.1 (fig 11) § 3.4.3

Name	Kind		Description	References
pick_from_press	*event*	*out* Press *in* Robot	Pick the plate from the press.	§ 3.1 (fig 11) § 3.4.3
go_load_position	*event*	*out* Robot *in* Press	Put the press in load position.	§ 3.1 (fig 11) § 3.4.4
forge	*event*	*out* Robot *in* Press	Forge the plate	§ 3.1 (fig 11) § 3.4.4
deposit_on_belt	*event*	*out* DepositBelt *in* Robot	Deposit a plate on the deposit belt	§ 3.1 (fig 11) § 3.4.3
bring_past_end	*event*	*out* Crane *in* DepositBelt	Bring the plate to the end of the deposit belt	§ 3.1 (fig 11) § 3.4.5
pick_from_belt	*event*	*out* DepositBelt *in* Crane	Pick a plate and bring it from the deposit belt to the container. Drop the plate into the container. Go back waiting over the deposit belt.	§ 3.1 (fig 11) § 3.4.6

5.3 Object Model

Name	Kind		Description	References
Actuator_Status	*type*	on | off	Enumeration	
Angle	*type*	-180 .. 180 range 1	Discrete range	
Boolean	*type*	true | false	Enumeration	
Number_Blanks	*type*	0 .. 2 range 1	Enumeration	
Extension	*type*	0 .. 1 range 0.0001	Floating range	
Command	type	belt1_start | belt1_stop | belt2_start | belt2_stop | table_left | table_right | table_stop_h | table_upward | table_downward | table_stop_v | robot_left | robot_right | robot_stop | arm1_forward | arm1_backward | arm1_stop | arm1_mag_on | arm1_mag_off | arm2_forward | arm2_backward | arm2_stop | arm2_mag_on | arm2_mag_off | press_upward | press_downward | press_stop | crane_to_contain | crane_to_belt2 | crane_stop_h | crane_lift | crane_lower | crane_stop_v | crane_mag_on | crane_mag_off | get_status	Enumeration	
Sensor_Name	*type*	press_bottom | press_middle | press_top | arm1_extension | arm2_extension | robot_angle | table_bottom | table_top | table_angle | crane_above_belt2 | crane_above_contain | crane_mag_height | belt1_blank_at_end | belt2_blank_at_end	Enumeration	
FeedBelt	*class*		The controller of the device feed belt	§ 3.2 (fig 15)

Name	Kind		Description	References
OnFeedBelt	*relationship*		The feed belt carries a plate.	§ 3.2 (fig 15)
Table	*class*		The controller of the device table	§ 3.2 (fig 15)
OnTable	*relationship*		The table carries a plate.	§ 3.2 (fig 15)
Robot	*class*		The controller of the device robot.	§ 3.2 (fig 15)
Robot.arm1_pick_extension	*attribute*	Extension	Extension of the first arm to pick a plate	§ 3.2 (fig 15)
Robot.arm1_pick_retraction	*attribute*	Extension	Retraction of the first arm after picking a plate	§ 3.2 (fig 15)
Robot.arm1_drop_extension	*attribute*	Extension	Extension of the first arm to drop a plate	§ 3.2 (fig 15)
Robot.arm1_drop_retraction	*attribute*	Extension	Retraction of the first arm after dropping a plate	§ 3.2 (fig 15)
Robot.arm2_pick_extension	*attribute*	Extension	Extension of the second arm to pick a plate	§ 3.2 (fig 15)
Robot.arm2_pick_retraction	*attribute*	Extension	Retraction of the second arm after picking a plate	§ 3.2 (fig 15)
Robot.arm2_drop_extension	*attribute*	Extension	Extension of the second arm to drop a plate	§ 3.2 (fig 15)
Robot.arm2_drop_retraction	*attribute*	Extension	Retraction of the second arm after dropping a plate	§ 3.2 (fig 15)
Robot.depositonbelt_counter	*attribute*	Number_Blanks	Count the number of blanks the robot can drop on the deposit belt	§ 4.2.3
Arm	*class*		The controller of an arm (of the robot).	§ 3.2 (fig 15)
OnArm	*relationship*		The arm carries a plate.	§ 3.2 (fig 15)
Press	*class*		The controller of the device Press	§ 3.2 (fig 15)
OnPress	*relationship*		The press carries a plate.	§ 3.2 (fig 15)
DepositBelt	*class*		The controller of the device deposit belt.	§ 3.2 (fig 15)
OnDepositBelt	*relationship*		The deposit belt carries a plate.	§ 3.2 (fig 15)
OnDepositBelt.position	*attribute*		Indicates whether a plate is at the begin or at the end of the deposit belt.	§ 3.2 (fig 15)
Crane	*class*		The controller of the device Crane	§ 3.2 (fig 15)
Crane.depositbelt_extension	*attribute*	Extension	Extension of the arm to pick a plate	§ 3.2 (fig 15)
Crane.container_extension	*attribute*	Extension	Extension of the arm to drop a plate	§ 3.2 (fig 15)
OnCrane	*relationship*		The crane carries a plate.	§ 3.2 (fig 15)
Plate	*class*		A piece of metal handled by the production cell.	§ 3.2 (fig 15)
Plate.forged	*attribute*	Boolean	forged indicates whether the plate has been forged or not.	§ 3.2 (fig 15)
Actuator	*class*		The class of all actuators	§ 3.2 (fig 17)

Name	Kind		Description	References
Actuator.status	*attribute*	Actuator_Status	The status of an actuator (on/off).	§ 3.2 (fig 15, fig 17)
Unidirectional_Electric_Motor	*class*		An electric motor running in only one direction.	§ 3.2 (fig 15, fig 17)
Unidirectional_Electric_Motor .progression	*attribute*	Command	The command to move the motor in forward/upward/left/extension direction	§ 3.2 (fig 15, fig 17)
Unidirectional_Electric_Motor .stop	*attribute*	Command	The command to stop the motor.	§ 3.2 (fig 15, fig 17)
Bidirectional_Electric_Motor	*class*		An electric motor running in two directions.	§ 3.2 (fig 15, fig 17)
Bidirectional_Electric_Motor.p rogression	*attribute*	Command	The command to move the motor in forward/upward/left/extension direction	§ 3.2 (fig 15, fig 17)
Bidirectional_Electric_Motor.r etrogression	*attribute*	Command	The command to move the motor in backward/downward/right/ retraction direction	§ 3.2 (fig 15, fig 17)
Bidirectional_Electric_Motor.s top	*attribute*	Command	The command to stop the motor.	§ 3.2 (fig 15, fig 17)
Electro_Magnet	*class*		A actuator that can hold plates.	§ 3.2 (fig 15, fig 17)
Electro_Magnet.action	*attribute*	Command	The command to magnetize the magnet.	§ 3.2 (fig 15, fig 17)
Electro_Magnet.inaction	*attribute*	Command	The command to demagnetize the magnet.	§ 3.2 (fig 15, fig 17)
Sensor	*class*		The class of all sensors	§ 3.2 (fig 16)
Sensor.name	*attribute*	Sensor_Name	Each sensor is identified by a name.	§ 3.2 (fig 16)
Switch	*class*		A sensor that can be either true or false.	§ 3.2 (fig 15, fig 16)
Switch.value	*attribute*	Boolean	The value of a switch.	§ 3.2 (fig 15, fig 16)
Potentiometer	*class*		A sensor the value of which is given among a range.	§ 3.2 (fig 15, fig 16)
Potentiometer.value	*attribute*	Extension	The value of a potentiometer.	§ 3.2 (fig 15, fig 16)
Photoelectric_Cell	*class*		A sensor that can be either true or false.	§ 3.2 (fig 15, fig 16)
Photoelectric_Cell.value	*attribute*	Boolean	The value of a photoelectric cell.	§ 3.2 (fig 15, fig 16)

5.4 Design Objects

Name	Kind	Description	References
Actuator_Server	*class*	Server to handle th ciritical ressource terminal	§ 4.1 (fig 19)
ReliesOn2	*relationship*	Connection between the actuators and their servers	§ 4.1 (fig 19)
Sensor_Server	*class*	Server to analyze the status vector and distinguish the value of each sensor	§ 4.1 (fig 19)
ReliesOn1	*relationship*	Connection between the sensors and their servers	§ 4.1 (fig 19)

5.5 Interaction Graphs

Name	Kind		Description	References
Direction	*type*	progressive | regressive	enumeration	
Table_Position	*type*	loadposition | unloadposition	enumeration	
Robot_Position	*type*	1 | 2 | 3 | 4 | 5	enumeration	
Vector	*type*	array	values of the sensor	
Actuator_Server: send	*method*	(x: Command)	send a command to the production cell (or to the simulator)	§ 4.2.1 (fig 20)
Actuator_Server: status	*method*	()	asks the production cell (or the simulator) for the values of the sensors	§ 4.2.1 (fig 21)
Arm: create	*method*	(progression, retrogression, stop, action, inaction: Command; extension_motor_name: Sensor_Name)	Create an instance of Arm	§ 4.2.7 (fig 41)
Arm: drop	*method*	()	Drop a plate from the arm	§ 4.2.3 (fig 28)
Arm: extend	*method*	(e: Extension)	Extend the arm	§ 4.2.3 (fig 29)
Arm: pick	*method*	()	Pick a plate with the arm	§ 4.2.3 (fig 27)
Arm: retract	*method*	(r: Extension)	Retract the arm	§ 4.2.3 (fig 27)
Bidirectional_Electric_Motor: create	*method*	(progression, retrogression, stop: Command)	Create an instance of Bidirectional_Electric_Motor	§ 4.2.7 (fig 41)
Bidirectional_Electric_Motor: turn_off	*method*	()	Turn off the motor	§ 4.2.2 (fig 23)
Bidirectional_Electric_Motor: turn_on	*method*	(d: Direction)	Turn on the motor	§ 4.2.2 (fig 23)

Name	Kind		Description	References
Crane: create	*method*	()	Create an instance of Crane	§ 4.2.7 (fig 44)
Crane: initialize	*method*	()	Initialize the crane	§ 4.2.6 (fig 37)
Crane: pick_from_belt	*method*	()	Pick a plate and bring it from the deposit belt to the container. Drop the plate into the container. Go back waiting over the deposit belt.	§ 4.2.6 (fig 38)
Crane: move	*method*	(d: Direction)	Move the crane	§ 4.2.6 (fig 37),§ 4.2. 6 (fig 38)
DepositBelt: create	*method*	()	Create an instance of DepositBelt	§ 4.2.7 (fig 43)
DepositBelt: bring_past_end	*method*	()	Bring the plate to the end of the deposit belt	§ 4.2.5 (fig 36)
DepositBelt: initialize	*method*	()	Initialize the deposit belt	§ 4.2.5 (fig 35)
Electro_Magnet: create	*method*	(action, inaction: Command)	Create an instance of Electro_Magnet	§ 4.2.7 (fig 41)
Electro_Magnet: demagnetize	*method*	()	Turn off the magnet	§ 4.2.3 (fig 28)
Electro_Magnet: magnetize	*method*	()	Turn on the magnet	§ 4.2.3 (fig 27)
FeedBelt: create	*method*	()	Create an instance of FeedBelt	§ 4.2.7 (fig 39)
FeedBelt: add_blank	*method*	()	Add a blank on the feed belt.	§ 4.2.1 (fig 21)
FeedBelt: feed_table	*method*	()	Bring the plate onto the table.	§ 4.2.1 (fig 22)
FeedBelt: initialize	*method*	()	Initialize the feed belt	§ 4.2.1 (fig 20)
Photoelectric_Cell: create	*method*	(n: Sensor_Name)	Create an instance of Photoelectric_Cell	§ 4.2.7 (fig 39)
Photoelectric_Cell: wait	*method*	(b: Boolean)	Wait for the photoelectric_cell to be b	§ 4.2.1 (fig 21)
Potentiometer: create	*method*	(n: Sensor_Name)	Create an instance of Potentiometer	§ 4.2.7 (fig 40)
Potentiometer: status	*method*	(): Angle	Returns the value of the potentiometer	§ 4.2.3 (fig 27)
Potentiometer: wait	*method*	(v: Value_Type)	Wait for the potentiometer to have the value v	§ 4.2.2 (fig 23) § 4.2.3 (fig 27)
Press: create	*method*	()	Create an instance of Press	§ 4.2.7 (fig 42)
Press: forge	*method*	()	Forge the plate	§ 4.2.4 (fig 34)

Name	Kind		Description	References
Press: go_load_position	*method*	()	Put the press in load position.	§ 4.2.4 (fig 33)
Press: initialize	*method*	()	Initialize the press	§ 4.2.4 (fig 32)
Robot: create	*method*	()	Create an instance of Robot	§ 4.2.7 (fig 41)
Robot: deposit_on_belt	*method*	()	Increment by 1 the number of blanks the robot can drop on the deposit belt	§ 4.2.3 (fig 30)
Robot: deposit_on_belt_int	*method*	()	Deposit a plate on the deposit belt	§ 4.2.3 (fig 31)
Robot: depositonbelt_increment	*method*	()	Increment by 1 depositonbelt_counter	§ 4.2.3 (fig 30)
Robot: depositonbelt_decrement	*method*	()	Decrement by 1 depositonbelt_counter	§ 4.2.3 (fig 31)
Robot: depositonbelt_init	*method*	()	Assign 0 to depositonbelt_counter	§ 4.2.7 (fig 41)
Robot: initialize	*method*	()	Initialize the robot	§ 4.2.3 (fig 26)
Robot: load_press	*method*	()	Load the plate onto the press	§ 4.2.3 (fig 28)
Robot: pick_from_press	*method*	()	Pick the plate from the press.	§ 4.2.3 (fig 29)
Robot: pick_from_table	*method*	()	Pick the plate from the table	§ 4.2.3 (fig 27)
Robot: move	*method*	(p: Robot_Position)	Move the robot in one of tis 5 predefined positions	§ 4.2.3 (fig 27)
Sensor_Server: status	*method*	(n: Sensor_Name): Angle	Status of sensor n	§ 4.2.2 (fig 23)
Sensor_Server: status	*method*	(n: Sensor_Name): Extension	Status of sensor n	§ 4.2.1 (fig 27)
Sensor_Server: status	*method*	(n: Sensor_Name): Boolean	Status of sensor n	§ 4.2.1 (fig 21)
Switch: create	*method*	(n: Sensor_Name)	Create an instance of Switch	§ 4.2.7 (fig 40)
Switch: wait	*method*	(b: Boolean)	Wait for the swith to become b	§ 4.2.2 (fig 23)
Table: create	*method*	()	Creation of an instance of Table	§ 4.2.7 (fig 40)
Table: go_load_position	*method*	()	put the table in load position.	§ 4.2.2 (fig 24)
Table: go_unload_position	*method*	()	Put the table in unload position.	§ 4.2.2 (fig 25)
Table: initialize	*method*	()	Initialize the table	§ 4.2.2 (fig 23)

Name	Kind		Description	References
Table: move	*method*	(p: Table_Position)	Move the table in load or unload position	§ 4.2.2 (fig 23), § 4.2.2 (fig 25)
Unidirectional_Electric_ Motor: create	*method*	(progression, stop: Command)	Creation of an instance of Unidirectional_Electric_Motor	§ 4.2.7 (fig 39)
Unidirectional_Electric_ Motor: turn_off	*method*	()	Turn off the motor	§ 4.2.1 (fig 20)
Unidirectional_Electric_ Motor: turn_on	*method*	()	Turn on the motor	§ 4.2.1 (fig 21)

6. References

[1] O. Biberstein, D. Buchs, and N. Guelfi. *Object-Oriented Nets with Algebraic Specifications: The CO-OPN/2 formalism*. Advances in Petri Nets on Object-Orientation, Lecture Notes in Computer Science. Springer-Verlag, 1997.

[2] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes. *Object-Oriented Development The Fusion Method*. Object-Oriented Series. Prentice Hall, 1994.

[3] C. Lewerentz and T. Lindner, editors. *Formal Development of Reactive System: Case Study Production Cell*, volume 891 of *LNCS (Lecture Notes in Computer Science)*. Springer Verlag, 1995.

[4] C. Péraire, S. Barbey, and D. Buchs. Testing the production cell case study from CO-OPN specification. Technical Report 98/299, EPFL-DI, 1998.