UML Design Documents for Amazoom Warehouse Project

60720109

Hironobu Kozu

**<<Executive Summary>>**

This is a system where it is designed to do the functions inside the warehouse automatically. The functions include: preparing orders, checking stock, alerting low stock, checking up on orders, restocking, delivery. This is made possible by a central warehouse computer that controls and manages the system and robots that collect the items and prepare the order. The warehouse computer will check the status of stocks and orders and send messages to the warehouse based on the data. The robots will collect the items based on the messages sent. The system also includes a interface so that the manager can manually access and check the data from the warehouse computer. By using this system and making the warehouse automatic, it will make the warehouse system more efficient and low cost.

**<<Use case scenarios and sequence diagrams>>**

**Use Case: Customer searches for item**

1.The customer inputs keywords for item into system

2.The system will read the keywords

3.The system will send the keywords to warehouse computer

4.The warehouse computer will search storage for related item

5.The warehouse computer will return results to system.

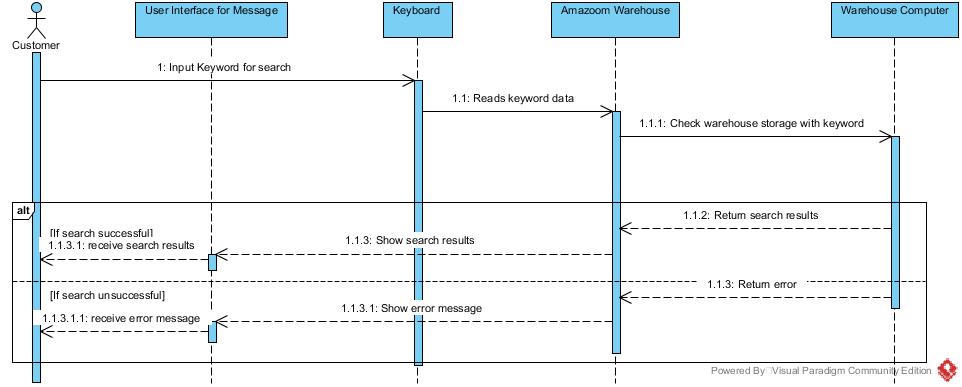
6.If the system cannot find the item <<Scenario 1>>.

7.The customer receives the search results

8.If (customer finds what he wants)

9.Extend use case: Order Item

Scenario 1: The system returns a failed search message to the customer. End-of-transaction



**Use Case: Customer orders item**

1.The customer sends item and quantity data to system as order data

2.The system sends order data to warehouse computer

3.The warehouse computer checks if item and quantity are available

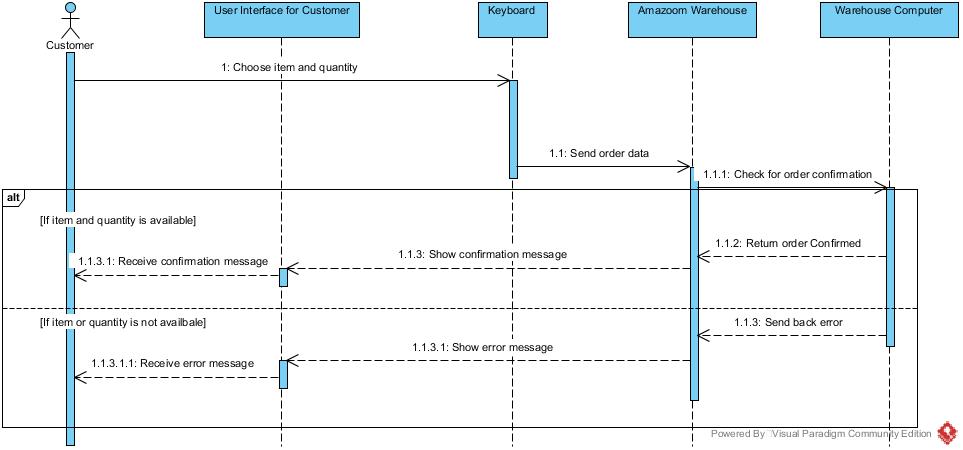
4.The warehouse computer confirms order and sends back notice back to system.

5.If the item is not available<<Scenario 1>>.

6.The system sends the customer a confirmation message

7.The customer receives order confirmation

Scenario 1: The warehouse computer sends back an error message. End-of-transaction.



**Use Case: Manager checks status of order**

1.The manager chooses which order to check

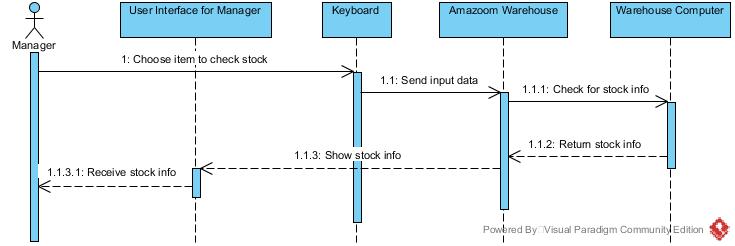
2.The system sends info to the warehouse computer

3.The warehouse computer checks the order status

4.The warehouse computer sends back info to the system

5.The system gives status info to the manager

6.The manager receives status info



**Use Case: Manager checks number of stock for an item**

1.The manager accesses the system to check stock

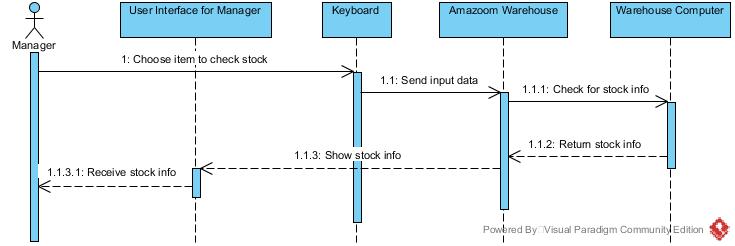
2.The system sends a command to the warehouse computer

3.The warehouse computer checks the stock for the item

4.The warehouse computer sends back info to the system

5.The system gives stock info to manager

6.The manager receives stock info



**Use Case: Manager gets alert on low stock**

1.The warehouse computer checks stock data at a regular basis

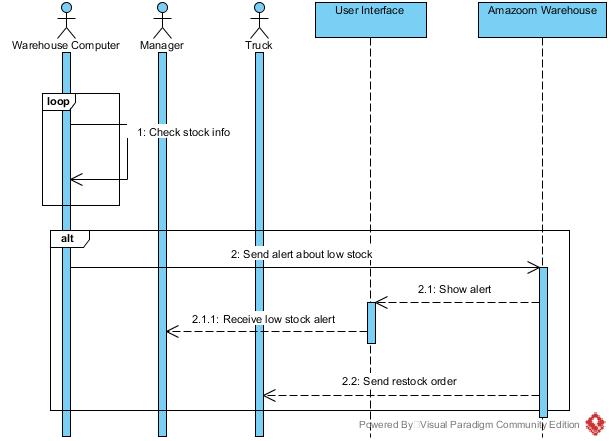
2.When an item is low on stock warehouse computer sends data to system

3.The system sends manager an alert on low stock

4.The manager receives an alert

5.The warehouse computer will also send a restock order to the truck

6. The truck delivers the restock



**Use Case: Deliver orders**

1.The truck waits until the order is prepared.

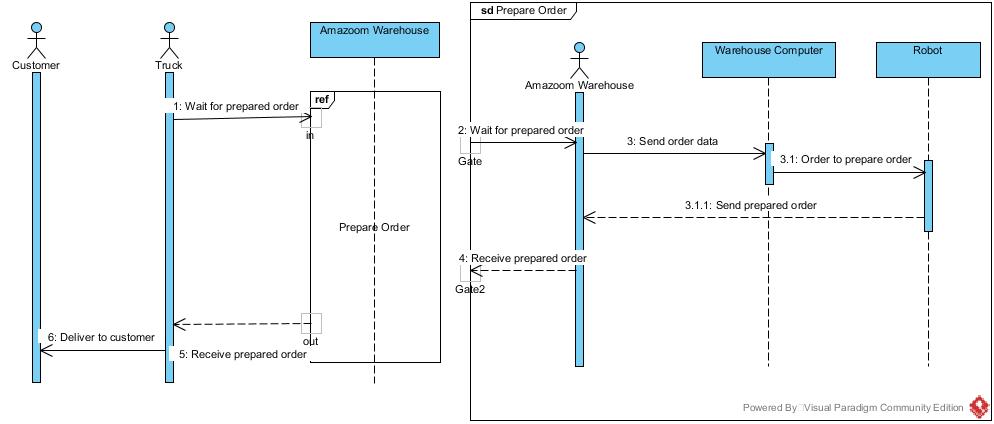
2.After the order is received, the Amazoom warehouse sends the order data to the warehouse computer.

3.The warehouse computer orders the robot to prepare order

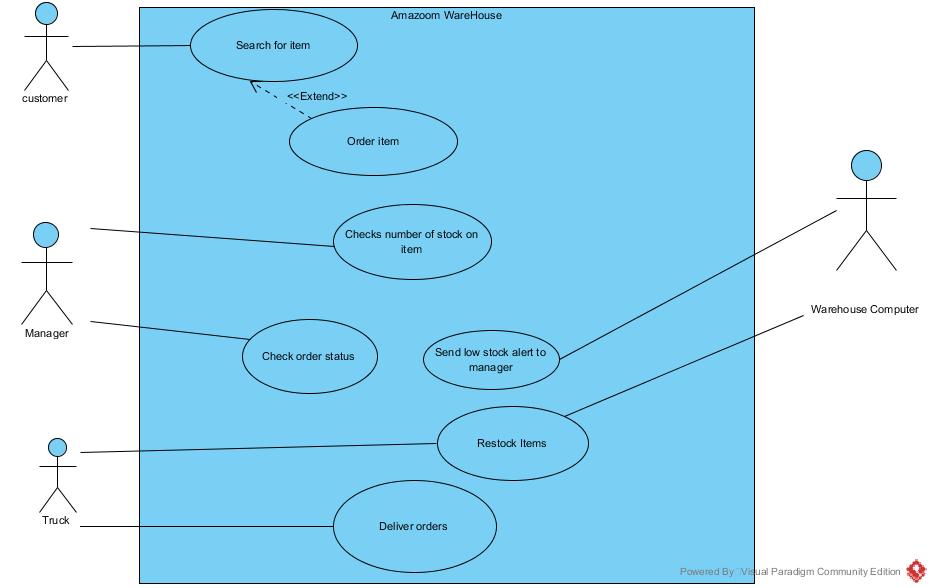
4.The robot prepares the order and places it in queue

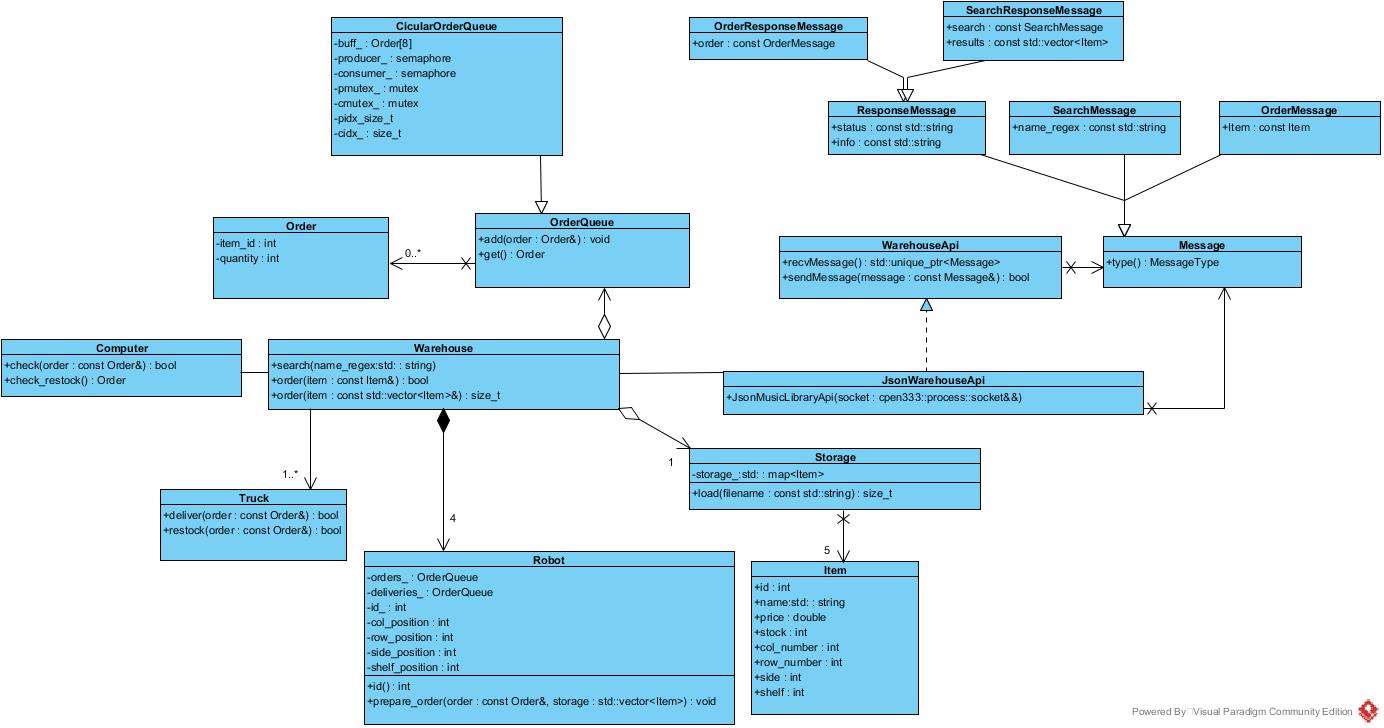
5.The truck waiting for the prepared order, receives the order

6.The truck delivers the order to customer



<**<Use Case Diagram for Amazoom>>**



**<<Class Diagram for Amazoom>>**

**<<Object Interaction Diagram for Amazoom>>**

Diagram of Amazoom consumer producer system

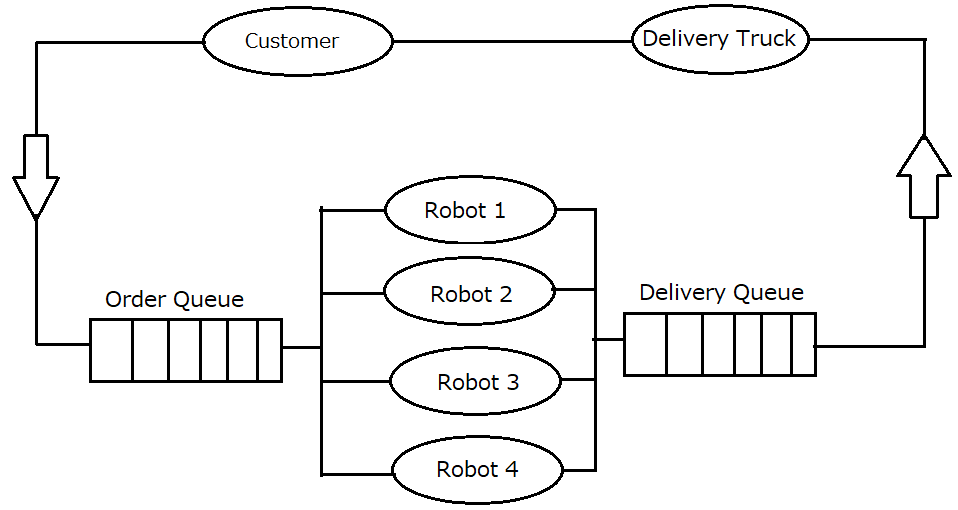
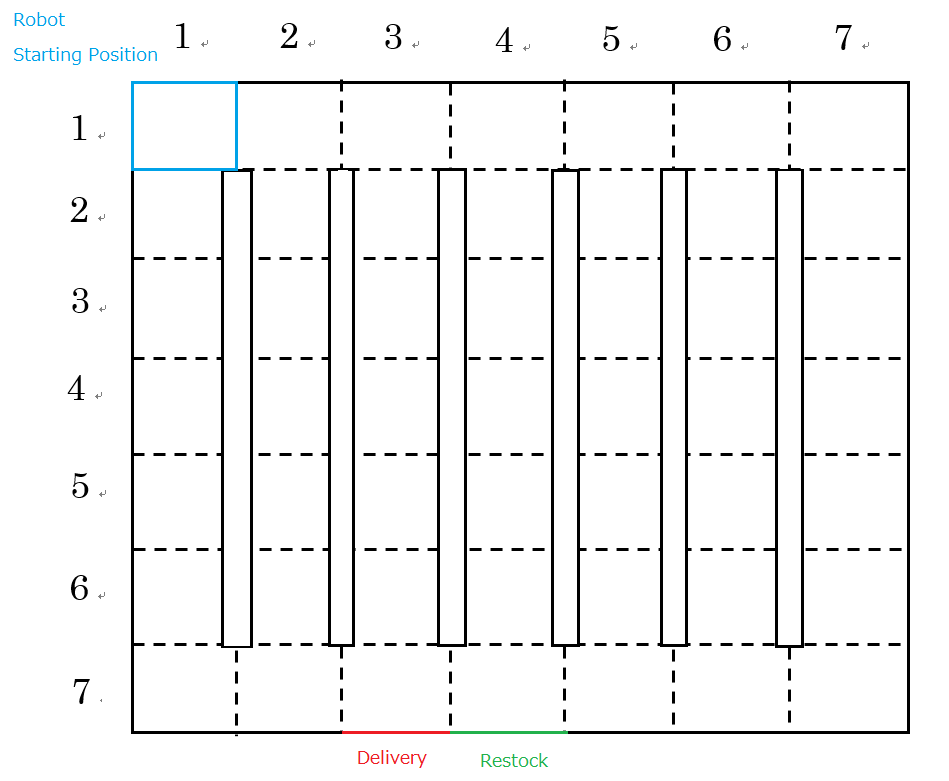


Diagram of warehouse layout



**<<Communication Protocols in Amazoom>>**

A socket is being used for the communication between the warehouse and the customer. Since, the size of primitives can differ from machine to machine, a communication protocol is needed. For the layout for the data that will be sent, the first byte will be a indicator notifying the machine that it is a JSON. The next four bytes will tell the size of the data and the message itself will be appended after that in the form of ASCII character string.

|  |  |  |
| --- | --- | --- |
| indicator | size | Message |
| 1-byte (0x55) | 4-byte big-endian integer | ASCII character string |

For the message API, The system will support the following messages for the communication between the customer and the warehouse.

search – search for items in the warehouse

order – order items in the warehouse

search\_response – returns the search result if successful, but if unsuccessful it will return a error message

order\_response – returns confirmation of order if order is available, if not it will send an error

Within the JSON-encoded messages the items will have the following attributes.

・”id”: string

・”name”: string

・”price”: double

・”stock”: int

・”col\_number”: int

・”row\_number”: int

・”side”: int

・”shelf”: int

The messages passed between the customer and the warehouse all contain a “msg” attribute that identifies the type of message sent.

**SearchMessage**

・”msg”: “search”

・”name\_regex”: string

**SearchResponseMessage**

・”msg”: “search\_response”

・”status”: “OK” or “ERROR”

・”info”: string

・”results”: Item[]

・”search”: SearchMessage

**OrderMessage**

・”msg”: “order”

・”order”: Order

**OrderResponseMessage**

・”msg”: “order\_response”

・”status”: “OK” or “ERROR”

・”info”: string

・”order”:OrderMessage

The system also contains a *WarehouseApi* class but it is a pure abstract class and is implemented by another class *JsonWarehouseApi.* The *JsonWarehouseApi* class a constructor that takes ownership of a socket which allows the communication between the customer and the warehouse. It makes the communication possible by reading and writing the messages using JSON-encoded strings based on the communication protocol.

**<<Function specifications for useful functions>>**

load\_items()

In the system the function is used to load the item data from the JSON file into the storage data. It takes a string as a filename and a map that uses a integer as a key and a *Item* class object as the value for arguments. It reads the file one item at a time, and then inputs the data one by one in to the corresponding attribute of the item.

prepare\_order()

It requires an *Order* class object and a map that uses a integer as a key and a *Item* class object as the value for arguments. This function reads the order, and from the order gets the id of the ordered item. It then searches through the list of storage and finds the item that corresponds with the id. After finding the item, it will get the data of where the item is stored to get the item. This function can be used to get information whose storage place is already known.