SE-2 COURSE PROJECT (PHASE 2 COVER SHEET)

Discussions Scheduled for Week 12 *(Specific dates TBA by the TAs).*

- Print 1 copy of this cover sheet and attach it to a printed copy of the documentation *(SRS, … etc.)*. You must submit softcopies of all your documents *(as PDFs)*; details will be announced later.
- Please write all your names in Arabic.
- Please make sure that your students' IDs are correct.
- Handwritten Signatures for the attendance of all team members should be filled in before the discussion.
- Please attend the discussion on time *(announced separately)*, late teams will lose 3 grades.

Project Name: "An Autonomous Robotic Fruit/Vegetable Grading & Sorting System"

Team Information *(typed not handwritten, except for the attendance signature)*:

	ID [Ordered by ID]	Full Name [In Arabic][i]	Attendance [Handwritten Signature]	Final Grade
1	20208052	أية احمد السيد أحمد		
2	20208038	آمنة مصطفي سعد		
3	20208070	بسمة محمد فتحي محمد		
4	20208254	ندي عصام محمد عبدالعزيز		
5	20208293	ياسمين صفوت عبدالرحمن		
6				

Grading Criteria:

Items		Grade	Notes
Functional Requirements & Non-Functional Requirements – *including any updates, and all timing constraints.*	1		
Bonus: System Architecture – *including any applied Architectural Pattern(s).*	1		
Use-Case Diagram(s) – *including all use-cases for the system, and the detailed use-cases description, and any alternative scenarios.*	1		

Sequence Diagram(s) – *including varying fragments, interaction references/gates, different types of messages & constraints, .. etc.*	**2**		
Collaboration/Communication Diagram(s) – *including different types of messages, and the objects must have stereotypes indicating their categories based on the given class/object structuring criteria.*	**1**		
State-Machine Diagrams – *for all state dependent objects, and for the entire system too, including "when necessary" Events/Actions, Guards, Entry and Exit events/actions, Composite and Orthogonal states, Submachines, History States.. etc.*	**2**		
Bonus: Object Diagrams – *including object diagrams that illustrate the preconditions and the post-conditions of selected functions.*	**1**		
Bonus: 2 Design Patterns Applied – *Including a typed description of the pattern and how is it applied.*	**1**		
Detailed Class diagram – *including "when necessary" Classes, Attributes & Methods, Interfaces & Abstract Classes, Associations / Aggregations / Generalizations / Association Classes / Qualified Associations, Constraints - including also the categories of the classes based on the given class/object structuring criteria, and stereotypes indicating the type/category of each class. All necessary types/categories should be modelled.*	**2**		
Stimuli/Response Identification (State Transition Table)	**1**		
Implementation & discussion. Marking the code will be based on the following criteria: **1) Requirements are fulfilled.** **2) Correctly mapping design models into executable code.** **3) Running correctly.** **4) Detailed Testing.** **5) Correct multithreading implementation and synchronization.**	**5**		

N.B. I .. **<u>You must update and resubmit the initial part of the documentation submitted in phase 1</u>** (including the Functional / Non-Functional requirements, Use-case Diagrams & Descriptions, Activity Diagrams, Interaction Diagrams, Object Diagrams, .. etc.).

Teaching-Assistant's Signature: ________________________

____
15

Functional Requirements & Non-Functional Requirements

Functional requirements:

- The administrator must only add crops.
- Providing robots scan the field.
- Providing robots should move if it can't find the ripened crops and scan the field again.
- Providing robots must suck the ripened crops.
- Sorting robots sort the ripened crops to damaged and undamaged.
- Grading robots check if the crop is organic or not.
- Grading robots give the crop two scores if it is organic, otherwise it will have a score of zero.
- Grading robots check if the crop has insect bite and fungi or not.
- Grading robots give the crop two scores if it is organic, otherwise it will have a score of zero.

- Grading robots evaluate the crops based on their degree of maturity, organic matter, and the absence of insects and fungi in them.
- Grading robot give a rate for each crop between (A-B-C-D) Based on their quality after grading.

Non Functional requirements:

- The view makes system easy to use, users can easily navigate its interface and they can understand how the application organize its content (**Usability**)
- The system has features that match the geographical location of its users including: languages, currencies, interests, purposes and needed.(**Localization**)
- The percentage of the time that the system is accessible for users and operation is very high, the system may be available 85% of the time during month. (**Availability**)
- The percentage of the probability of failure is low, system functions normally most of the time. (**Reliability**)
- The system is quite fast, the system not take much time to return the results .(**Performance**)

Timing Constraints in Real-time System

1. Performance Constraints:

-The system is quite fast.

-The system not take much time to return the results.

- For the update app, the upper limit of the time constraint is 23602 ms and the lower limit is 5997 ms

-For the dashboard app, the upper limit of the time constraint is 12236 ms and the lower limit is 7540 ms

-This is fairly quick, and appropriately suitable to the system

2. Behavioral Constraint:

-Environment of a system is well behaved.

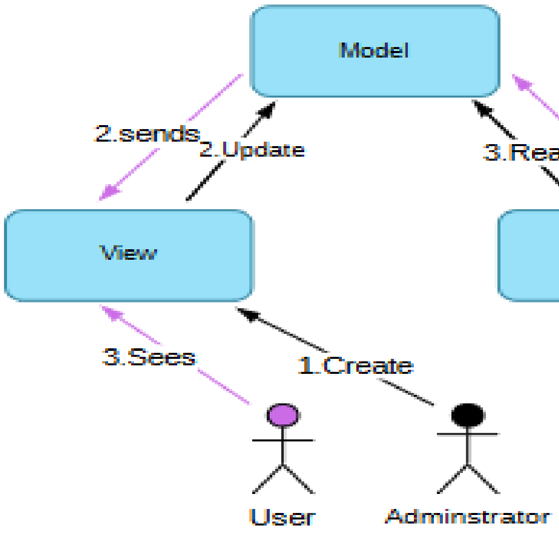
-system functions normally most of the time.

-System can respond efficiently to the actions of the users under certain workload.

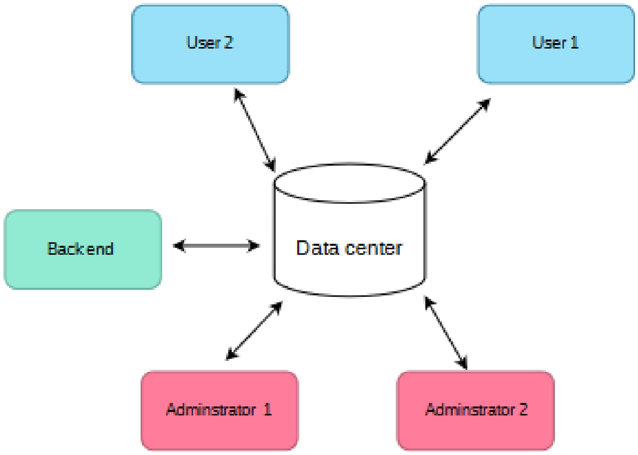
-user experience is not affected by any latency issues.

System Architecture

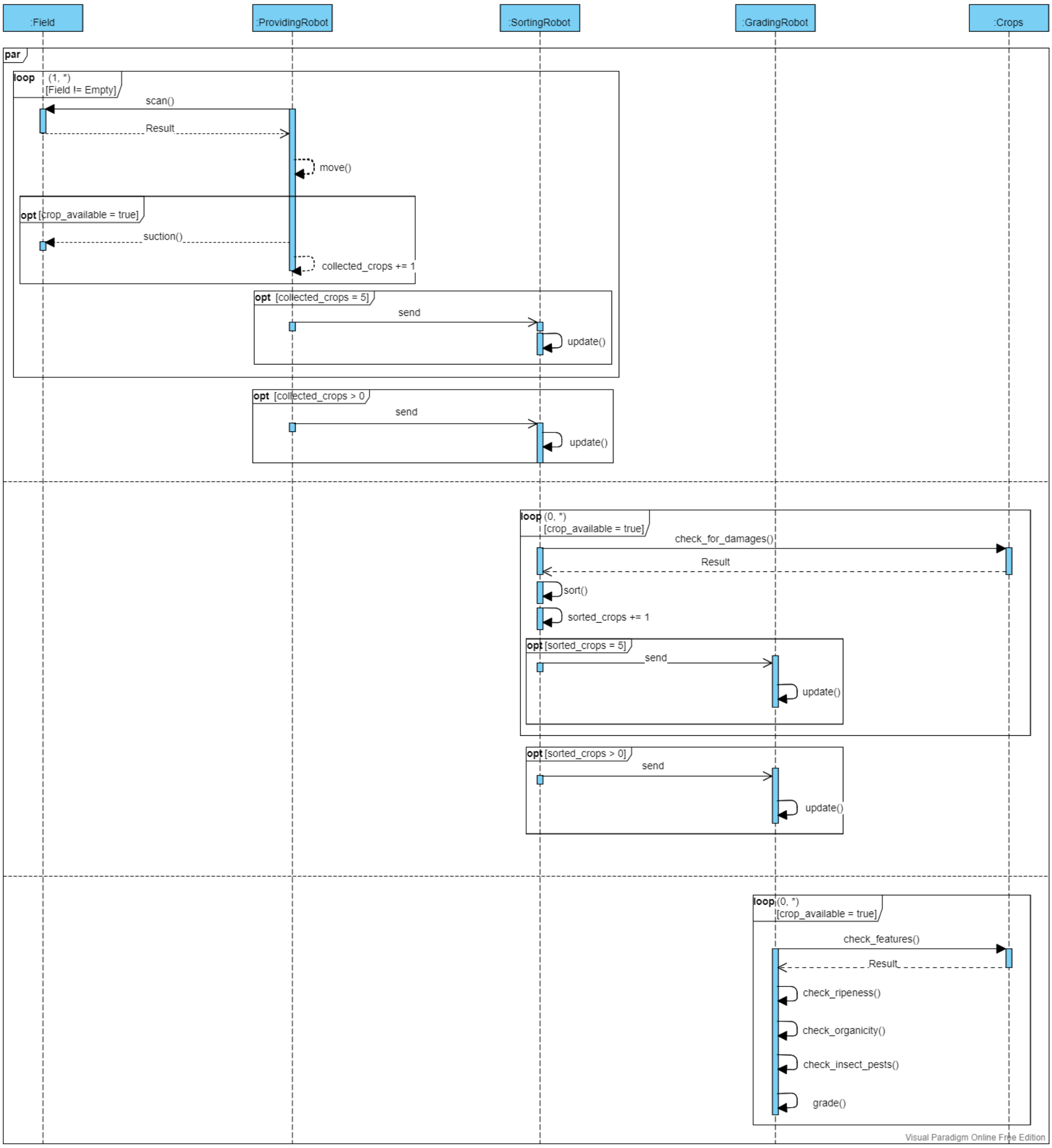
1- MVC



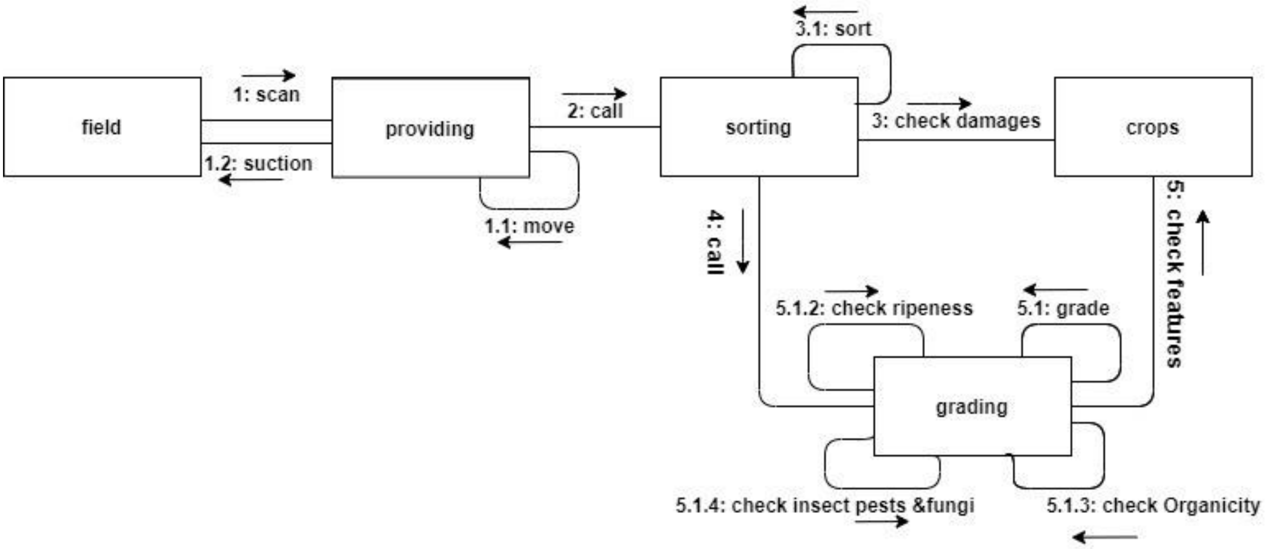
2- Data center

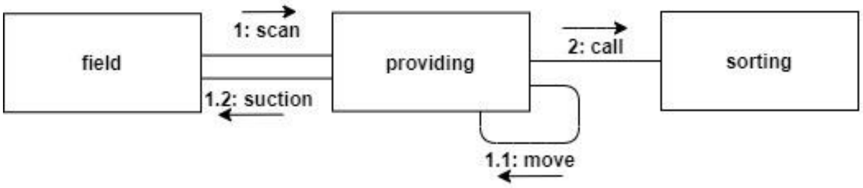


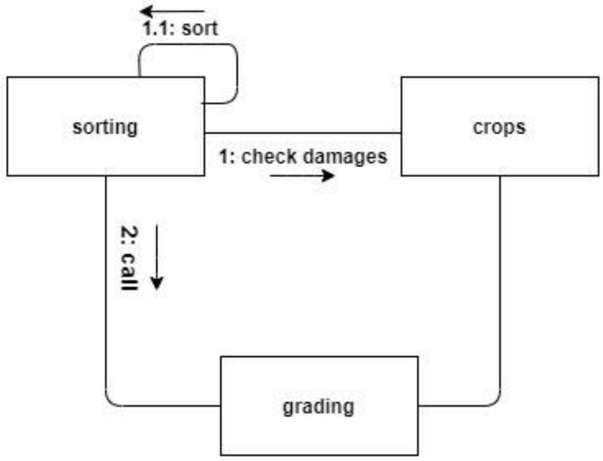
Sequence Diagram

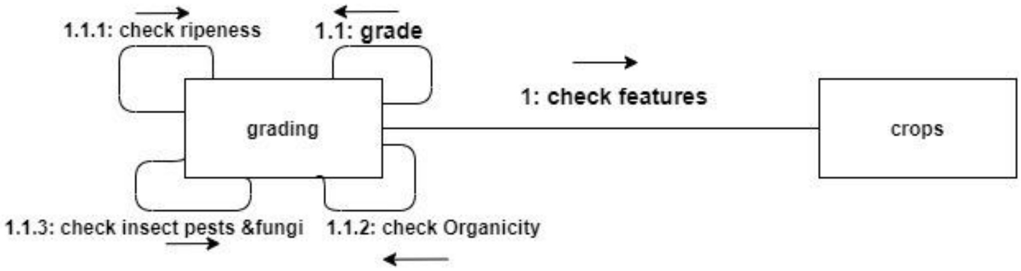


Collaboration/Communication Diagram

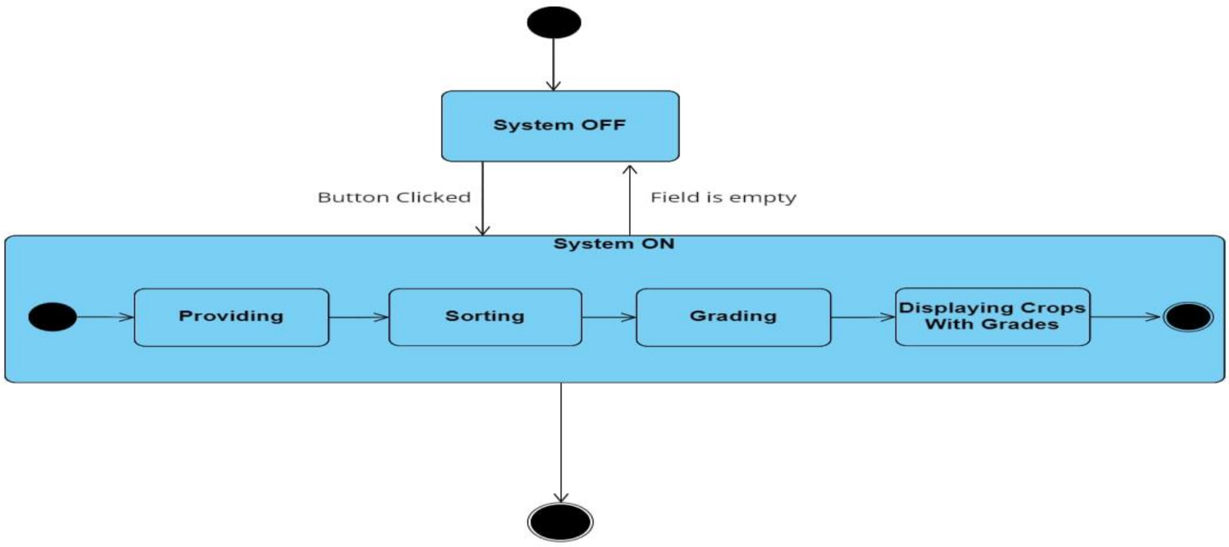


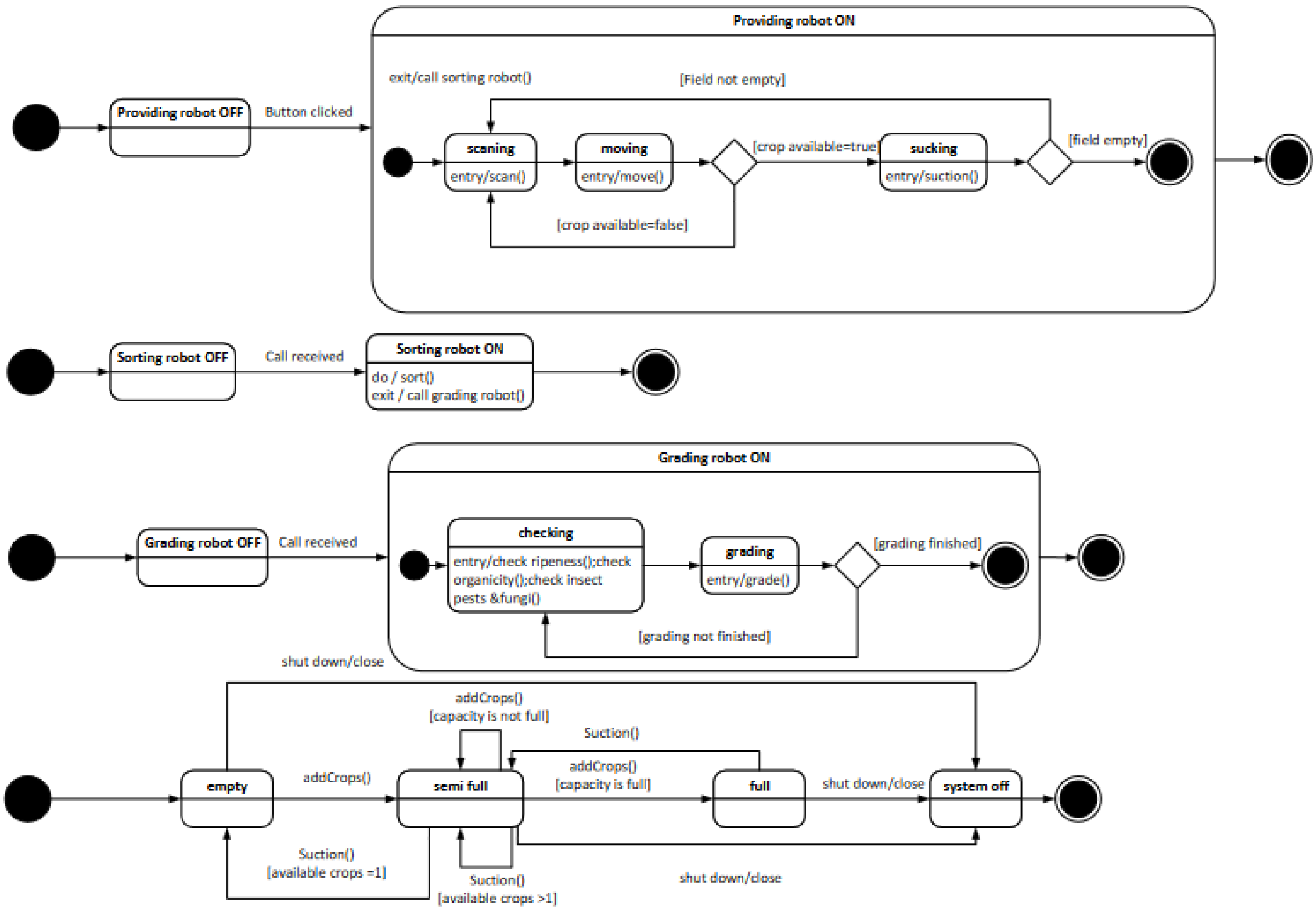




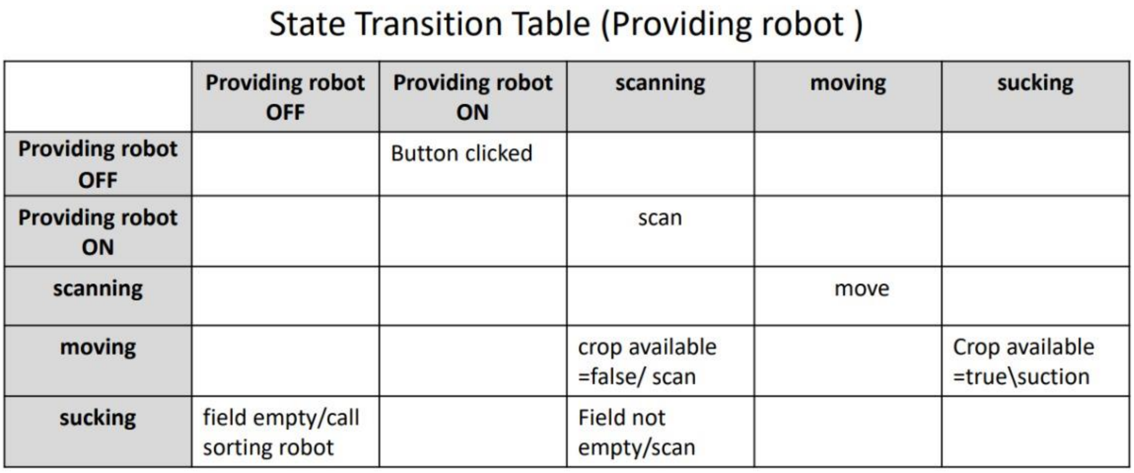


State-Machine Diagrams



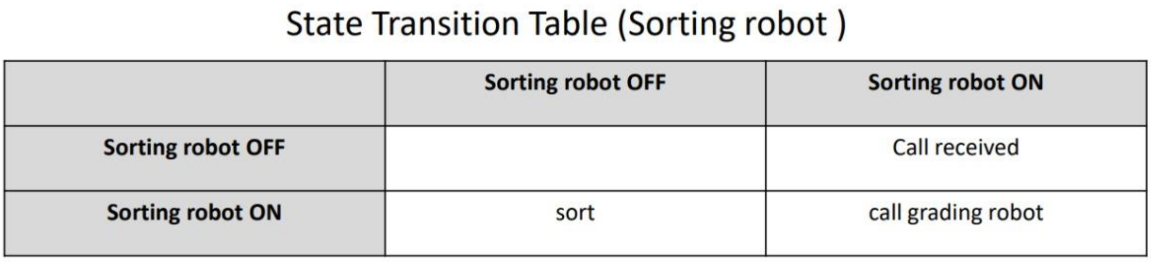


Stimuli/Response Identification (State Transition Table)



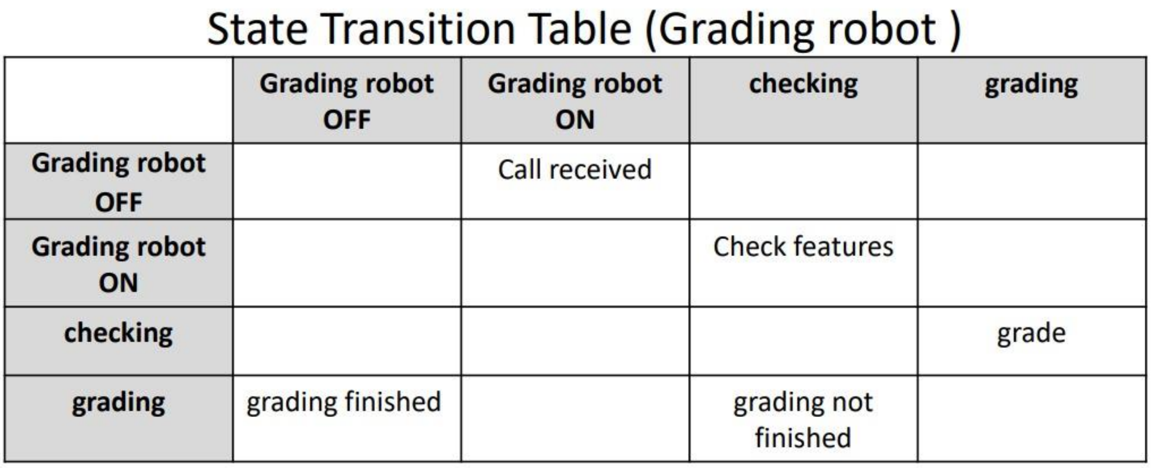
State Transition Table (Providing robot)

	Providing robot OFF	Providing robot ON	scanning	moving	sucking
Providing robot OFF		Button clicked			
Providing robot ON			scan		
scanning				move	
moving			crop available =false/ scan		Crop available =true\suction
sucking	field empty/call sorting robot		Field not empty/scan		



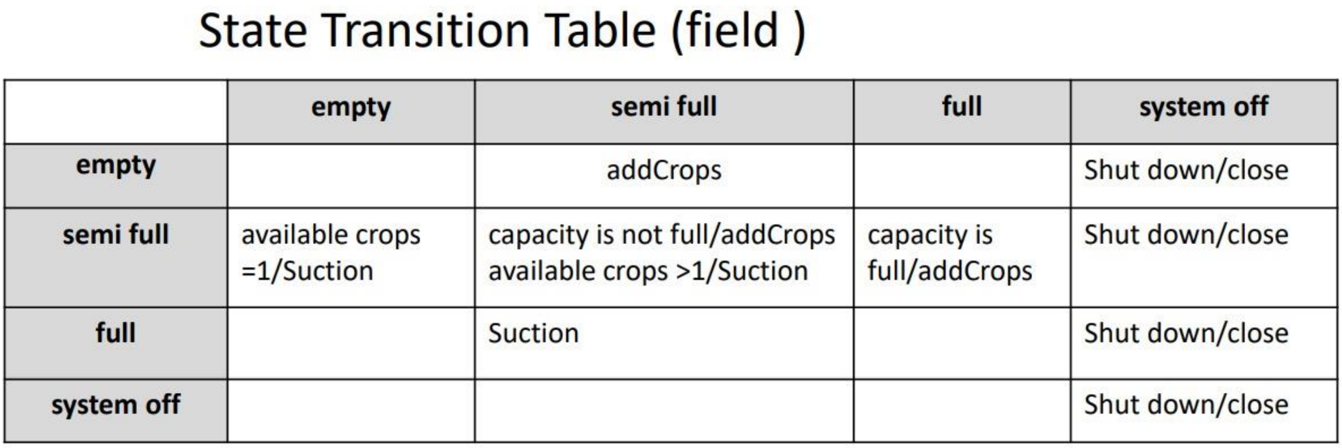
State Transition Table (Sorting robot)

	Sorting robot OFF	Sorting robot ON
Sorting robot OFF		Call received
Sorting robot ON	sort	call grading robot



State Transition Table (Grading robot)

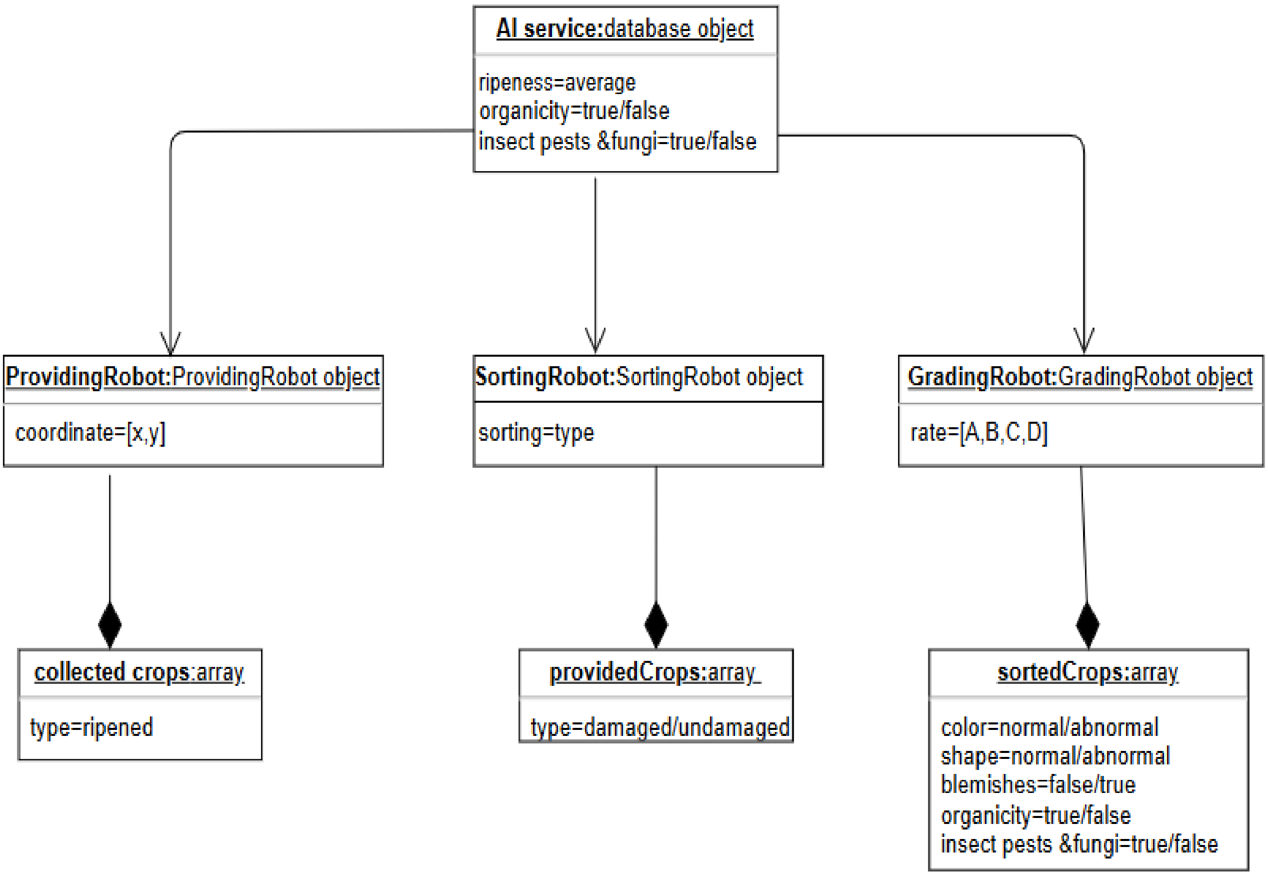
	Grading robot OFF	Grading robot ON	checking	grading
Grading robot OFF		Call received		
Grading robot ON			Check features	
checking				grade
grading	grading finished		grading not finished	



State Transition Table (field)

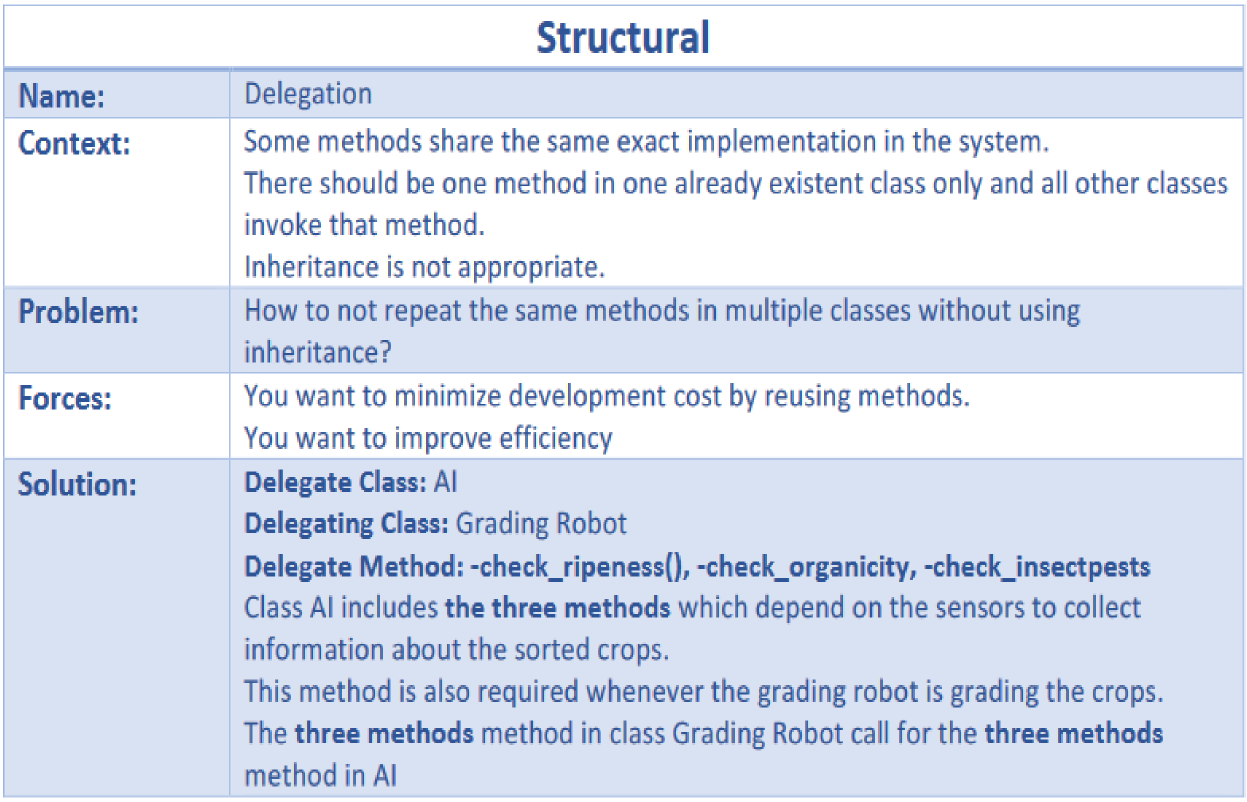
	empty	semi full	full	system off
empty		addCrops		Shut down/close
semi full	available crops =1/Suction	capacity is not full/addCrops available crops >1/Suction	capacity is full/addCrops	Shut down/close
full		Suction		Shut down/close
system off				Shut down/close

Object Diagrams

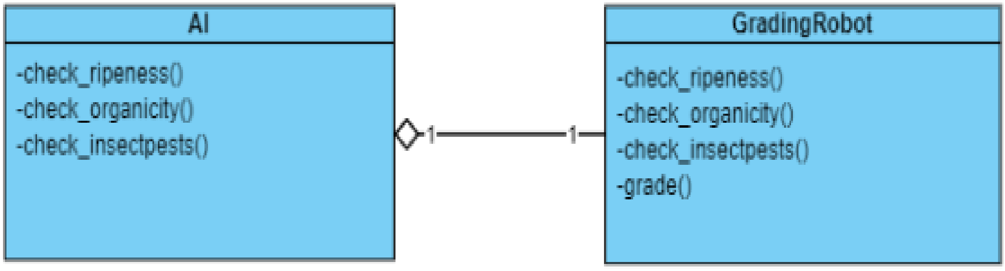


Design Patterns Applied

1-Delegation

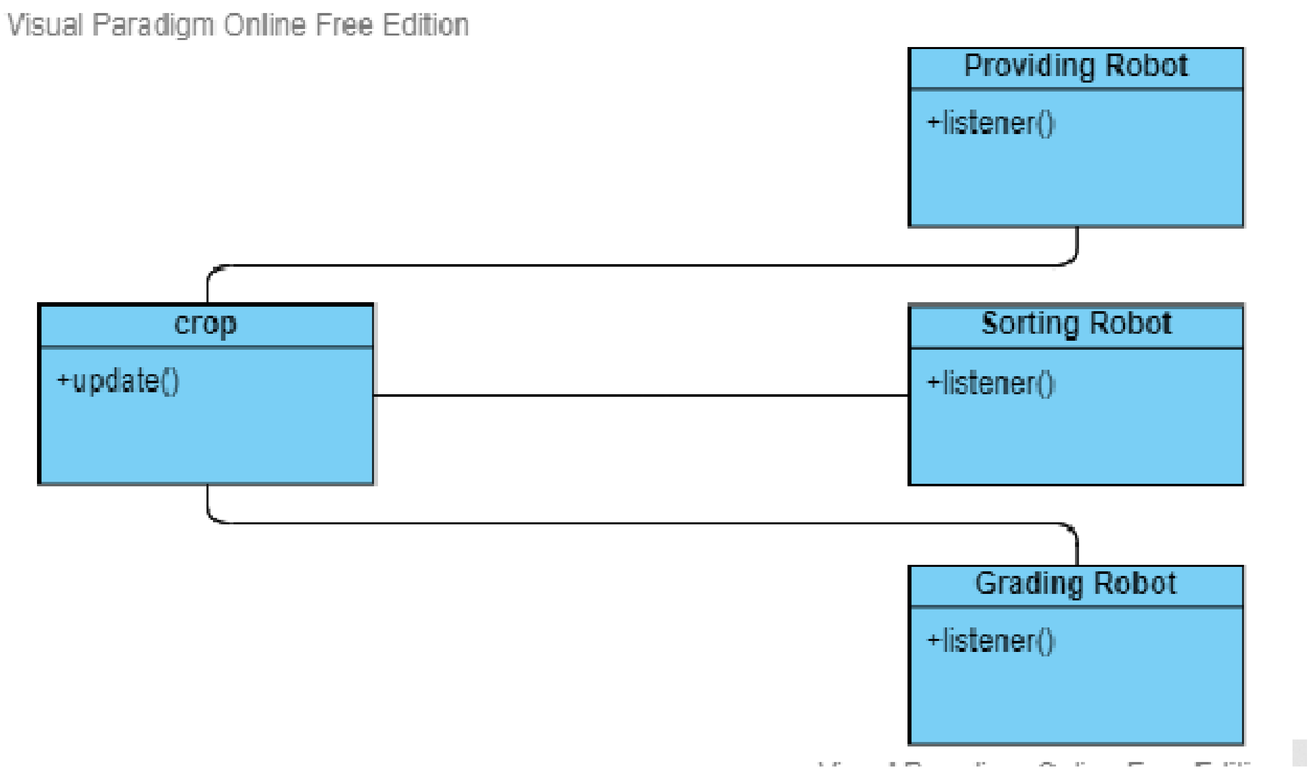


Structural	
Name:	Delegation
Context:	Some methods share the same exact implementation in the system. There should be one method in one already existent class only and all other classes invoke that method. Inheritance is not appropriate.
Problem:	How to not repeat the same methods in multiple classes without using inheritance?
Forces:	You want to minimize development cost by reusing methods. You want to improve efficiency
Solution:	**Delegate Class:** AI **Delegating Class:** Grading Robot **Delegate Method: -check_ripeness(), -check_organicity, -check_insectpests** Class AI includes **the three methods** which depend on the sensors to collect information about the sorted crops. This method is also required whenever the grading robot is grading the crops. The **three methods** method in class Grading Robot call for the **three methods** method in AI

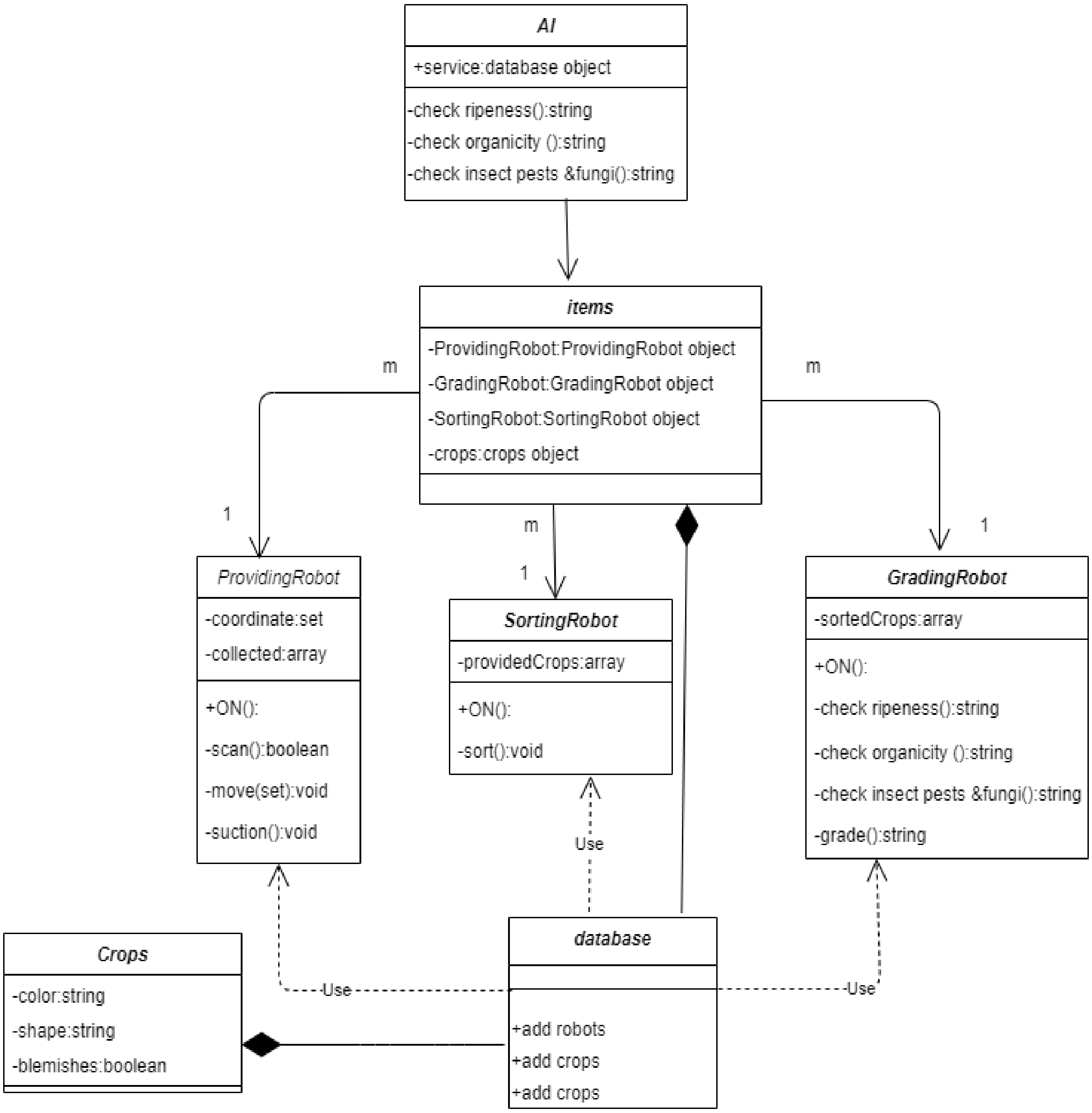


2-observer

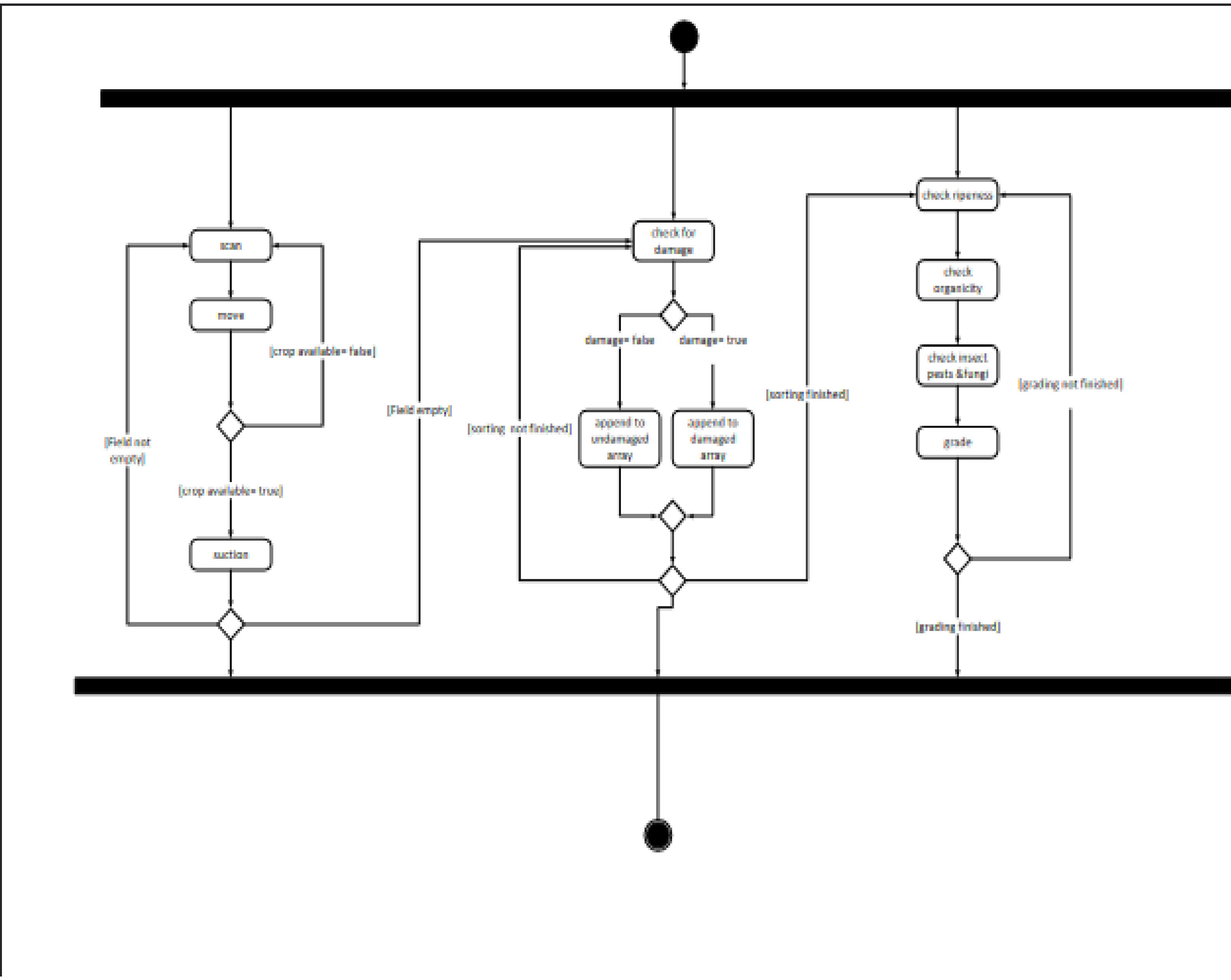
Behavioral	
Name:	Observer
Context:	An observer registers to receive notifications whenever the state of an observable (a.k.a. subject) changes. When partitioning a system into individual classes you want the coupling between then to be loose so you have the flexibility to vary them independently.
Problem:	A mechanism is needed to ensure that when the state of the class crop changes The three robots are updated to keep them in step.
Forces:	The different parts of a system have to kept in step with one another without being too tightly coupled.
Solution:	**This diagram depends on push model.** **Observer:** Providing Robot, Sorting Robot and Grading Robot **Observable:** Crop. The Admin adds a new crop and the crop data is updated in the system. The 3 listeners on the 3 robots are all updated with the new crop.



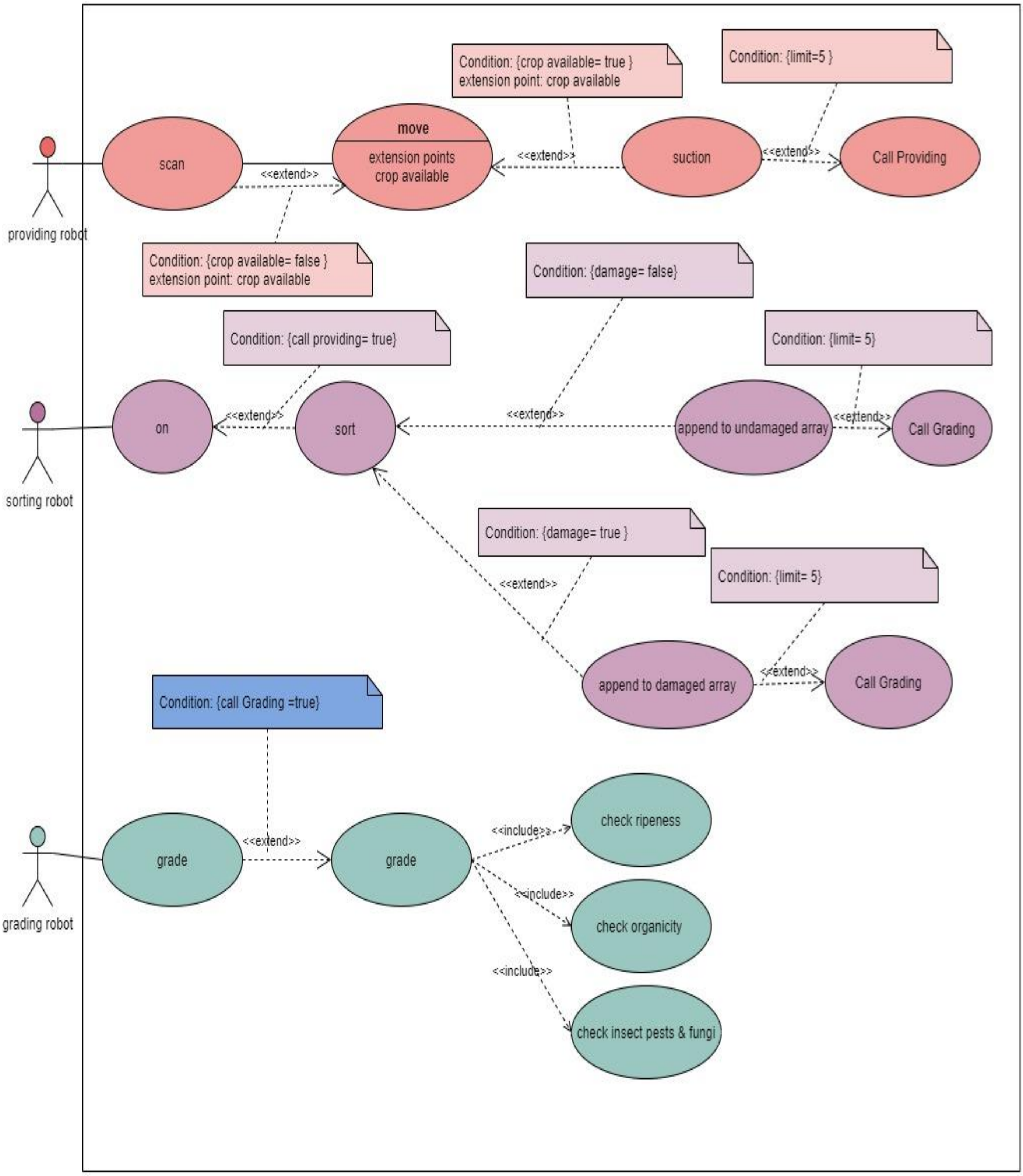
Detailed Class diagram



Activity Diagram



Use-Case Diagram



Use-Case Description:

Identifier	UC
Name	An Autonomous Robotic Fruit/Vegetable Grading & Sorting System
Initiator	Providing robot/Sorting robot/grading robot
Pre-conditions	Item created/robot on/existing crop
Post-Conditions	Graded crops /robot off
Main success scenario	1.open the dashboard page ,enters create page foe adding items(crops),access firebase 2.retrive data from database to robots home page , grading the items 3.show graded items with grade(A,B,C,D)
Goal	The user accesses the Robotic Fruit/Vegetable Grading & Sorting System to be able to make the relevant functions according to the robots role.