Store 24X7 Model Service Design Document

Date: 9 OCT 2019
Author: Connor Garet
Reviewer(s): Owen Murphy, Robert Collins

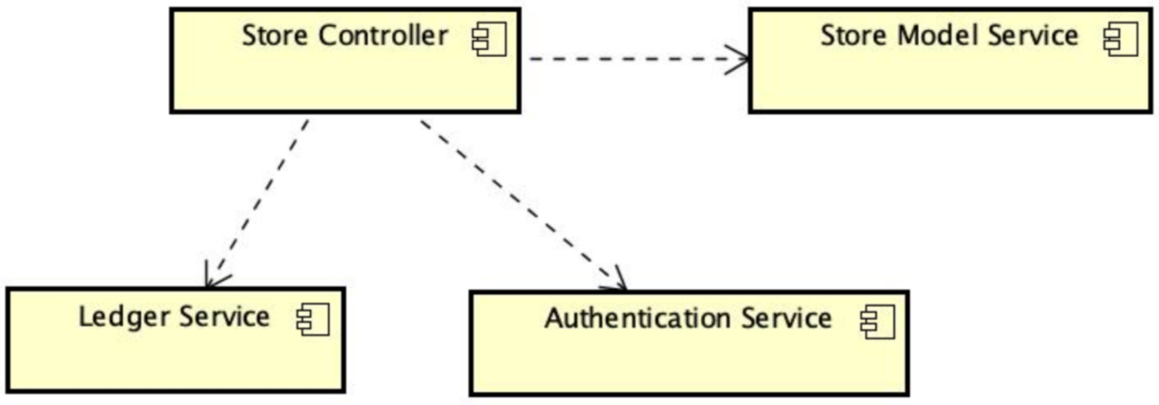
Introduction

This document defines the design for the Store 24X7 Model Service, henceforth simply referred to as the Store Model Service.

Overview

The Store Model Service is one of four components in the Store 24X7 System. When combined with the other components, this system represents all software required to drive the store of the future, in which human employment and interaction is not required. For example, patrons of the store may purchase items by simply removing them from the shelves and walking out the turnstile with them. Without additional input, the system is able to detect customer movements around the store, track their purchases, and remove required funds from the customer accounts to rectify payments. Payments are continually validated via a Block Chain Ledger system, defined in the Ledger Component.

The Store Model Service has no dependencies to any other components in the Store 24X7 System. Instead, the Store Model Service defines an interface with which external components may interact. In the Store 24X7 System, the only other component with which there is an anticipated interaction is the Store Controller.



Caption: UML Component Diagram describing the high-level services for the Store 24X7 Software System. (Taken from Store 24X7 Software System Architecture, E. Gieseke)

The Store Model Service represents the data model of the Store 24X7 System. Its responsibilities include interactions with the physical world via Sensors and Appliances, as well as the management of customers and products in the system.

Requirements

This section provides a summary of the requirements for the Store Model Service. These requirements were developed based on the specifications in the Store Model Service Requirements document, as well as the 24X7 Store System Architecture document.

The Store Model Service shall allow users to:

Store Management

1. Create Stores
2. Get Stores by name
3. Add Shelves and Aisles to Stores
4. Get Shelves and Aisles
5. Create Sensors which can report information to the Store Model Service
6. Create Appliances, which can report information to the Store Model Service as well as take action as commanded by the Store Model Service

Customer Account Management

1. Create Customer accounts
2. Get Customer accounts by name
3. Track Customer location within a Store or across Stores
4. Track Products in a Customer's Basket

Inventory Management

1. Manage Products available for Stores to sell
2. Manage Inventory of all Shelves in Stores

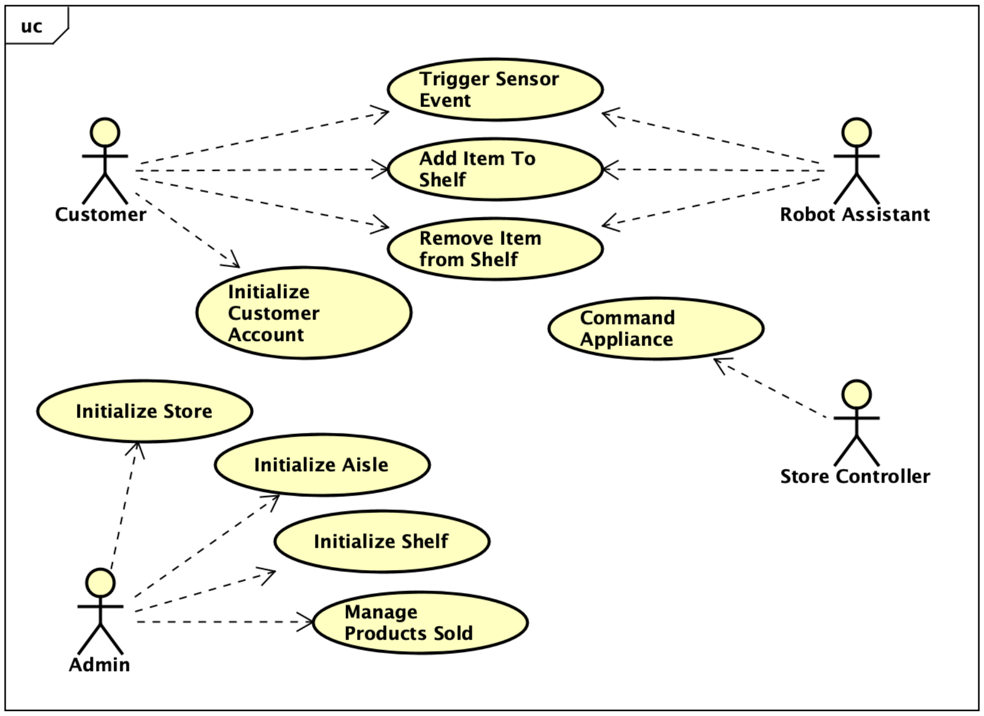
Not Required

Automatic Sensor Events

- As this is a system implemented entirely in software, the Sensor instances cannot self-trigger.
- To simulate a Sensor requesting an event be handled, users of the Store Model Service must use the HandleSensorEvent command as defined in the *StoreModelServiceInterface* API.
    - Alternatively, users may use the CommandProcessor's `create event` command, as described in the Store Model Ser

Use Cases

This design supports the following use cases:



Caption: Use Case diagram for the Store Model Service

Actors

Admin

The Admin of the system is any user permitted to manage information about Stores. Typically, these will be Store managers who add their store to the Store Model Service.

Customer

Customers are patrons of the store. They are permitted to interact with items on the shelves and with their own Customer accounts.

Robot Assistant

The Robotic Assistant is the pseudo-employee in the Store. These appliances are tasked with stocking shelves and handling Customer interactions when necessary.

Store Controller

The Store Controller is another component in the Store 24X7 System. The Store Controller makes decisions as to what the Appliances should be doing, and requests that the Store Model Service take the appropriate action.

Use Cases

Initialize Store

The Admin may initialize Stores with a unique identifier, a Store name, and an address.

Initialize Aisle
The Admin may initialize Aisles within an identified Store with an aisle number unique to the store, a name, a description, and a location

Initialize Shelf
The Admin may initialize Shelves within a given Aisle and Store with a Shelf number unique to the Aisle, and information about the shelf height and food that can be stored on the shelf.

Manage Products sold
The Admin may add new Product types to the Store Model Service that can be stored on shelves and sold to customers

Initialize Customer Accounts
Customers may create their store account to be linked to their payer account and email. Customers must indicate whether or not they would like to register with the system.

Trigger Sensor Event
Customers and Robotic Assistants may inadvertently trigger sensor events based on their movement and noises made. This includes the tracking of Customer positions.

Add Items to Shelf
Customers and Robotic Assistants may add items to shelves to replenish inventory or return items that are no longer desired.

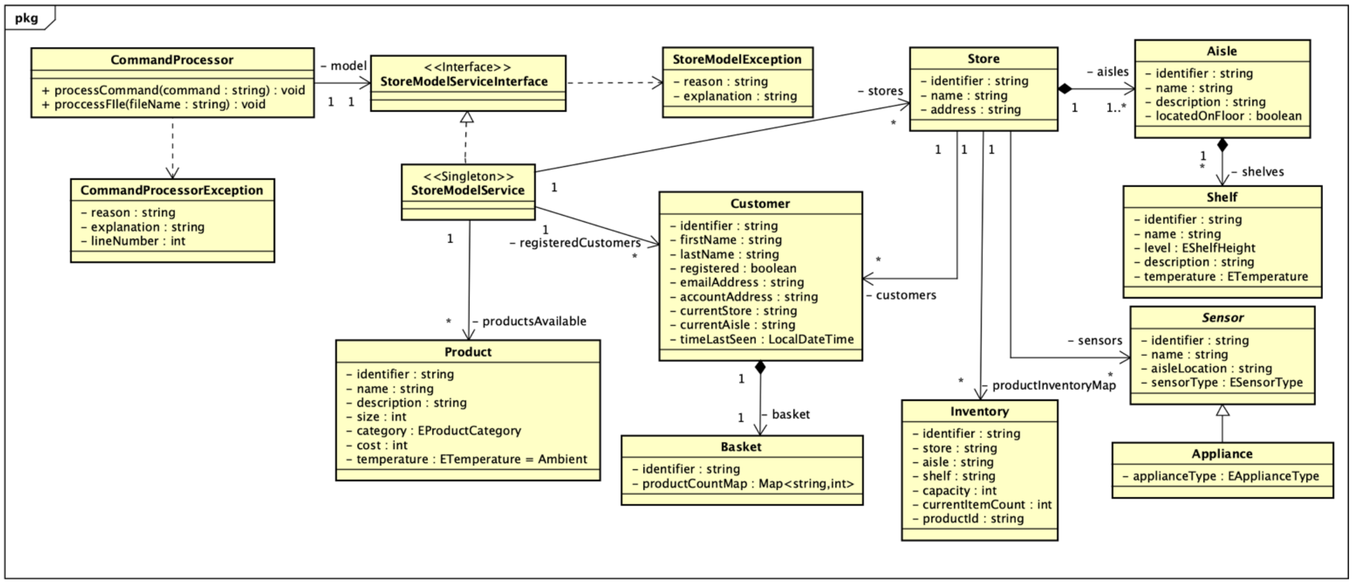
Remove Items from Shelf
Customers and Robotic Assistants may remove items from shelves to be added to Customer Baskets or to be removed from Inventory

Command Appliance
The Store Controller may command Appliance movement to satisfy the needs of Customers. Examples of this include commanding Robotic Assistant movement and opening turnstiles for customers to enter or exit through.

Implementation

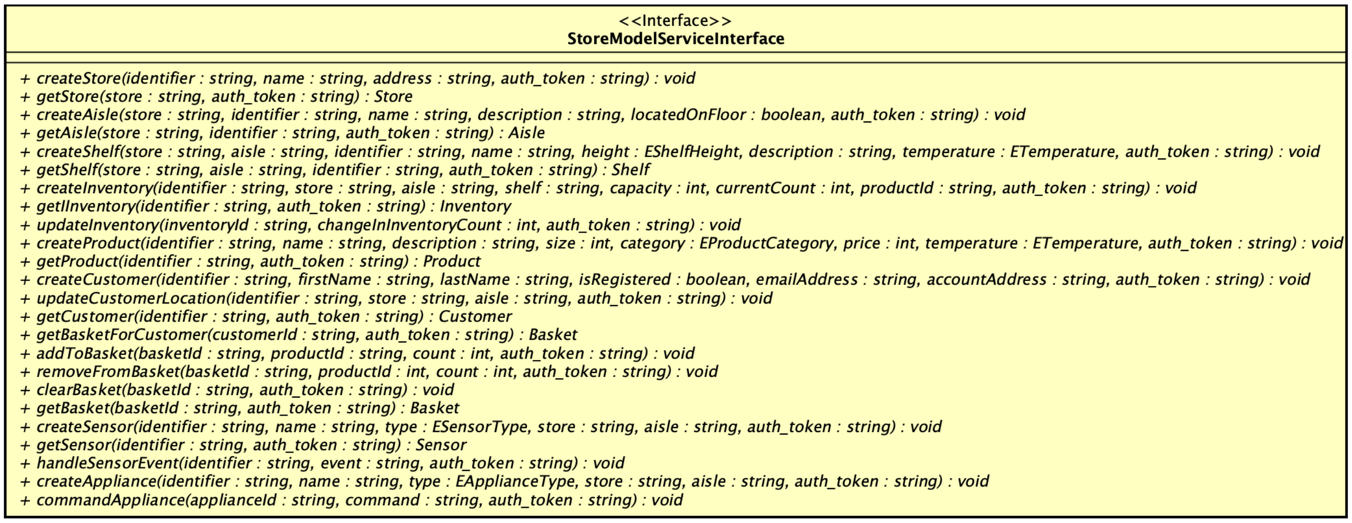
Class Diagram

The following class diagram defines the classes defined in this design.



Caption: UML Class Diagram of the Store Model Service component.

Because the *StoreModelServiceInterface* is rather large, an independent Class Diagram is provided below to show a detailed view of the interface.



<<Interface>>
StoreModelServiceInterface

+ createStore(identifier : string, name : string, address : string, auth_token : string) : void
+ getStore(store : string, auth_token : string) : Store
+ createAisle(store : string, identifier : string, name : string, description : string, locatedOnFloor : boolean, auth_token : string) : void
+ getAisle(store : string, identifier : string, auth_token : string) : Aisle
+ createShelf(store : string, aisle : string, identifier : string, name : string, height : EShelfHeight, description : string, temperature : ETemperature, auth_token : string) : void
+ getShelf(store : string, aisle : string, identifier : string, auth_token : string) : Shelf
+ createInventory(identifier : string, store : string, aisle : string, shelf : string, capacity : int, currentCount : int, productId : string, auth_token : string) : void
+ getIInventory(identifier : string, auth_token : string) : Inventory
+ updateInventory(inventoryId : string, changeInInventoryCount : int, auth_token : string) : void
+ createProduct(identifier : string, name : string, description : string, size : int, category : EProductCategory, price : int, temperature : ETemperature, auth_token : string) : void
+ getProduct(identifier : string, auth_token : string) : Product
+ createCustomer(identifier : string, firstName : string, lastName : string, isRegistered : boolean, emailAddress : string, accountAddress : string, auth_token : string) : void
+ updateCustomerLocation(identifier : string, store : string, aisle : string, auth_token : string) : void
+ getCustomer(identifier : string, auth_token : string) : Customer
+ getBasketForCustomer(customerId : string, auth_token : string) : Basket
+ addToBasket(basketId : string, productId : string, count : int, auth_token : string) : void
+ removeFromBasket(basketId : string, productId : int, count : int, auth_token : string) : void
+ clearBasket(basketId : string, auth_token : string) : void
+ getBasket(basketId : string, auth_token : string) : Basket
+ createSensor(identifier : string, name : string, type : ESensorType, store : string, aisle : string, auth_token : string) : void
+ getSensor(identifier : string, auth_token : string) : Sensor
+ handleSensorEvent(identifier : string, event : string, auth_token : string) : void
+ createAppliance(identifier : string, name : string, type : EApplianceType, store : string, aisle : string, auth_token : string) : void
+ commandAppliance(applianceId : string, command : string, auth_token : string) : void

Caption: UML Class Diagram of the *StromeModelServiceInterface* class within the Store Model Service.

Class Dictionary

This section specifies the class dictionary for the classes defined within the package cscie97.store.model for use in the Store Model Service

StoreModelServiceInterface

The StoreModelServiceInterface interface is the main interaction point for outside entities and users to interact with the Store Model Service. It defines all available functionality for the system as a whole as described by the Requirements document (pg 7-9).

Methods

As an interface, all methods have only a signature and not an implementation.

Method Name	Signature
createStore	(identifier:string, name:string, address:string, auth_token:string) → void
getStore	(identifier:string, auth_token:string) → Store
createAisle	(store:string, identifier:string, name:string, description:string, locatedOnFloor:boolean, auth_token:string) →void
getAisle	(store:string, identifier:string, auth_token:string) →Aisle
createShelf	(store:string, aisle:string, identifier:string, name:string, height:EShelfHeight, description:string, temperature:ETemperature, auth_token:string) → void
getShelf	(store:string, aisle:string, identifier:string, auth_token:string) →Shelf
createInventory	(identifier:string, store:string, aisle:string, shelf:string, capacity:int, currentCount:int, productid:string, auth_token:string) → void
getInventory	(identifier:string, auth_token:string) → Inventory
updateInventory	(identifier:string, changeInInventoryCount:int, auth_token:string) → Inventory
createProduct	(identifier:string, name:string, description:string, size:int, category:EProductCategory, price:int, temperature:ETemperature, auth_token:string) → void
getProduct	(identifier:string, auth_token:string) → Product
createCustomer	(identifier:string, firstName:string, lastName:string, isRegistered:boolean, emailAddress:string, accountAddress:string, auth_token:string) → void
updateCustomerLocation	(identifier:string, store:string, aisle:string, auth_token:string) → void
getCustomer	(identifier:string, auth_token:string) → Customer
getBasketForCustomer	(customerId:string, auth_token:string) → Basket
addToBasket	(basketId:string, productId:string, count:int, auth_token:string) → void
removeFromBasket	(basketId:string, productid:string, count:int, auth_token:string) → void
clearBasket	(basketId:string, auth_token:string) → void
getBasket	(basketId:string, auth_token:string) → Basket
createSensor	(identifier:string, name:string, type:ESensorType, store:string, aisle:string, auth_token:string) → void

getSensor	(identifier:string, auth_token:string) → Sensor
handleSensorEvent	(identifier:string, event:string, auth_token:string) → void
createAppliance	(identifier:string, name:string, type:EAppliacneType, store:string, aisle:string, auth_token:string) → void
commandAppliance	(identifier:string, command:string, auth_token:string) → void

Attributes
As an interface, the StoreModelServiceInterface does not have any attributes

Associations
As an interface, the StoreModelServiceInterface does not have any associations.

StoreModelService

The StoreModelService class implements StoreModelServiceInterface. This class maintains all data in the Store Model Service software system, including registered Customers, Products that can be sold, and Stores in the system.

The StoreModelService is implemented as a Singleton so that only one may exist for managing all Stores and Customers in the Store Model Service.

Methods
All methods described in this table are implementations of the interface defined in StoreModelServiceInterface. As such, the signature for these methods is not defined here as it is already defined above.

Method Name	**Description**
createStore	Creates a Store within the Store Model Service
getStore	Gets a Store based on its unique identifier
createAisle	Creates an Aisle in a particular Store
getAisle	Gets an Aisle from a specific Store
createShelf	Creates a Shelf in a particular Aisle
getShelf	Gets a Shelf from a specific Aisle
createInventory	Creates an Inventory for tracking the quantity of a particular Product on a Shelf
getInventory	Gets an Inventory to examine its contents
updateInventory	Updates the number of items in an inventory, the number of items remaining must be between 0 and the maximum capacity for the inventory

createProduct	Creates a new Product for which Inventories can exist
getProduct	Gets the information about a Product
createCustomer	Creates a customer account for tracking purchases, movements, and payments. Locations will be empty by default until the customer position is updated.
updateCustomerLocation	Updates the location of a Customer
getCustomer	Gets a Customer based on their identifier
getBasketForCustomer	Gets the Basket owned by a particular Customer
addToBasket	Adds an instance of a Product to a Basket
removeFromBasket	Removes an instance of a Product from a Basket
clearBasket	Removes all items from a Basket
getBasket	Gets a Basket based on its identifier
createSensor	Creates a Sensor in a Store for tracking activity
getSensor	Gets a Sensor by its identifier
handleSensorEvent	Simulate a Sensor event occurring
createAppliance	Create an Appliance in a Store for tracking activity and for performing tasks
commandAppliance	Command an Appliance to perform a task

Attributes
No attributes are stored by the StoreModelService.

Associations

Property Name	**Type**	**Description**
stores	Map<string, Store>	This field contains a list of all Stores registered in the Store Model Service. Stores are accessed via a unique identifier string.
productsAvailable	Map<string, Product>	This field contains a list of all Products that can be sold in a Store. Products are accessed by a unique identifier string.
registeredCustomers	Map<string, Customer>	This field contains a list of all Customers registered to the Store Model Service. Customers are accessed by a unique identifier string. These customers may be

		shared with the Customers owned by a Store. This Map contains a subset of the values in allCustomers below.
allCustomers	Map<string, Customer>	This field contains a list of all Customers, registered or not. Customers are accessed by a unique identifier string. These customers may be shared with the Customers owned by a Store.

Store

The Store class represents a physical store in the Store Model Service. This class maintains acts as a static container for Aisles and Shelves, and dynamically keeps track of Customers who are located in the Store. It also maintains a list of Sensors and Appliances with which it can interact. Finally, the Store is able to maintain an inventory of items that is updated as customers and Robotic Assistants add and remove products from the shelves.

Stores contain an identification string which must be globally unique.

Note, while the Requirements document (pg 4) states that Stores must have “One or more Aisles and Shelves”, a change was made to allow for a Store to own 0 Aisles, and an Aisle to own 0 Shelves. This was done to avoid confusion with default parameters for these associations when a new Store is created; users of the StoreModelService should be aware of all Aisles and Shelves created because all are done manually.

Methods

Method Name	**Signature**	**Description**
createInventory	(identifier:string, aisle:string, shelf:string, capacity:int, currentCount:int, productid:string) → void	Creates an Inventory for a product if one does not already exist. The temperature of the Product must match the Temperature of the Shelf. Upon failure, StoreModelException will be thrown.
updateInventory	(identifier:string, changeInInventoryCount:int) → void	Updates the count in an inventory. Upon failure, StoreModelException will be thrown
createSensor	(identifier:string, name:string, type:ESensorType, aisle:string) → void	Creates a Sensor in the Store. If the identifier is already in use, a StoreModelException is thrown

getSensor	(identifier:string) → Sensor	Finds a Sensor by its identifier. If none exist, a StoreModelException is thrown
handleSensorEvent	(identifier:string, event:string) → void	Handles a Sensor event. If no such device exists a StoreModelException is thrown
createAppliance	(identifier:string, name:string, type:EApplianceType, aisle:string) → void	Creates an Appliance in the Store. The sensorType field of the Appliance will be set to ESensorType.Appliance. This field is required in the base class (Sensor).
commandAppliance	(identifier:string, command:string) → void	Commands an appliance to perform a task. If no such device exists a StoreModelException is thrown.

Attributes

Property Name	Type	Description
identifier	string	A unique identification string
name	string	The name of the store. This string does not have to be unique
address	string	The address of the store. While intuitively it may make sense to ensure this field is unique, it is not required to be so in the Requirements document (pg 4) and therefore does not need to be unique.

Associations

Property Name	Type	Description
aisles	Map<String, Aisle>	This field contains a list of all Aisles in a store. Aisles are indexed by their Aisle number
sensors	Map<string, Sensor>	This field contains a list of all Sensors that exist in a Store. Sensors are identified by a globally unique device ID string
productInventoryMap	Map<string, Inventory>	This field contains a list of all Inventories for the Products available in the Store. Each Inventory can be accessed by the identification string.

		Products that are supported by the StoreModelService but are not sold by this Store are not populated. This is to allow the independent creation of Products and Inventories, as described in the Requirements document (pg 7-8).
customers	List<Customer>	This field contains a list of all Customers in the Store. The Customer data is shared with the StoreModelService for registered customers so that the service can continually track Customer locations.

Aisle

The Aisle class represents a physical aisle in a Store. A store may not have two Aisles with the same aisle identifier. This identifier is therefore used for identification.

Methods
No explicit methods are necessary for the Aisle class.

Attributes

Property Name	**Type**	**Description**
identifier	string	The Aisle identifier. This must be unique within a Store.
name	string	The name of the Aisle
description.	string	A description of the Aisle, typically including what kind of food is found there.
locatedOnFloor	boolean	True if the Aisle is located on the store floor, False if the Aisle is located in the storage room

Associations

Property Name	**Type**	**Description**
shelves	List<Shelf>	This field contains a list of all Shelves in an Aisle. Shelves are indexed by their Shelf

		number

Shelf

The Shelf class represents a physical shelf in an Aisle. An Aisle may not have two Shelves with the same shelf identifier. This identifier is therefore used for identification.

Methods
No explicit methods are necessary for the Shelf class.

Attributes

Property Name	**Type**	**Description**
identifier	string	The Shelf identifier. This must be unique within an Aisle.
name	string	The name of the Shelf
level.	EShelfLevel	Values can be: ○ Low ○ Medium ○ High
description	string	A description of the Shelf, typically including the types of food it holds
temperature	ETemperature	Values can be: ○ Frozen ○ Refrigerated ○ Ambient ○ Warm ○ Hot If no value is specified, Ambient is assumed

Associations
No associations exist for the Shelf class.

Sensor

The Sensor class represents a device in a Store capable of sensing information from the physical world. Examples can be microphones or cameras, but can also be complex Appliances (defined further below) which can be commanded.

Methods
No explicit methods are necessary for the Sensor class.

Attributes

Property Name	Type	Description
identifier	string	A globally unique identifier for the Sensor device, such as a Serial Number
name	string	The name of the Sensor
aisleLocation	string	The Aisle at which the Sensor resides
sensorType	ESensorType	Values can be: o Microphone o Camera o Appliance If value is Appliance, it is assumed the Sensor is truly of type Appliance and further information about the Appliance type can be found in the applianceType attribute.

Associations
No associations exist for the Shelf class.

Appliance

The Appliance class represents a device in a Store capable of sensing information from the physical world as well as interacting with it. Examples can be the robotic assistants, turnstiles, or speakers. Because the Appliance class shares much of its behavior with the Sensor class, the Appliance class extends the Sensor class in an effort to reduce duplication of code.

Methods
No explicit methods are necessary for the Appliance class.

Attributes

Property Name	Type	Description
applianceType	EApplianceType	Values can be: o Speaker o Robot o Turnstile

Associations
No associations exist for the Appliance class.

Inventory

The Inventory class represents a stock of a Product on a Shelf in a Store. As humans and robots interact with the store, items will be added and removed from the shelves. The Inventory class is a way to maintain a count of an individual type of Product on an individual Shelf in the system so that it can know when stock is low.

Note, while the Requirements (pg 4) call out an attribute “location” of type string which would be formatted as store:aisle:string, these fields were split up to facilitate individual operations based on the Store or Shelf independently.

Methods

Method Name	Signature	Description
updateItemCount	(changeInInventory:int) → void	Updates the number of items in the inventory. If outside the range of 0 to capacity, a StoreModelException is thrown

Attributes

Property Name	Type	Description
identifier	string	A globally unique identification string for this Inventory
store	string	The Store for which the Inventory is tracking
aisle	string	The Aisle for which the Inventory is tracking
shelf	string	The Shelf for which the Inventory is tracking
capacity	int	The number of items that can be placed on this Shelf

currentCount	int	The current number of items on this Shelf
productid	string	The identification of the Product for which the Inventory is tracking

Associations
No associations exist for the Inventory class.

Customer

The Customer class represents an individual who wishes to shop in a store managed by the Store 24X7 System. The term wishes was used because Customers may be registered or unregistered, but only registered customers may actually purchase items from the Stores. Registered customers are tracked in the StoreModelService as well as the Store in which they are currently located, while unregistered customers are only tracked in the Store they are in. While the Requirements document (pg 5) specifies that Customers may be either children or adults, no distinction was made in either the commands for creating accounts nor in the required attributes of the Customer. For this reason, it has been determined that while a Customer may represent either a child or adult patron, the Store Model Service has no reason to track the information.

Methods

Method Name	**Signature**	**Description**
getBasket	() → Basket	Gets the Customer's Basket

Attributes

Property Name	**Type**	**Description**
identifier	string	A globally unique identification string for this Customer
firstName	string	The given name of the Customer
lastName	string	The surname of the Customer
registered	boolean	True if the Customer is registered with the StoreModelService
emailAddress	string	The email address of the Customer
accountAddress	string	The identifier of the Customer's payment account

currentStore	string	The last known Store which the Customer entered, or an empty string if the Customer has never been seen
currentAisle	string	The last known Aisle in which the Customer was located, or an empty string if the Customer has never been seen
timeLastSeen	LocalDateTime	The time at which the Customer was last seen by a Sensor in a Store, or LocalDateTime.MIN if the customer has never been seen.

Associations

Property Name	**Type**	**Description**
basket	Basket	The virtual Basket of the customer

Basket

The Basket class represents a virtual shopping basket for a Customer. The Basket is automatically assigned to a Customer when the Customer account is created. As the Customer removes items from the Shelves, the same items are added to their Basket so that when they go to check out, their account can be deducted the appropriate amount.

Methods

Method Name	**Signature**	**Description**
addToBasket	(productid:string, count:int) → void	Increments the count for a Product in the basket. If no count exists, a new key is added to the map.
removeFromBasket	(productid:string, count:int) → void	Decrease the count for a Product in the basket. If no count exists, a StoreModelException is thrown
clearBasket	() → void	Clears the productCountMap

Attributes

Property Name	**Type**	**Description**

identifier	string	A globally unique identification for the Basket
productCountMap	Map<string, int>	A list of all products in the basket and a count of how many of each exist. The key for the map is the productid so that there is no need to store the entire Product information.

Associations
No associations exist for the Basket class.

Product

The Product class contains all relevant information for a product that could be sold in the Store Model Service. Examples of this information include the cost of the unit, the type of Shelf on which it must be stored, and its size.

Methods
No explicit methods exist for the Product class.

Attributes

Property Name	**Type**	**Description**
identifier	string	A globally unique identification for the product
name	string	The name of the product
description	string	A description of the product
size	int	The size of the product. This may be a weight in pounds or a volume in gallons, whichever is larger
category	EProductCategory	Values can be: o Produce o Dairy o Deli o FrozenMeat o Etc.
cost	int	The cost of the product in units

temperature	ETemperature	The temperature at which the Product must be stored. Values can be: o Frozen o Refrigerated o Ambient o Warm o Hot If no value is specified, Ambient is assumed

Associations
No associations exist for the Product class.

CommandProcessor

The CommandProcessor class is a utility used for simulating inputs to the Store Model Service from various actors. For a complete list of commands available, see the Requirements document (pg 7-9).

Methods

Method Name	**Signature**	**Description**
processCommand	(command:string) → void	Processes an individual command and interacts with the Store Model Service appropriately
processFile	(fileName:string) → void	Processes a file of commands and interacts with the Store Model Service appropriately for each

Attributes
No attributes exist for the CommandProcessor class

Associations

Property Name	**Type**	**Description**
model	StoreModelServiceInterface	This member in reality will be the full StoreModelService. However, by defining this member as an interface

		the possibility of dependency injection arrises to allow for other non-standard behaviors for testing or other purposes.

Implementation Details

While there are many individual classes in this design and seemingly endless multiplicity of associations, the overall implementation is not a complex one. Generally, the StoreModelService is the master of all interactions. This means that any case in which two classes interact must somehow trace back to the StoreModelService commanding it to be so. Errors in these interactions are also handled by the StoreModelService, which can choose either to mitigate the failures or to pass the Exception up to its caller.

Because this system single-threaded, little care need be given to race conditions or performance optimizations. Commands are read from the test script as quickly as the host machine allows, and the time to process these commands within the Store Model Service is imperceptible.

Exception Handling

There are two types of Exceptions in the system.

The first type of Exception is the CommandProcessorException. This exception type is thrown by the CommandProcessor when a command is read that does not match the expected format or command type. One example of this would be a typographical error swapping the word “create” with “craete". The CommandProcessor would not recognize this command and would therefore be forced to throw an Exception. Errors that would not lead to exceptions include additional whitespace and extra characters at the end of a line because both of these errors can easily be handled without obscuring the intent of the command.

The CommandProcessor exception has three fields: a string “reason” which contains the reason for the failure, a string “explanation”, which provides more details as to what the failure was, and an integer “lineNumber” which is 0 or the line in a script which caused the failure.

The second type of Exception in the system is the StoreModelException. This Exception is used to indicate any data failure in the Store Model Service. Examples of this kind of data failure include attempting to re-use a unique identifier, incrementing an Inventory beyond its capacity, placing a Product on a Shelf with a different temperature setting, and more.

The StoreModelException has two fields: a string “reason” which contains the reason for the failure, and a string “explanation”, which provides more details as to what the failure was.

Testing

Testing of the StoreModelService is done using the CommandProcessor utility. To test the system thoroughly, a script is written which sets up various scenarios and tests that the results are valid. This strategy is similar to that of an integration test, in which the boundaries of the component are the StoreModelServiceInterface API calls and returns.

A simple example of this testing is shown below:

- show store “Store1”
  - This should fail because no Store has been created yet
- define store “Store1” name test address “test address”
- show store “Store1”
  - This should pass now that the Store has been defined
- show store “NewStore”
  - This should still fail

Risks

The biggest system risk with the design as it is currently is that the Sensor and Appliance classes do not yet have any functionality. Sensor events Appliance commands are currently treated as opaque strings and therefore have no functionality in the software system. In order to improve the system towards a truly self-sustaining Store system, this functionality must be defined and implemented.