Five Guys

u24569772 Josua Louw, u14439141 Frikkie Malan, u24586189 Darius Erasmus, u23565722 Nico Theron, u24711358 Heindrich Jansen

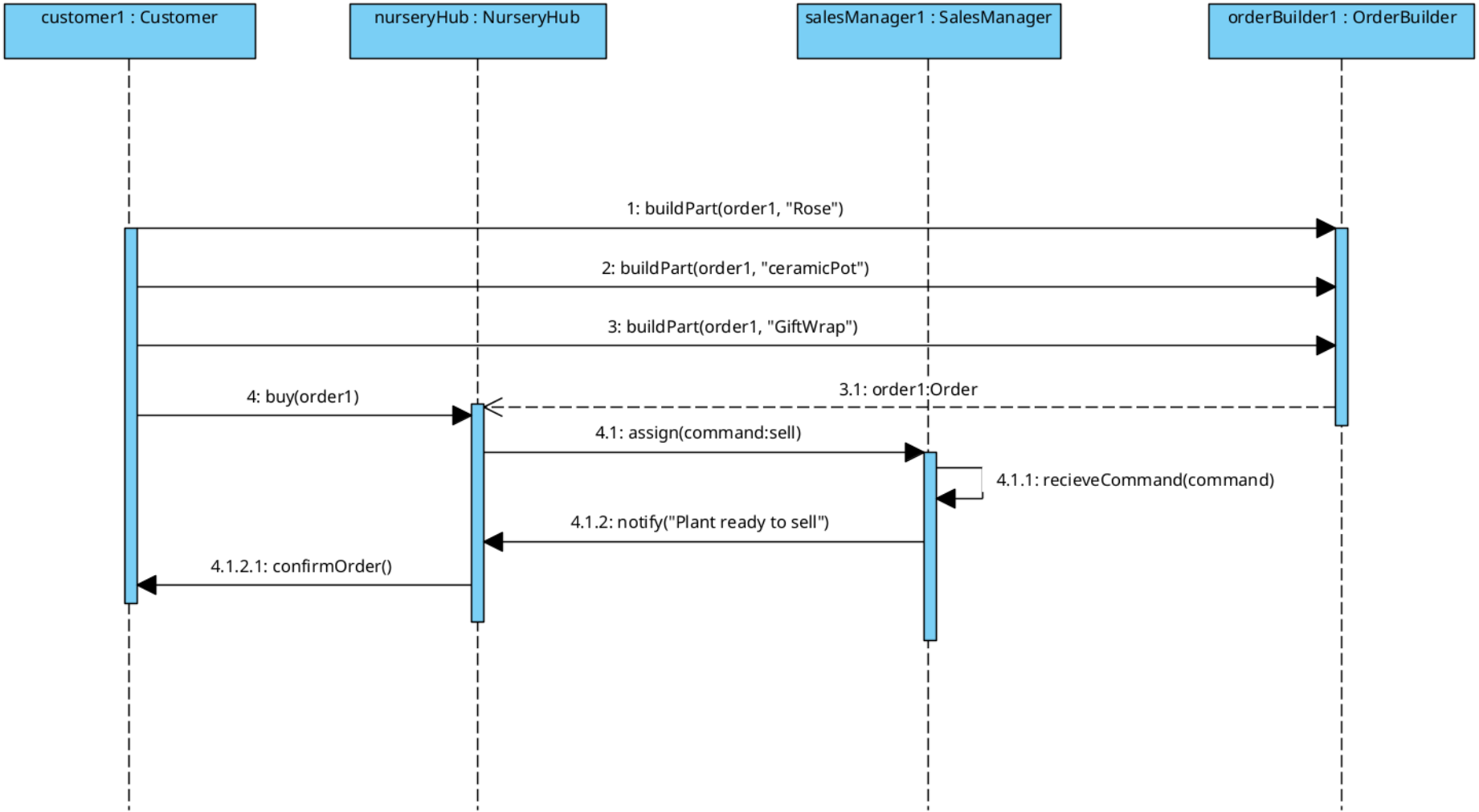
Functional requirements

- FR1: Plant Collection Management (Composite Pattern)
  The system shall allow greenhouses to manage collections of plants, treating individual plants and groups of plants uniformly. It does this by dividing the greenhouse into subsections.
- FR2: Plant Inventory Iteration (Iterator Pattern)
  The system shall provide a mechanism to iterate through the plant inventory in a greenhouse, enabling staff to access and display plants without exposing the underlying collection structure.
- FR3: Plant Care Strategy (Strategy Pattern)
  The system shall apply different care strategies based on the type of plant, allowing dynamic switching of care routines.
- FR4: Plant Life Cycle Management (State Pattern)
  The system shall track and update the life cycle stages of plants, supporting transitions between stages based on care and time events.
- FR5: Staff Task Commands (Command Pattern)
  The system shall allow staff to receive and execute commands for tasks like preparing plants for sale, with the ability to store and process commands in a task list.
- FR6: Staff Task Delegation (Chain of Responsibility Pattern)
  The system shall delegate tasks to available staff members in a chain, where a task is passed to the next available staff member if the current staff is busy.
- FR7: Staff-Customer Coordination (Mediator Pattern)
  The system shall enable NurseryHub to mediate communication between customers, staff, and plant inventory, coordinating actions like order processing and plant preparation.
- FR8: Custom Order Construction (Builder Pattern)
  The system shall allow customers to build custom orders step-by-step (e.g., adding plant, pot, decorations) using specialized builders (AddPlant, AddPot, AddDecoration, AddSeed) that validate and construct the order.
- FR9: Plant Cloning (Prototype Pattern)
  The system shall create clones of existing plants with their state reset to seedling, enabling rapid propagation of new plants.
- FR10: Order Customization (Decorator Pattern)
  The system shall allow customers to customize orders by adding components (e.g., plant, pot, decorations) in a linked list structure, calculating the total price dynamically.

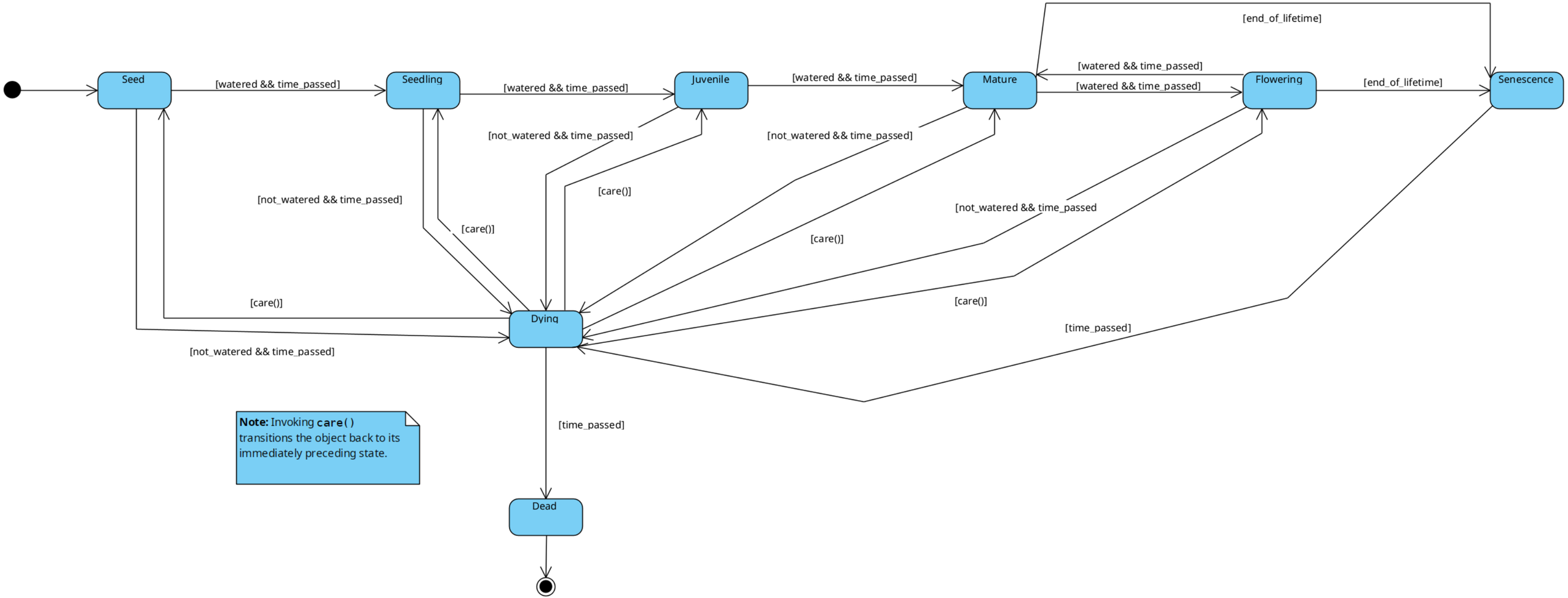
Non-Functional requirements

- NFR1: Scalability
  The system shall support up to 50 simultaneous users (e.g., manager, staff, customers) interacting with the nursery (e.g., browsing inventory, processing orders) without performance degradation, ensuring uptime remains above 99%.
- NFR2: Maintainability
  The system shall be designed with modular components (e.g., separate classes for plants, greenhouses, staff) and clear documentation, allowing developers to add new plant types or staff roles in under 2 hours of development time.
- NFR3: Usability
  The system shall provide a text-based interface that enables users (manager, staff, customers) to perform key tasks (e.g., ordering plants, managing inventory) with no more than 3 interactions per task, ensuring intuitive navigation.

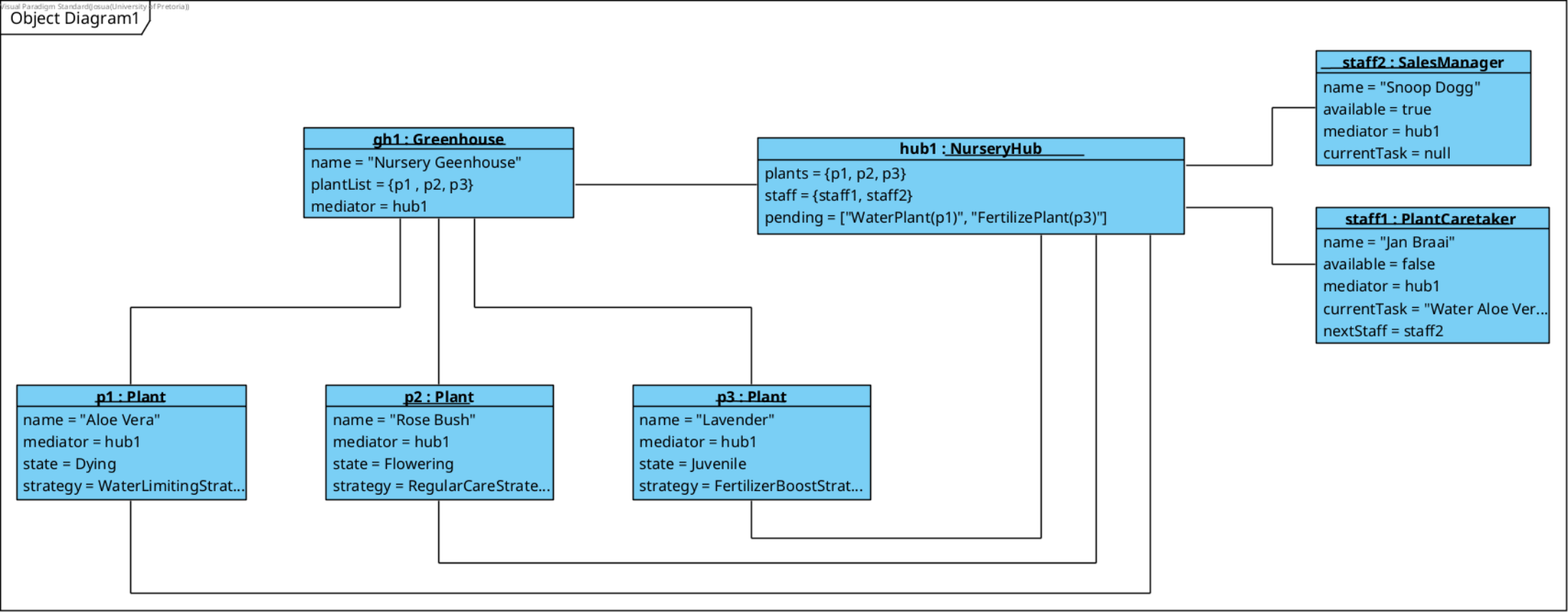
sd [Sequence Diagram1]



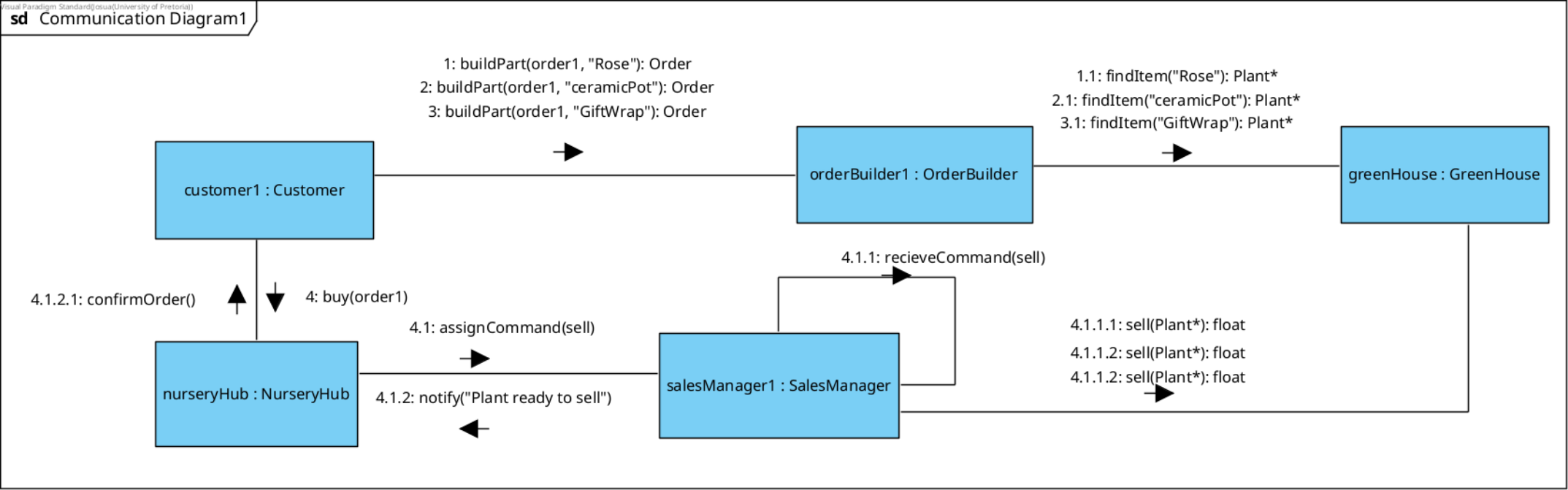
stm [Plant_StateDiagram]



Object Diagram1



sd Communication Diagram1



Class Diagram1

SellCommand
+execute() : void

Chain of Responsibility design pattern: Used to distribute commands / work between staff members.

Person

Customer
-orderBuilder : OrderBuilder*
-nurseryHub : NurseryMediator*
+buy() : void

OrderBuilder
#greenHouse : GreenHouse*
+buildPart(order : Order, itemName : string) : Order**
#checkType(plant : Plant) : boolean*

AddDecoration
+buildPart(order : Order*, itemName : string) : Order*
#checkType(plant : Plant*) : boolean

AddPot
+buildPart(order : Order*, itemName : string) : Order*
#checkType(plant : Plant*) : boolean

AddPlant
+buildPart(order : Order*, itemName : string) : Order*
#checkType(plant : Plant*) : boolean

AddSeed
+buildPart(order : Order*, itemName : string) : Order*
#checkType(plant : Plant*) : boolean

Template method design pattern: Used to have subparts of the building process change depending on the builder.

Staff
-nextStaff : Staff
-taskList : vector<Command*>
-nurseryHub : NurseryMediator*
+receiveCommand(command : Command) : void*

Command
+execute() : void

Client

Command design pattern: Used to tell what a staff member needs to do at the Nursery

SalesManager
+receiveCommand(command : Command*) : void

PlantCaretaker
+receiveCommand(command : Command*) : void

WaterPlant
+execute() : void

FertilizePlant
+execute() : void

NurseryMediator
+notify(sender : void, event : string, data : any) : void
+assign(cmd : Command) : void
+registerPlant(p : Plant) : void*
+registerStaff(s : Staff) : void*

Order
-plants : vector<Plant*>
+getTotalCost() : double
+addPlant(plant : Plant*) : void

Builder design pattern: Used to build an order for a customer to buy plants.

Mediator design pattern: Used to communicate plant needs to staff members to care the plants. As well as between staff members and customers.

NurseryHub
-plants : vector<Plant*>
-staff : vector<Staff*>
-customers : vector<customer*>
+assign(cmd : Command) : void
+notify(sender : void, event : string, data : any) : void
+registerPlant(p : Plant*) : void
+registerStaff(s : Staff*) : void

FertilizerBoostStrategy
+applyCare() : void
+getStrategyName() : void

Strategy design pattern: Used to decide what care strategy a plant needs.

RegularCareStrategy
+applyCare() : void
+getStrategyName() : void

WaterLimitingStrategy
+applyCare() : void
+getStrategyName() : void

CareStrategy
+applyCare() : void
+getStrategyName() : void

Plant
-strategy : CareStrategy*
-state : PlantState*
-name : string
+water() : void
+feed() : void
+getPrice() : double
*+clone() : Plant**
+getName() : string
+getType() : string

GreenHouse
-nurseryHub : NurseryMediator*
+expand(greenHouse : GreenHouse) : void*
+sell(Item : GreenHouse) : double*
*+getSubsection(sectionName : string) : GreenHouse**
*+CreateIterator() : Iterator**
*+findItem(itemName : string) : Plant**

Iterator
+first() : void
+next() : void
+isDone() : boolean
*+currentItem() : Plant**

Iterator design pattern: Used to traverse the greenhouse and see each plant.

PlantState
+handleCare() : void
+transitionToNext() : void

SeedlingState
+handleCare() : void
+transitionToNext() : void

Section
-Subsection : vector<GreenHouse>
+expand(greenHouse : GreenHouse*) : void
+getSubsection(sectionName : string) : GreenHouse*
+sell(Item : GreenHouse*) : double
+CreateIterator() : Iterator*
+findItem(itemName : string) : Plant*

<<instantiates>>

PlantIterator
+first() : void
+next() : void
+isDone() : boolean
+currentItem() : Plant*

Composite design pattern: Used to build the greenhouse with plants and subsections.

SeedState
+handleCare() : void
+transitionToNext() : void

FloweringState
+handleCare() : void
+transitionToNext() : void

SenescenceState
+handleCare() : void
+transitionToNext() : void

DeadState
+handleCare() : void
+transitionToNext() : void

Decorator design pattern: Used to make unique plant types with different pots and decorations.

PlantDecorator
-component : Plant*
+getPrice() : double
+clone() : Plant*

Seeds
+getPrice() : double
+clone() : Plant*
+getType() : string

State design pattern: Used to keep track of the state of the plant during its life time.

MatureState
+handleCare() : void
+transitionToNext() : void

JuvenileSate
+handleCare() : void
+transitionToNext() : void

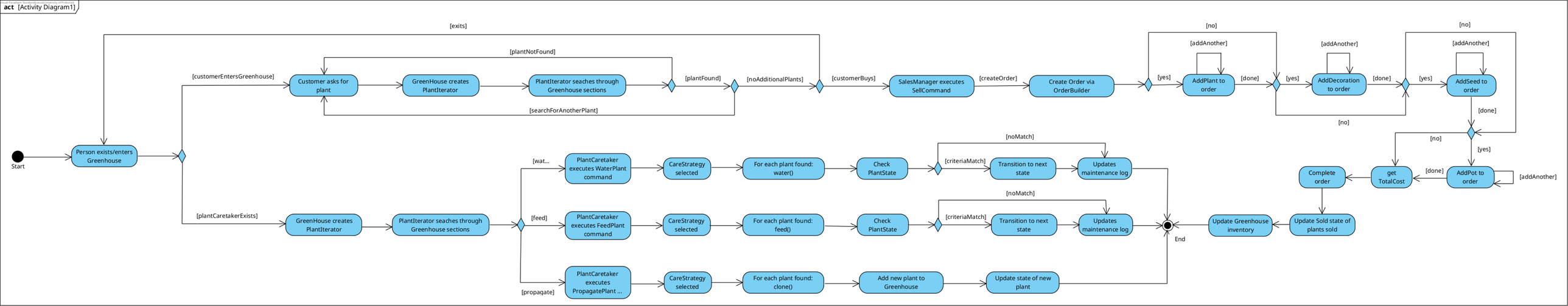
DyingState
+handleCare() : void
+transitionToNext() : void

Decoration
+getPrice() : double
+addRibbon()
+addCard()
+clone() : Plant*
+getType() : string

PlantType
-seedlingTime : int
-juvenileTime : int
-matureTime : int
-timeForNextCare : int
+water() : void
+feed() : void
+getPrice() : double
+clone() : Plant*
+getType() : string

Pot
+getPrice() : double
+clone() : Plant*
+getType() : string

Prototype design pattern: Used to easily clone plants to produce young plants for replanting.


act [Activity Diagram1]
Start
Person exists/enters Greenhouse
[exits]
[customerEntersGreenhouse]
Customer asks for plant
GreenHouse creates PlantIterator
PlantIterator seaches through Greenhouse sections
[plantNotFound]
[plantFound]
[searchForAnotherPlant]
[noAdditionalPlants]
[customerBuys]
SalesManager executes SellCommand
[createOrder]
Create Order via OrderBuilder
[yes]
AddPlant to order
[addAnother]
[done]
[no]
[yes]
AddDecoration to order
[addAnother]
[done]
[no]
[yes]
AddSeed to order
[addAnother]
[done]
[no]
[yes]
AddPot to order
[addAnother]
[done]
get TotalCost
Complete order
Update Sold state of plants sold
Update Greenhouse inventory
End
[plantCaretakerExists]
GreenHouse creates PlantIterator
PlantIterator seaches through Greenhouse sections
[wat...
PlantCaretaker executes WaterPlant command
[feed]
PlantCaretaker executes FeedPlant command
[propagate]
PlantCaretaker executes PropagatePlant ...
CareStrategy selected
For each plant found: water()
For each plant found: feed()
For each plant found: clone()
Check PlantState
[noMatch]
[criteriaMatch]
Transition to next state
Updates maintenance log
Add new plant to Greenhouse
Update state of new plant