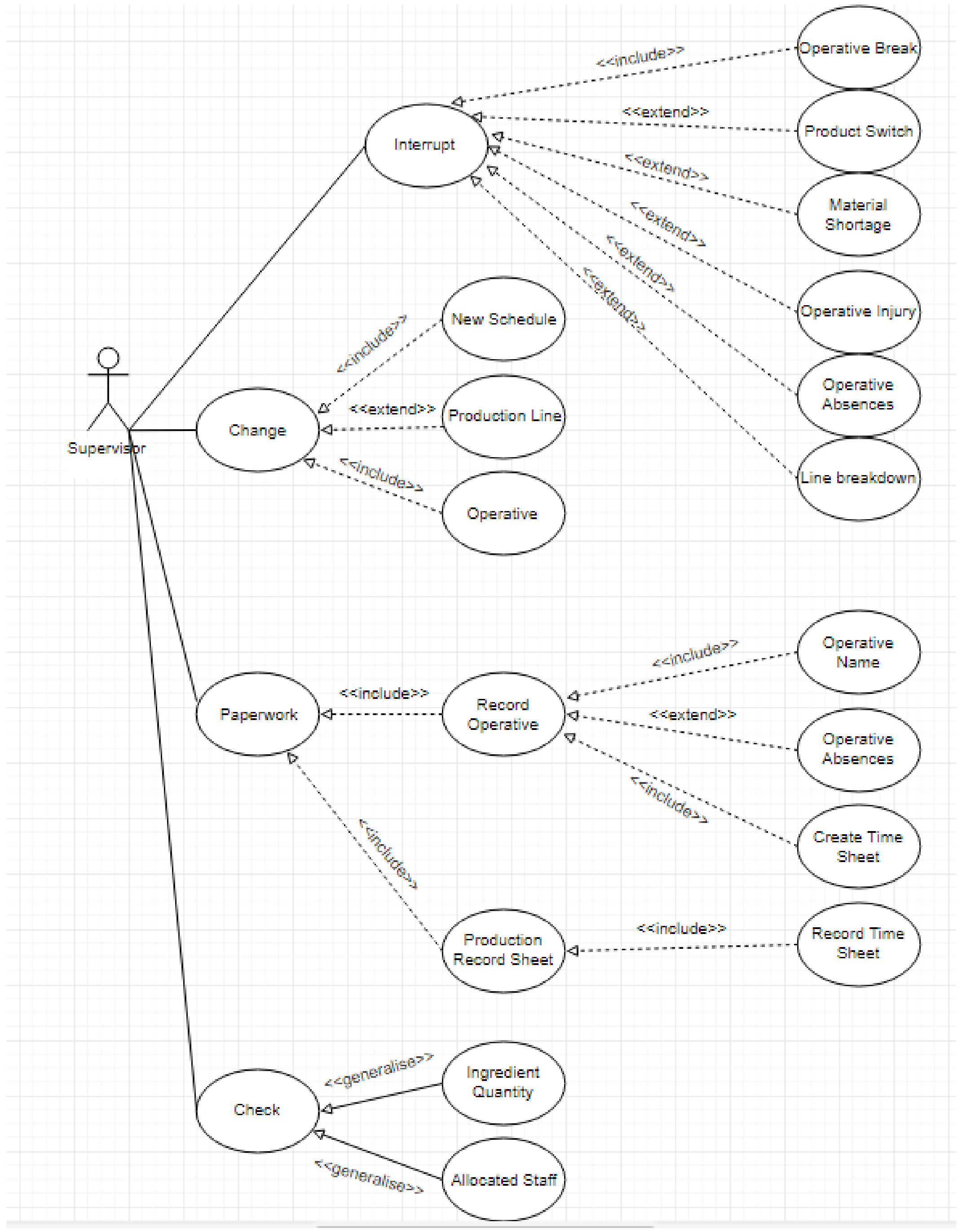
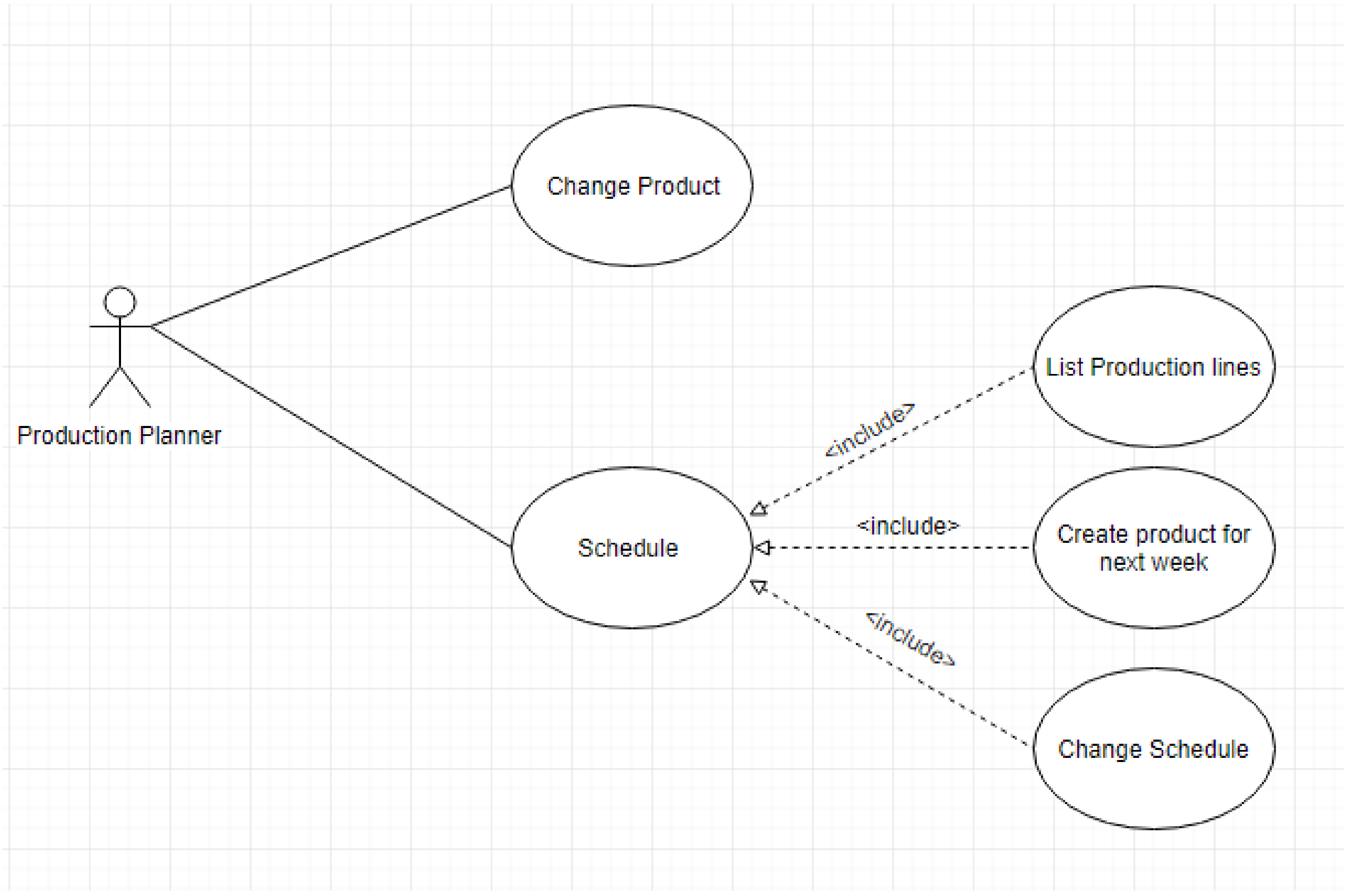
Software Engineering Assignment1: Diagrams (A, B)




Change Product
Production Planner
List Production lines
<include>
<include>
Create product for next week
Schedule
<include>
Change Schedule

Use-Case Descriptions (C)

Name of use case: • START LINE RUN

Pre-conditions:

1. Check there are enough ingredients.
2. Are allocated staff present.
3. Supervisor present to switch line on.

Post conditions:

1. Production line starts successfully.

Actor: Supervisor.

Purpose: Description of the beginning of daily line run.

Description:

1. Check storage for enough ingredients (at least enough for a smooth run until next supply drop.)
2. Check allocated staff turn up to operate line.
3. Supervisor switches on line if previous two conditions are met.

Name of use case: • RECORD EMPLOYEE JOINING THE LINE

Pre-conditions:

1. Operative doesn’t have a timesheet.
2. Employee joining present.
3. Supervisor present to record employee.

Post conditions:

1. Employee joins the production line successfully.

Actor: Supervisor, Operatives.

Purpose: Describing the process of operatives when joining a production line.

Description:

1. Take name of staff at beginning of run.
2. Copy the job number from the job card to the production record sheet.
3. If it's the first time the operatives worked this week, make a new timesheet.

Name of use case: • RECORD EMPLOYEE LEAVING THE LINE

Pre-conditions:

1. All operatives are present at the line.
2. Supervisor present to record employee.

Post conditions:

1. Operative successfully leaves the line.
2. Supervisor takes note of absence.

Actor: Supervisor, Operator.

Purpose: Description of the process an operative needs to do to leave the production line.

Description:

1. If an operative leaves the line, a rough note is taken of this person, and how long they are absent for.
2. If someone is missing, or leave on sick leave, a replacement has to be found as quickly as possible.

Name of use case: • STOP LINE

Pre-conditions:

1. Production line is running as normal.

Post conditions:

2. Production line has been successfully stopped.

Actor: Supervisor

Purpose: Description of the process of stopping the line, and what would case a stop.

Description:

1. Lines stops.

Name of use case: • RECORD LINE PROBLEM

Pre-conditions:

1. Line has been stopped to resolve problem.
2. Maintenance present

Post conditions:

1. Line problem has been recorded
2. Line has been successfully fixed.
3. Line has been started again.

Actor: Maintenance, Supervisor.

Purpose: Describing process of recording and fixing a line problem

Description:

1. If the line brakes down, maintenance would have to be called and record the downtime the production line isn’t running.

2. If the line runs out of ingredients, this would involve contacting the warehouse, farm or outside supplier.
3. If people go missing, or leave early because they are ill, a replacement for them must be found as quickly as possible.

Name of use case: • END LINE RUN

Pre-conditions:

1. Line is running
2. Supervisor present
3. Product Control present

Post conditions:

4. Line run has been successfully stopped.

Actor: Supervisor, Production Control.

Purpose: Describing the formal process of ending a line run, and relevant checks.

Description:

1. Supervisor notes finishing time on production record sheet.
2. Supervisor then phones Production Control to verify quantity produced, this is then noted on the production record sheet.
3. Supervisor then totals absences.
4. Total hours then recorded for each operative.
5. If someone joined mid run, a new timesheet is made and hours added in.
6. Quantity recorded, and unused ingredients returned to warehouse.
7. Line tidied, and readied for next run.

CANDIDATE CLASSES (D):

>is it beyond the scope of the system?

>does it refer to the system as a whole?

>does it duplicate another class?

>is it too vague?

>is it too tied up with physical inputs and outputs?

>is it really an attribute?

>is it really an oepration>

?is it really an association?

BOUNDARY CLASS(objects that interface with system actors):

Maintenance

Production Line control

Production record sheet

CONTROL CLASS(objects co-ordinate and control other objects):

Supervisor

Timesheet

Operative

Job card

ENTITY CLASS(objects representing information and behaviour in the application domain):

Product

Farm

Supplier

Warehouse

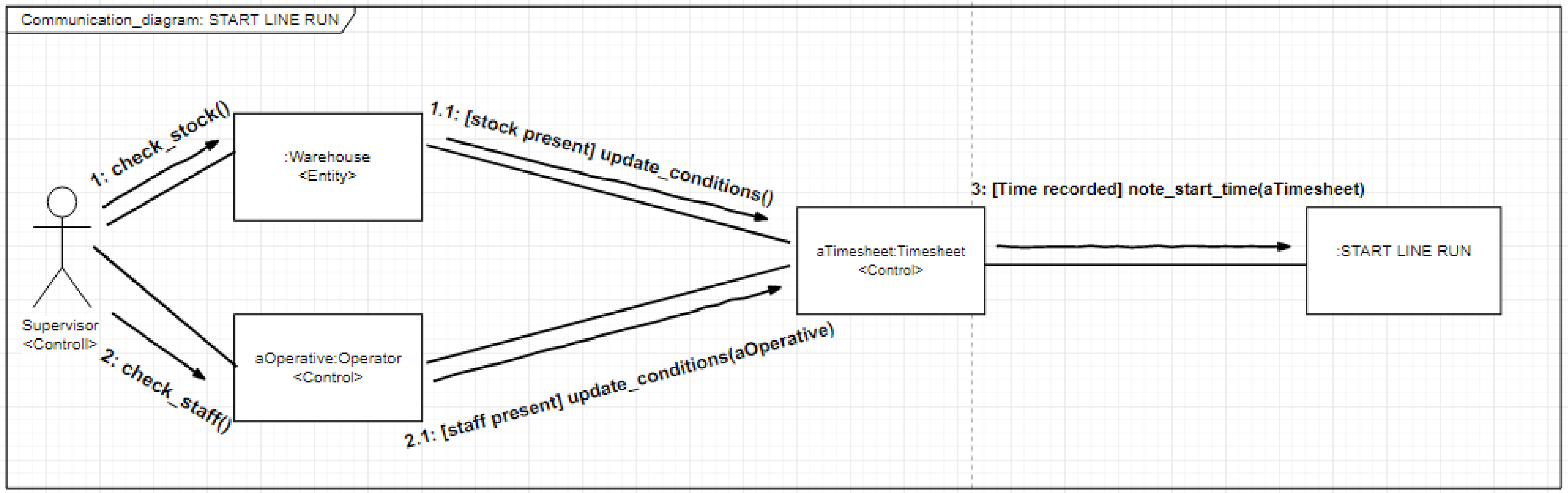
Ingredients

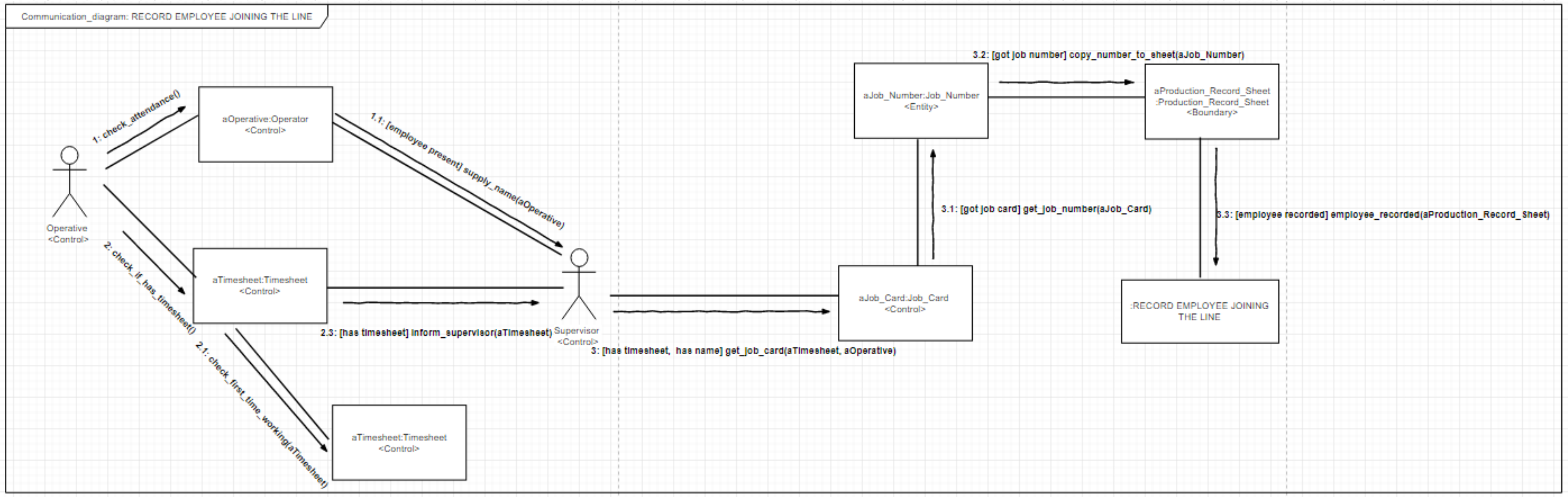
Job Number

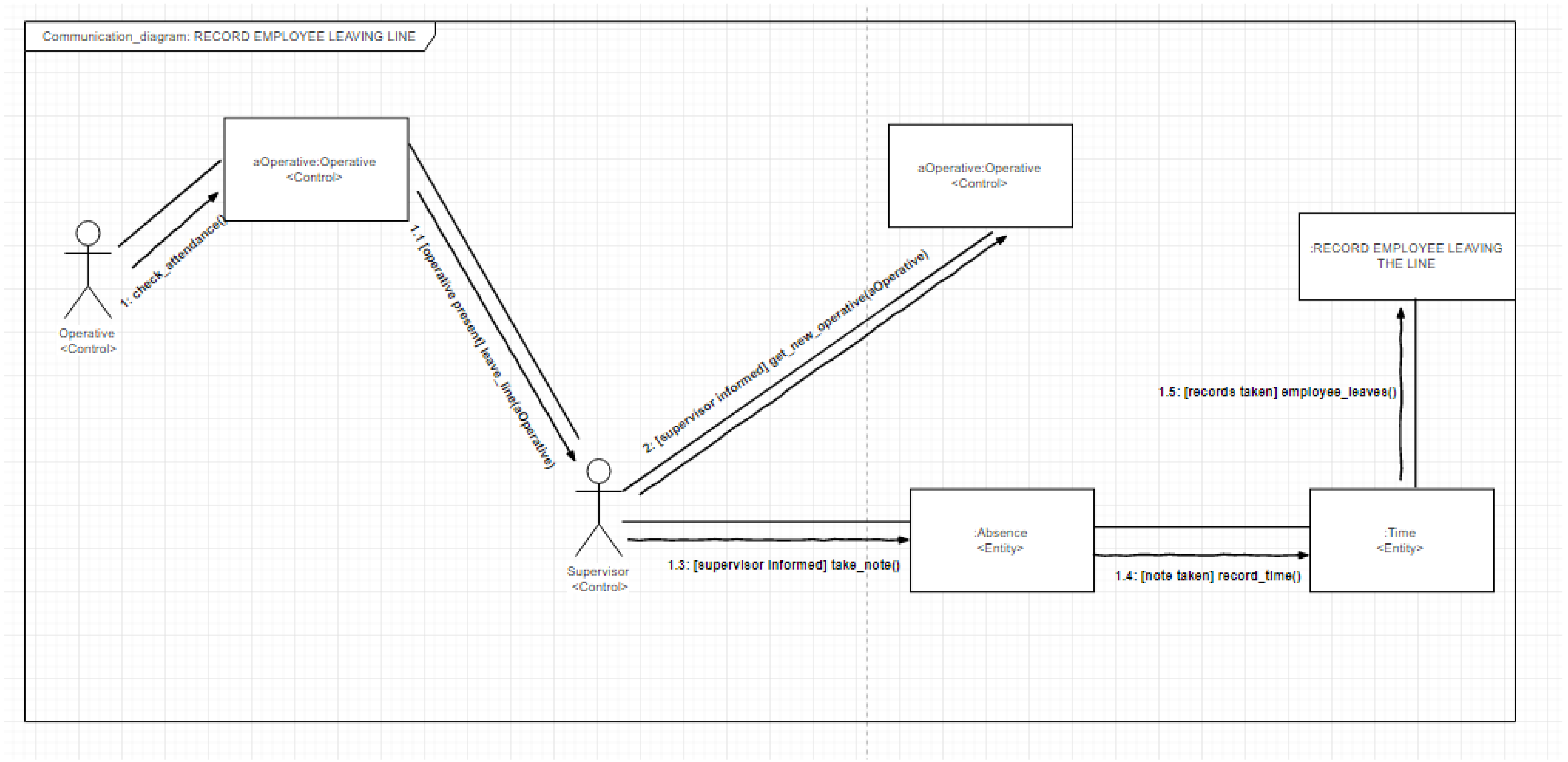
Time

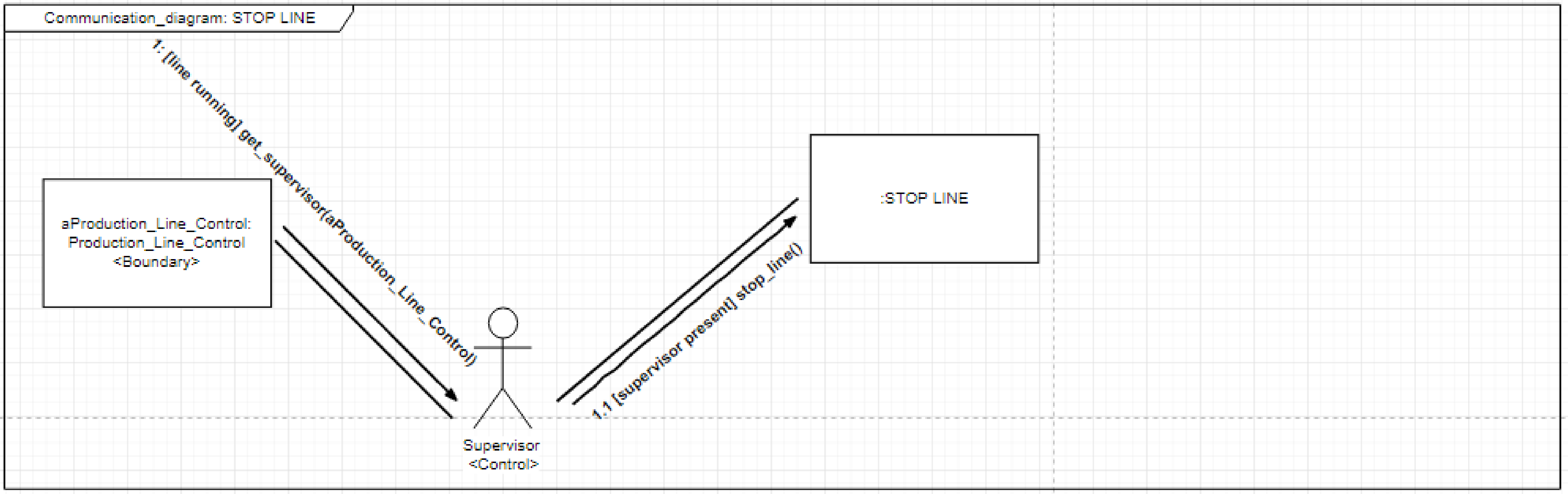
Absence

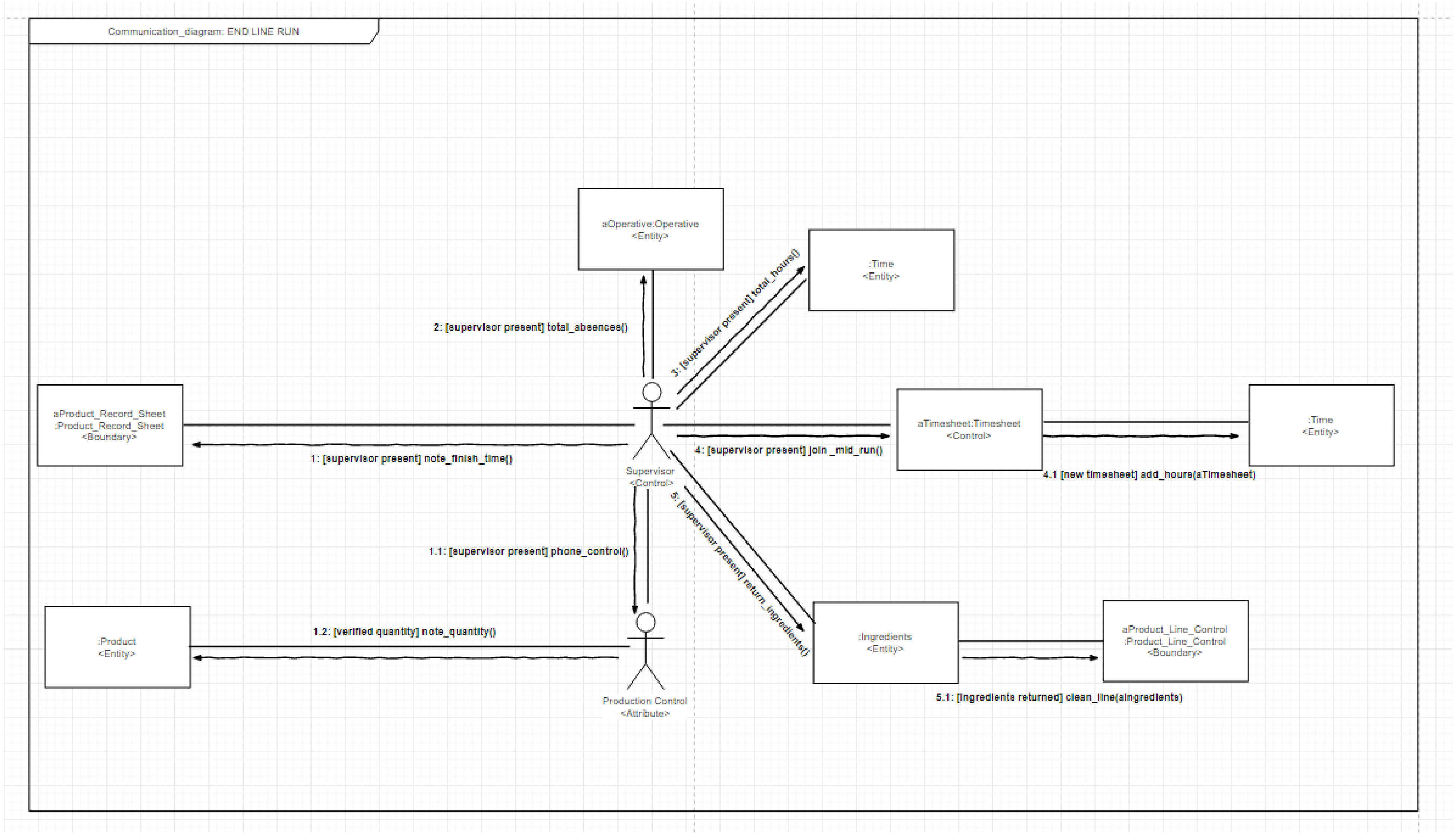
Communication Diagrams (2):

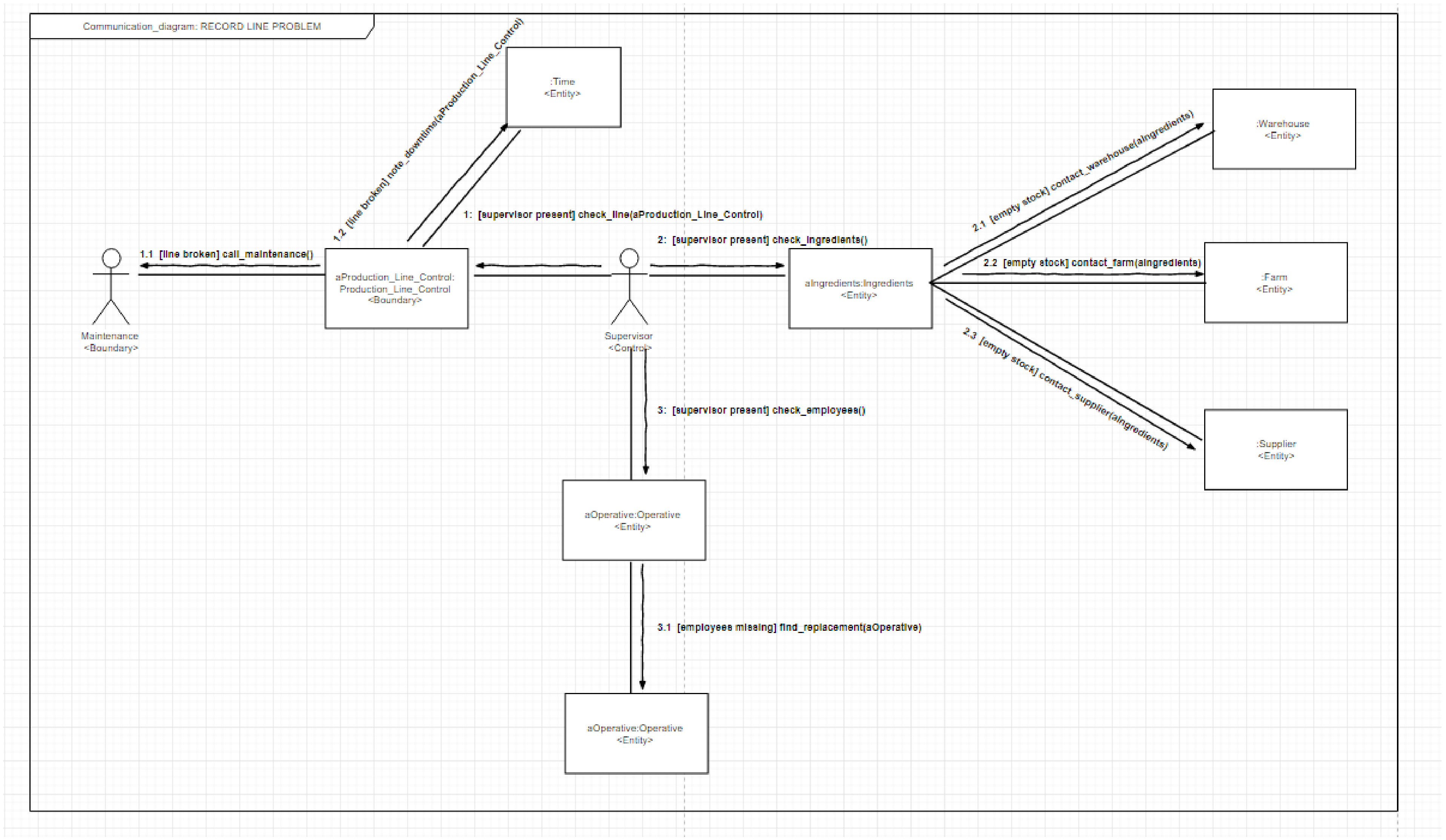

Communication_diagram: START LINE RUN
Supervisor <Controll>
1: check_stock()
:Warehouse <Entity>
2: check_staff()
aOperative:Operator <Control>
1.1: [stock present] update_conditions()
2.1: [staff present] update_conditions(aOperative)
aTimesheet:Timesheet <Control>
3: [Time recorded] note_start_time(aTimesheet)
:START LINE RUN


Communication_diagram: RECORD EMPLOYEE JOINING THE LINE
Operative <Control>
1: check_attendance()
aOperative:Operator <Control>
1.1: [employee present] supply_name(aOperative)
2: check_if_has_timesheet()
aTimesheet:Timesheet <Control>
2.1: check_first_time_working(aTimesheet)
aTimesheet:Timesheet <Control>
2.3: [has timesheet] inform_supervisor(aTimesheet)
Supervisor <Control>
3: [has timesheet, has name] get_job_card(aTimesheet, aOperative)
aJob_Card:Job_Card <Control>
3.1: [got job card] get_job_number(aJob_Card)
aJob_Number:Job_Number <Entity>
3.2: [got job number] copy_number_to_sheet(aJob_Number)
aProduction_Record_Sheet :Production_Record_Sheet <Boundary>
3.3: [employee recorded] employee_recorded(aProduction_Record_Sheet)
:RECORD EMPLOYEE JOINING THE LINE

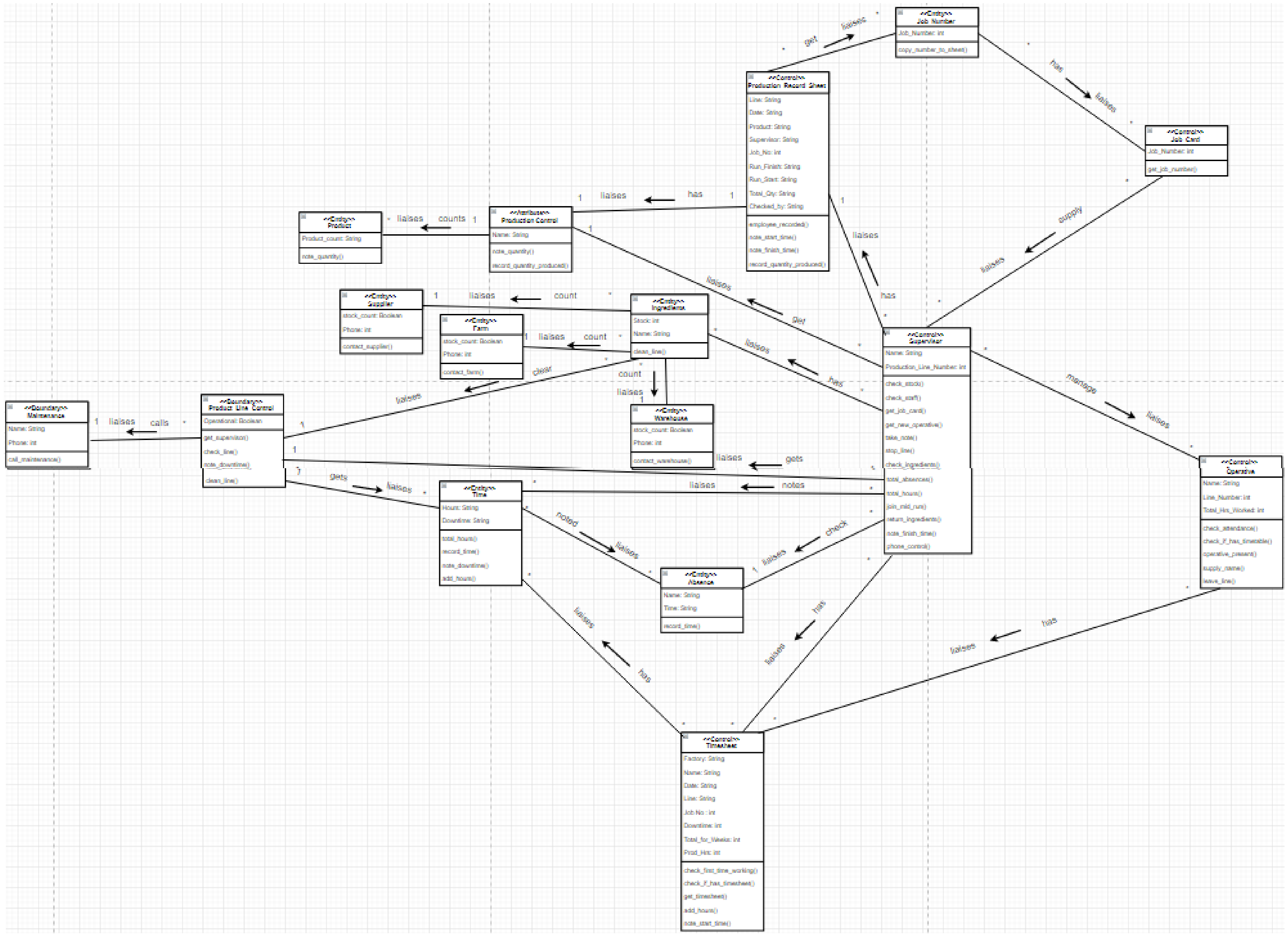

Communication_diagram: RECORD EMPLOYEE LEAVING LINE
aOperative:Operative <Control>
aOperative:Operative <Control>
:RECORD EMPLOYEE LEAVING THE LINE
Operative <Control>
1: check_attendance()
1.1 [operative present] leave_line(aOperative)
2: [supervisor informed] get_new_operative(aOperative)
1.5: [records taken] employee_leaves()
Supervisor <Control>
:Absence <Entity>
:Time <Entity>
1.3: [supervisor informed] take_note()
1.4: [note taken] record_time()


Communication_diagram: STOP LINE
1: [line running] get_supervisor(aProduction_Line_Control)
aProduction_Line_Control:
Production_Line_Control
<Boundary>
Supervisor
<Control>
1.1 [supervisor present] stop_line()
:STOP LINE


Communication_diagram: END LINE RUN
aOperative:Operative <Entity>
:Time <Entity>
2: [supervisor present] total_absences()
3: [supervisor present] total_hours()
aProduct_Record_Sheet :Product_Record_Sheet <Boundary>
Supervisor <Control>
aTimesheet:Timesheet <Control>
:Time <Entity>
1: [supervisor present] note_finish_time()
4: [supervisor present] join _mid_run()
4.1 [new timesheet] add_hours(aTimesheet)
5: [supervisor present] return_ingredients()
1.1: [supervisor present] phone_control()
:Product <Entity>
1.2: [verified quantity] note_quantity()
Production Control <Attribute>
:Ingredients <Entity>
aProduct_Line_Control :Product_Line_Control <Boundary>
5.1: [ingredients returned] clean_line(aIngredients)


Communication_diagram: RECORD LINE PROBLEM
:Time <Entity>
1.2 [line broken] note_downtime(aProduction_Line_Control)
1: [supervisor present] check_line(aProduction_Line_Control)
2: [supervisor present] check_ingredients()
1.1 [line broken] call_maintenance()
Maintenance <Boundary>
aProduction_Line_Control: Production_Line_Control <Boundary>
Supervisor <Control>
aIngredients:Ingredients <Entity>
2.1 [empty stock] contact_warehouse(aIngredients)
2.2 [empty stock] contact_farm(aIngredients)
2.3 [empty stock] contact_supplier(aIngredients)
:Warehouse <Entity>
:Farm <Entity>
:Supplier <Entity>
3: [supervisor present] check_employees()
aOperative:Operative <Entity>
3.1 [employees missing] find_replacement(aOperative)
aOperative:Operative <Entity>

Class Diagram (3):


<<Entity>>
Job Number
Job_Number: Int
copy_number_to_sheet()
<<Control>>
Production Record Sheet
Line: String
Date: String
Product: String
Supervisor: String
Job_No: Int
Run_Finish: String
Run_Start: String
Total_Qty: String
Checked_by: String
employee_recorded()
note_start_time()
note_finish_time()
record_quantity_produced()
<<Control>>
Job Card
Job_Number: Int
get_job_number()
<<Entity>>
Product
Product_count: String
note_quantity()
<<Attribute>>
Production Control
Name: String
note_quantity()
record_quantity_produced()
<<Entity>>
Supplier
stock_count: Boolean
Phone: Int
contact_supplier()
<<Entity>>
Farm
stock_count: Boolean
Phone: Int
contact_farm()
<<Entity>>
Ingredients
Stock: Int
Name: String
clean_line()
<<Entity>>
Warehouse
stock_count: Boolean
Phone: Int
contact_warehouse()
<<Boundary>>
Maintenance
Name: String
Phone: Int
call_maintenance()
<<Boundary>>
Product Line Control
Operational: Boolean
get_supervisor()
check_line()
note_downtime()
clean_line()
<<Control>>
Supervisor
Name: String
Production_Line_Number: Int
check_stock()
check_staff()
get_job_card()
get_new_operative()
take_note()
stop_line()
check_ingredients()
total_absences()
total_hours()
join_mid_run()
return_ingredients()
note_finish_time()
phone_control()
<<Control>>
Operative
Name: String
Line_Number: Int
Total_Hrs_Worked: Int
check_attendance()
check_if_has_timetable()
operative_present()
supply_name()
leave_line()
<<Entity>>
Time
Hours: String
Downtime: String
total_hours()
record_time()
note_downtime()
add_hours()
<<Entity>>
Absence
Name: String
Time: String
record_time()
<<Control>>
Timesheet
Factory: String
Name: String
Date: String
Line: String
Job No : Int
Downtime: Int
Total_for_Weeks: Int
Prod_Hrs: Int
check_first_time_working()
check_if_has_timesheet()
get_timesheet()
add_hours()
note_start_time()
get
liaises
has
liaises
supply
liaises
has
liaises
liaises
has
1
1
liaises
counts
1
liaises
day
liaises
has
manage
liaises
1
liaises
count
1
liaises
count
clear
liaises
count
liaises
1
1
liaises
calls
liaises
gets
liaises
notes
gets
liaises
noted
liaises
check
liaises
liaises
has
has
liaises
liaises
has

Software Engineering reflection:

For the STOPLINE requirement in the transcript, there was no explicit way to stop the line, so there was no exact specification on how I documented stopping the line in terms of the implementation of a class; it just stops without running past any conditions.

I also documented each precondition in the diagram, as I felt these were an integral part on how the system ran each subsection, (eg: STOPLINE, RECORD EMPLOYEE JOINING THE LINE etc.) even though it wasn't necessarily part of the original requirements.

My Communication and Class diagrams contain, I feel, too much detail; I captured more than the core requirements of the system. However, even though there is too much detail, I feel that my diagrams are consistent in terms of the relations, content, attributes and methods.