Store Model Service Design Document

Date: 9/30/23
Author: Eric Margolis
Reviewer(s): Veronika Post, Jonh Wendell Sousa de Santana

Table of Contents

- Introduction ……1
- Overview ……1
- Requirements ……2
- Use Cases ……2
- Design ……4
- Class Diagram ……..4-5
- Class Dictionary ……5
- Exception Handling ……17
- Design Details ……18
- Testing ……18
- Risks ……19

Introduction

This document defines the Store Model Service of the 24X7 Store. The Store Model Service supports the Store architecture by modeling the physical stores customers visit. Store models track the states of sensors, customers, inventory, and appliances within the store. The Store Model Service provides an API for the Store 24X7 Controller to monitor and maintain each store in the system.

Overview

Grocery stores provide a place for customers to purchase fresh food and other necessities, but the traditional grocery store model creates a service bottleneck around checkout that costs shoppers valuable time. The cumbersome checkout procedure involves taking all the items out of one's bag or cart, having the customer or an employee scan every item, pay for the item, and then re-pack the items into bags. This process is a weekly recurrence for most, and that wasted time adds up.

The 24X7 Store provides an automated shopping experience in which customers can visit, shop for their goods, and walk out without the need for a lengthy checkout process. Instead, cameras and computer vision are used to track customer items, and the customers are automatically billed to their blockchain account when they leave the store. Common store tasks such as stocking, cleaning, and inventorying are performed by robotic assistants. The 24X7 Store reduces operating costs by minimizing human worker intervention and provides a quicker, more

efficient shopping experience for the customer.

Requirements

The Store Model Service must support an arbitrary number of store instances. The Store Model Service must be able to support variations on individual stores, which may have differences in layout, inventory, and the number and type of sensors and appliances.

The Store Model must represent the physical store (including identifying information), aisles, shelves, inventory, products, customers, baskets, turnstiles, robotic assistants, microphones/speaker systems, and cameras.

Furthermore, the Store Model Service must provide an API that allows monitoring, controlling, modifying, and displaying some or all of the above, as well as simulating events for testing purposes. This will be achieved by a Command API that takes command line arguments as listed in the Requirements Document.

Use Cases

Actors

Store 24X7 Controller
The Store Model Service provides an API for the Store Controller Service, allowing the latter to manage the stores in the system. The Controller Service issues commands as outlined in the Requirements Document to define new stores, inventory, products, aisles, shelves, customers, sensors, and appliances; to query the status of sensors and appliances; to assign appliances tasks; and to display the store-specific information regarding all of the above entities.

Appliances
Appliances perform tasks to facilitate the operation of the store. Several sub-classes fall under the Appliance class. Robots are tasked with cleaning, stocking, and assisting customers. Speakers output audio in response to events. Turnstiles check out known customers as they leave, register guests, and manage entry and egress.

Use Cases

Defining a Store
Creates a new Store instance. The store is defined with a unique identifier and a physical address. Aisles and shelves are defined within the store and given numbers and designations. Sensors and appliances are defined and ascribed to the store as resources.

Defining Inventory
Inventory objects are created to represent the qualities and quantity of the products on the

store's shelves.

Defining Products
Products carried by the store are defined with unique identifiers and characteristics. Products are put into or removed from the store inventory by appliances or customers.

Defining Customers
Customers are of two types: registered or guest. Guests must register before they can purchase items. Registration involves assigning a unique ID number to the customer, storing their email and blockchain address, and storing a photo for facial recognition.

Defining Sensors
The two sensor types are camera and microphone. Cameras are placed throughout the store to track customer activity and monitor for events. Microphones are placed to record customer queries or commands. Microphones are accompanied by speakers so that a customer query such as "Where's the milk?" may be answered.

Defining Appliances
Appliances are store assets and include robotic staff, speakers, and turnstiles. Robotic staff clean the store and manage inventory. Speakers accompany microphones, as noted above. Turnstiles check customers out and prevent unregistered guests from leaving with products.

Managing Inventory
Inventory is monitored and updated throughout the day-to-day operation of the store. The inventory life-cycle is delivery, storage, stocking, and purchase. Care must be taken to ensure that items are not over- or under-stocked, that inventory loss is minimized, and that purchased inventory is properly recorded.

Monitoring Sensors
The Store Model Service monitors the states and locations of the sensors placed throughout the store.

Managing Appliances
The Store Model Service provides an API for managing the appliances within the store. The states of the appliances can be queried, and commands can be issued to appliances.

Displaying Information
The Store Model Service accepts various commands (see Command Processor) for displaying information about stores and their components. This list includes details regarding stores, aisles, shelves, inventories, products, customers, baskets, sensors, and appliances.

Simulating Events
Sensors and Appliances have various events that they need to respond to. Sensors detect customer activity and monitor for emergencies, and appliances respond to customer activity and

commands from the store controller. The Store Model Service provides a means of simulating events for testing purposes.

Design

The requirements for this project fall into three categories: modeling stores and their contents (Store Model Service Requirements pages 3-6) and managing stores (pages 7-9).

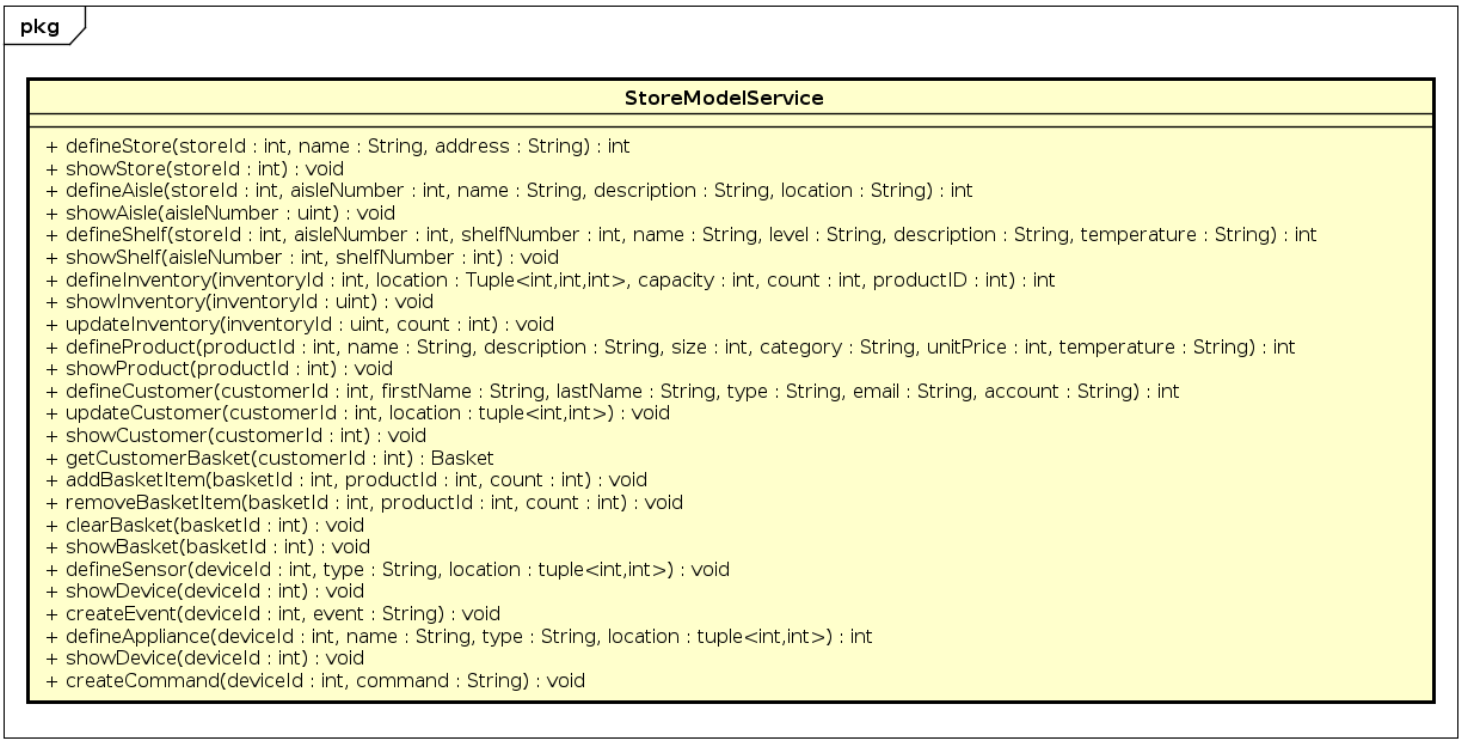
Modeling is achieved by the majority of the classes defined below: the store, the products it sells, its current inventory, its current customers, its layout, its sensors, and its appliances are all represented by instances of the classes in the Class Diagram. Information is updated as events occur and object details can be queried and displayed by "show" commands, which will initiate a printout of the requested information retrieved by getter methods. Additionally, sensors in the store provide information about active customer locations.

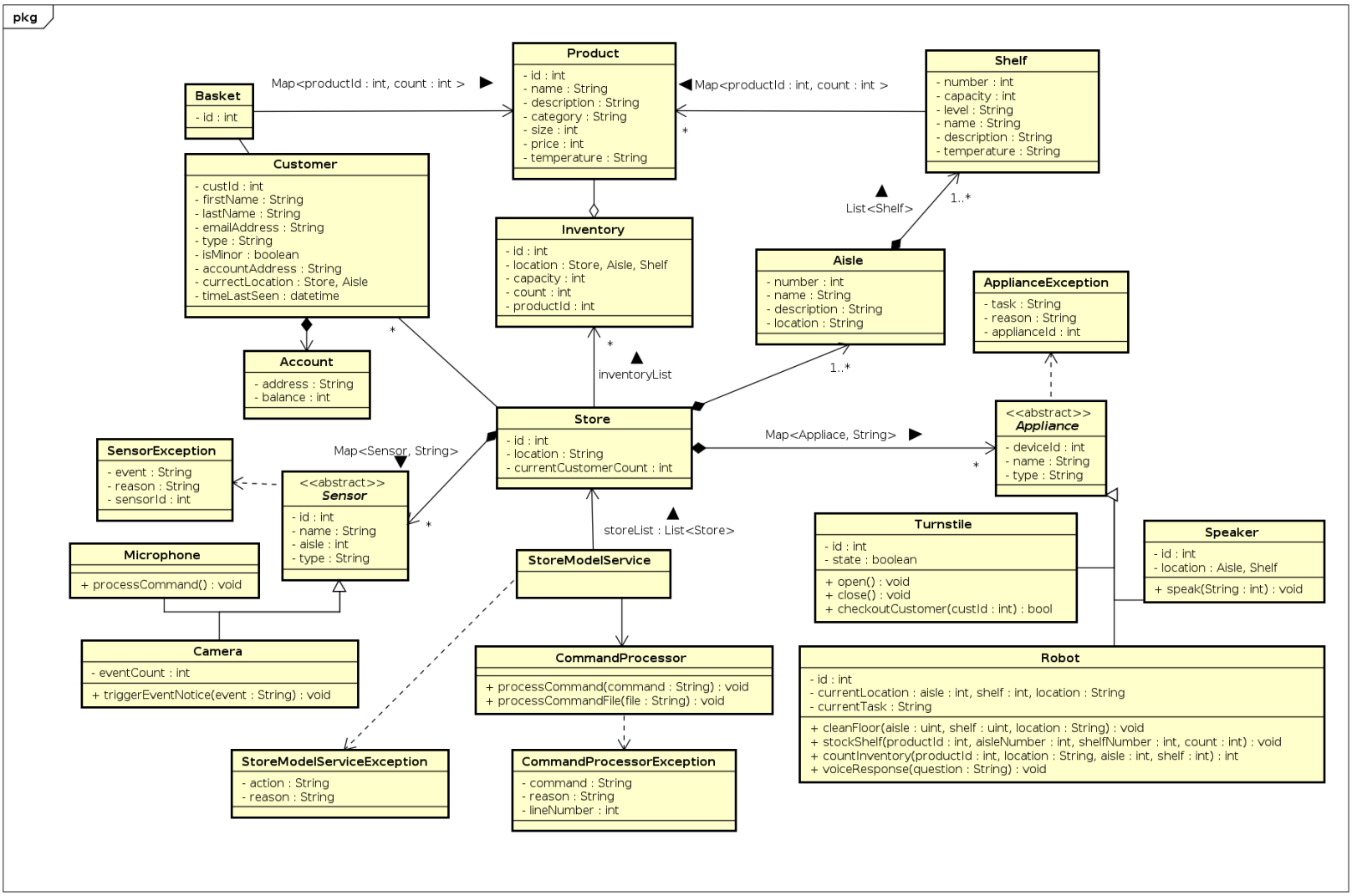
Managing is conducted by the Store Controller Service through the Command Processor. The Store Controller Service sends commands to the Model Service line by line or as script files. The commands are interpreted by the Command Processor and acted upon by store appliances.

This design fulfills the Store Model Service Requirements document by providing the modeling and managing functionality.

Class Diagram

The following class diagram displays the classes defined in this design. The StoreModelService class methods are displayed separately for ease of reading.





Class Dictionary

This section specifies the class dictionary for the Store Model Service. The classes should be defined within the package "cscie97.store.model".

StoreModelService

The StoreModelService class is a Singleton that represents the top-level interface for managing stores. The StoreModelService acts on commands from the Store Controller Service by way of the CommandProcessor class. This class meets the requirements for a command API.

Methods

Method Name	Signature	Description
defineStore	(storeId : int, name : String,	Creates a new Store instance.

	address : String) : int	
showStore	(storeId : uint, authToken : String) : void	Displays the following details of the Store instance to stdout: id, name, address, active customers, aisles, inventory, sensors, and appliances.
defineAisle	(storeId : int, aisleNumber : int, name : String, description : String, location : String) : int	Defines an aisle within a Store.
showAisle	(aisleNumber : uint) : void	Display to stdout the aisle's name, description, and list of shelves.
defineShelf	(storeId : int, aisleNumber : int, shelfNumber : int, name : String, level : String, description : String, temperature : String) : int	Defines a shelf within a Store. See Shelf class for level and temperature definitions.
showShelf	(aisleNumber, shelfNumber, authToken : String) : void	Display to stdout the shelf's ID, name, description, and temperature.
defineInventory	(inventoryId : int, location : Tuple<int,int,int>, capacity : int, count : int, productID : int) : int	Defines an Inventory for a shelf instance. Location is set by <storeId, aisleId, shelfId>.
showInventory	(inventoryId : uint, authToken : String) : void	Display to stdout the inventory details.
updateInventory	(inventoryId : uint, count : int, authToken : String) : void	Increment or decrement the inventory count.
defineProduct	(productId : int, name : String, description : String, size : int, category : String, unitPrice : int, temperature : String) : int	Defines a new product to be put into store inventory.
showProduct	(productId : int) : void	Displays details of the product.
defineCustomer	(customerId : int, firstName : String, lastName : String, type : String, email : String, account : String) : int	Defines a new customer.
updateCustomer	(customerId : int, location : tuple<int,int>) : void	Updates the given customer's last seen location. Location is <storeId, aisleId>,
showCustomer	(customerId : int) : void	Display the customer's details.
getCustomerBasket	(customerId : int) : Basket	Returns the Basket instance that the customer is using.

addBasketItem	(basketId : int, productId : int, count : int) : void	Adds a product to the basket.
removeBasketItem	(basketId : int, productId : int, count : int) : void	Removes the product from the basket when the customer puts the item back.
clearBasket	(basketId : int) : void	Clears the basket's product map.
showBasket	(basketId : int) : void	Display the contents of the basket.
defineSensor	(deviceId : int, type : String, location : tuple<int, int>) : void	Defines a new Sensor object at location <store, aisle>.
showDevice	(deviceId : int) : void	Display device details.
createEvent	(deviceId : int, event : String) : void	Creates a sensor event. This is a simulated event for testing purposes.
defineAppliance	(deviceId : int, name : String, type : String, location : tuple<int, int>) : int	Defines a new Appliance object at location <store, aisle>.
createCommand	(deviceId : int, command : String) : void	Creates a command message to send to an Appliance.

Associations

Association Name	**Type**	**Description**
storeList	List<Store>	A list of all the 24X7 stores in the system by ID number.

Store

The Store class presents a virtual view of physical 24X7 stores. Each store has a globally unique ID number, a location, and a count of customers in the store at any given time. Its associations are a list of Inventory instances, and maps of the store's Sensors and Appliances.

Properties

Property Name	**Type**	**Description**
id	uint	Unique store identifier.
location	String	The store's physical (mailing) address.
currentCustomerCount	uint	The number of customers currently in the store.

Associations

Association Name	Type	Description
inventoryList	List<Inventory>	A list that stores the store's inventory of products.
sensorMap	Map<Sensor, String>	A map of sensors in the store along with their types.
applianceMap	Map<Appliance, String>	A map of appliances in the store along with their types.

Appliance

Appliances are the active resources of each store. Appliances fall into the subclasses of Robot, Turnstile, and Speaker. Appliances have tasks, such as cleaning floors, stocking shelves, or checking out customers.

Properties

Property Name	Type	Description
deviceId	uint	The store-unique ID for the appliance.
name	String	The name of the appliance.
type	String	Appliance type (robot, speaker, turnstile).

Robot

The Robot class represents robotic support staff that move around the store and perform operations such as restocking, counting inventory, and cleaning. A robot has a unique identifier, live location information (its current location in the store), and an identifier for the task it is performing at a given moment.

Methods

Method Name	Signature	Description
cleanFloor	aisle : uint shelf : uint location : String	Cleans the floor at the designated location.
stockShelf	productId : uint, aisleNumber : uint, shelfNumber : uint, count : uint	Stocks the given shelf with <count> number of the designated product. Shelf location is specified by aisle number. The shelf is always on the store floor.
countInventory	productId : uint	Returns a count of the inventory of a given

	location : String aisle : uint shelf : uint	product on a given shelf. Shelf location is specified by aisle number and location: floor or storeroom.

Properties

Property Name	Type	Description
id	uint	The robot's unique identifier.
currentLocation	aisle : uint shelf : uint location : String	The robot's current location in the store.
currentTask	String	The task that the robot is currently performing.

Turnstile

This class represents the turnstiles through which customers exit the store. Each turnstile has a unique identifier and a state, open or closed, and automatically checks the customer out based on the items in their basket as they leave.

Methods

Method Name	Signature	Description
open		Opens the turnstile if it's closed.
close		Closes the turnstile if it's open.
checkoutCustomer	customerId : uint	Queries the cost of the customer's basket and initiates a transfer of funds from the customer's account to the store account using the ledger service. Returns a boolean signifying transaction success (True) or failure (False).

Properties

Property Name	Type	Description
id	Int	The turnstile's id within the store.
state	bool	The state of the turnstile (open or closed).

Inventory

The Inventory class represents the quantity and qualities of items stocked on shelves

throughout the store. Inventory items are identified by product ID and each product has its own capacity in units of weight or volume, as well as a current count and location.

Properties

Property Name	Type	Description
id	uint	The unique identifier for the inventory object.
location	store : Store, aisle : Aisle, shelf : Shelf	Where the inventoried products are held.
capacity	int	The number of products that can fit on one shelf in arbitrary units.
count	int	The number of products currently in inventory.
productId	int	The unique ID of the inventoried product.

Aisle

The Aisle class represents divisions in the store by which shelves are ordered. Each aisle has a number of shelves in it, each of which may hold products. Aisles are identified by their store-unique number, as well as the general type of products they contain (dairy, imported foods, frozen foods, etc.) with a description to go along with it. They also have a location attribute that informs whether they are on the floor or in the storeroom.

Properties

Property Name	Type	Description
number	int	The aisle's number.
name	String	The aisle's name is based on the general class of goods contained within it (e.g., Dairy).
description	String	The description expands on the name. The Dairy aisle may have a description like "Milk, yogurt, cheese, eggs, vegan milk alternatives," and so on.
location	String	Floor or storeroom.

Associations

Association Name	Type	Description

shelfList	List<Shelf>	A list of the shelves within the aisle instance.

Shelf

The Shelf is the fundamental storage region for Products in the store. Shelves are located within Aisles and have an identifying number, a capacity, and a map of the Products on them.

Properties

Property Name	**Type**	**Description**
number	uint	The shelf's number within its aisle.
capacity	uint	The shelf's maximum capacity
level	String	The vertical location of the shelf (low, medium, high).
name	String	The name of the shelf (e.g., bread).
description	String	The shelf's description expands on its name (whole wheat, white, multigrain, hamburger rolls, etc.).
temperature	String	The temperature at which the shelf is kept (frozen, refrigerated, ambient, warm, or hot. Default = ambient).

Associations

Association Name	**Type**	**Description**
productMap	Map<uint, uint>	The products on the shelf are tracked by their productID and quantity (count).

Product

The Product class represents the items that customers buy from the store. Products have a name, a description, a unique identifier, and a category. Products are placed on shelves. Customers take products off of shelves and move them to their shopping baskets.

Properties

***Property* Name**	**Type**	**Description**
productId	uint	The product's unique identifier.
name	String	The product's name (e.g., yogurt).

description	String	The description of the product (e.g., low fat, Greek, blueberry).
size	uint	The weight of the product.
price	uint	The price in blockchain currency units.
temperature	String	Temperature the item is kept at in the store. Values are (frozen, refrigerated, ambient, warm, hot. Default: ambient).

Customer

The Customer provides an abstraction for the store patrons. Customers are identified by facial and voice recognition by the System. Customer locations are monitored and updated while they are in the store. Customers can be Adults or Children. Customers may be registered users or guests. Guests are not allowed to remove items from the store without registering.

Properties

Property Name	Type	Description
id	uint	The customer's unique identifier.
firstName	String	Customer's first name.
lastName	String	Customer's last name.
isMinor	bool	True if the customer is under the age of 18.
type	String	Registered or guest. Registered customers walk out of the store and are billed automatically. Guests must register before purchasing items.
emailAddress	String	The email address that the customer registered with.
accountAddress	String	Address of the customer's blockchain account. Used for billing.
currentLocation	Tuple<uint, uint>	The customer's current location (Store, Aisle).

Associations

Association Name	Type	Description
Basket	int	The ID of the basket the customer is using.

Sensor(abstract)

The Sensor class is an abstract class that represents the store's cameras and microphone/speaker systems. Each sensor has the following attributes: a store-unique ID, a name, a location, and a type.

Properties

Property Name	Type	Description
id	int	The store-unique ID number of the sensor.
name	String	Sensor name, e.g., "Aisle 3 Camera"
aisle	int	The Aisle that the sensor is located in within the store.
type	String	Sensor type, e.g., "Microphone"

Camera

The Camera class extends the Sensor interface. Cameras have all the properties of Sensors, have an eventCount property, and have an event signaler method.

Methods

Method Name	Signature	Description
triggerEventNotice	(event : String) : void	Notifies the System controller when an event is detected and the type of event (e.g., "fire").

Properties

Property Name	Type	Description
eventCount	uint	The number of events that the camera has detected.

Microphone

The Microphone class represents microphone/speaker combination systems placed throughout the store that respond to voice commands. Commands include questions, such as "Where is the milk?" to which the speaker may output "The milk is stocked in aisle 1." The Microphone inherits all the properties of the Sensor class.

Methods

Method Name	Signature	Description

processCommand		This method responds to an audio input from a customer. The vocal command is converted to a String and is processed. The response is delivered audibly to the customer by the speaker.

Properties

Property Name	Type	Description
		All properties inherited from Sensor.

Account

Account instances are representative of the blockchain accounts of each Customer. Accounts are identified by a unique address and have a balance in units of blockchain units.

Properties

Property Name	Type	Description
address	String	The unique account identifier.
balance	uint	The account balance in blockchain units.

Basket

Baskets are used by customers to hold products while the customer is shopping. Baskets are identified by a store-unique ID number. Baskets have contents of type <Product, count>.

Properties

Property Name	Type	Description
id	uint	The basket's store-unique identifier.

Associations

Association Name	Type	Description
productMap	Map<Product, uint>	The basket's contents are represented by a map of products (keys) and counts (values).

CommandProcessor

The CommandProcessor class provides an API between the Store Model Service and the Store Controller Service. The Controller can send commands to the Model. Commands may create new object instances to model new stores, products, and so forth; they may cause changes to

existing stores by tasking appliances to perform operations; or they may request information about a store and its contents. Throws a CommandProcessorException on failure.

Methods

Method Name	Signature	Description
processCommand	(command : String) : void	Processes a single line from the CLI (see commands and syntax below).
proccessCommandFile	(file : String) : void	Takes a path to a script file as an argument and attempts to read from the file. Commands in the script should be separated by newline characters. Lines starting with a hash symbol are ignored.

The CommandProcessor supports the following commands (taken from the Store Model Service Requirements document. Credit Eric Gieseke, CSCIE-97). Note that lines starting with a hash are considered comments and ignored. Note also that although word wrapping may occur in this document, commands must be a single line.

Store Commands
Define a store
define store <identifier>
name <name> address <address>
Show the details of a store; print out the details, including the ID, name, address, active customers, aisles, inventory, sensors, and devices.
show store <identifier>

Aisle Commands
Define an aisle within the store
define aisle <store_id>:<aisle_number> name <name> description <description>
location (floor | store_room)
Show the aisle details, including the name, description, and list of shelves.
show aisle <store_id>[:<aisle_number>]
Shelf Commands
Define a new shelf within the store
define shelf <store_id>:<aisle_number>:<shelf_id> name <name> level (high | medium | low) description <description> [temperature (frozen | refrigerated | ambient | warm | hot)]
Show the details of the shelf, including ID, name, level, description, and temperature
show shelf <store_id>[:<aisle_number>[:<shelf_id>]]

Inventory Commands
Define a new inventory item within the store
define inventory <inventory_id> location <store_id>:<aisle_number>:<shelf_id>

capacity <capacity> count <count> product <product_id>
Show the details of the inventory
show inventory <inventory_id>
Update the inventory count; count must >= 0 and <= capacity
update inventory <inventory_id> update_count <increment or decrement>

Product Commands
Define a new product
define product <product_id> name <name> description <description> size
<size> category <category> unit_price <unit_price> [temperature (frozen |
refrigerated | ambient | warm | hot)]
Show the details of the product
show product <product_id>

Customer Commands
Define a new customer
define customer <customer_id> first_name <first_name> last_name <last_name>
type (registered|guest) email_address <email> account <account_address>
Update the location of a customer
update customer <customer_id> location <store:aisle>
Show the details of the customer
show customer <customer_id>

Basket Commands
Get basket_id associated with the customer, and create a new basket if the customer does not
already have a basket associated.
get_customer_basket <customer_id>
Add a product item to a basket
add_basket_item <basket_id> product <product_id> item_count <count>
Remove a product item from a basket
remove_basket_item <basket_id> product <product_id> item_count <count>
Clear the contents of the basket and remove the customer association
clear_basket <basket_id>
Get the list of product items in the basket, include the product_id, and count
Show basket_items <basket_id>

Sensor Commands
Define a device of type sensor
define device <device_id> name <name> type (microphone|camera) location
<store>:<aisle>
Show device details
show device <device_id>
Create a sensor event; this simulates a sensor event
create_event <device_id> event <event>

Appliance Commands
Define a device of type appliance
define device <device_id> name <name> type (speaker | robot | turnstile)
location <store>:<aisle>
Show device details
show device <device_id>
Create an appliance event; this simulates a sensor event
create event <device_id> event <event_description>
Send the appliance a command
create command <device_id> message <command>

Exception Handling

Errors are expected to occur during store operation due to the unpredictable nature of the store's customers and the physical environment each store is in. It is crucial that errors are handled gracefully to allow the system to continue to operate in an automated fashion.

CommandProcessorException

A CommandProcessorException is thrown when the Command Processor reads an invalid command line. The CommandProcessorException records the command, the reason for failure, and the line number of the command (0 if the command was issued by the CLI and not read from a script file).

Properties

Property Name	Type	Description
command	String	The command that caused the exception.
reason	String	The reason that the command was invalid.
lineNumber	uint	The line number of the command in the script file, or 0 for a CLI command.

StoreModelServiceException

A StoreModelServiceException is thrown when the store encounters an error. The StoreModelServiceException records the action and the reason for failure.

Properties

Property Name	Type	Description
action	String	The action that caused the exception.
reason	String	The reason that the command was invalid.

ApplianceException

An ApplianceException is thrown when an appliance is unable to complete a task. The ApplianceException records the action and the reason for failure.

Properties

Property Name	Type	Description
action	String	The action that caused the exception.
reason	String	The reason that the command was invalid.

Design Details

The Command Processor is the main workhorse of this design as it provides the API for the Store Controller Service and facilitates the project's functional requirements.
The Store class is another key component as it maintains much of the system's information. It also houses the methods that are called to display information about the state of the store and its elements, which is one of the main requirements of the Store Model Service.

The Store class acts as a central hub from which all the component elements radiate because the system is required to scale to an arbitrary number of stores. With the current design, the store acts as the first part of the identifier for each component, and because of this feature, component IDs need only be unique to the store, not to the system as a whole (that is to say that Store 1 may have a Robots 1, 2, and 3 and Store 2 may have Robots 1, 2, 3, and 4, so long as each store maintains that no two robots have the same in-store identifiers.)

The exception is the Customer class; a registered customer is free to shop at any store they wish and should not have duplicate accounts within the 24X7 Store system. Therefore, special care should be taken to ensure that Customers have globally unique identifiers.

Testing

A test driver class called TestDriver should be implemented, which houses a static main() method. This test driver class should accept a single string-type argument, which may represent a single command delivered via a command line interface, or the path to a script file containing one or more commands. The TestDriver class should be defined in the "cscie97.store.test" package.

Risks

The current design does not address concurrency or persistence issues. Future updates should address these issues to ensure a robust and reliable system.

Special care should be taken to ensure that the Store Model Service is prepared to handle emergencies, such as a fire or medical emergency. If an emergency is detected, the store should contact local emergency services and ensure entry and egress are allowed where appropriate. For example, in the event of a fire, the turnstiles should be opened and the speakers should play a message to alert customers of the danger.

The store must also be prepared for non-emergent criminal behavior, such as a guest jumping over or vandalizing turnstiles in order to shoplift. The authorities should be notified and evidence provided in such cases.