Inventory Design

Maxim Noah Khailo

November 13, 2015

1 Purpose

An inventory system provides a way to determine if a SKU is in stock.

2 Forces

There are several forces that the architecture of the inventory system has to balance.

- Connectability to WMSes.
- Connectability to storefronts
- Filtering and grouping by SKUs
- Filtering and grouping by Catalog information
- Managing an audit trail of inventory changes.

The design here will provide a space to balance these forces.

3 Concepts

This section provides several concepts which can be combined to provide the functionality of the inventory system.

3.1 Identifiable

Something you can assign an id to. This is used for equality checking, filtering, and search.

```
type Id = String
trait Identifiable {
    def id(): Id
}
```

3.2 SingularUnit

Something where all values are of a single unit.

```
type UnitType = String
trait SingularUnit {
    def unit(): UnitType
}
```

3.3 Location, ShippingRestriction

A trait for filtering based on shipping. This needs to be flushed out more based on Jeff's shipping restrictions work.

```
type Location = String
trait Locatable {
    def location() : Location
}

trait HasShippingRestrictions {
    def shippingRestrictions(): Seq[ShippingRestrictions]
}
```

3.4 OnHand

Something which can be counted using some units, for example a SKU. The setAmount function is used by a WMS for updating. This function can also be a good place to implement audit trail.

```
trait OnHand extends Identifiable with SingularUnit{
    def onHand(): Int
}
```

3.5 Handy

A collection of On Hand things. For example, a Sku

```
trait Handy {
    def findOnHandById(id: Id): Result[OnHand]
}
```

3.6 Holdable

An holdable is a trait which modifies adjusts the on hand amount without mutating the original amount.

```
trait Holdable extends Identifiable with SingularUnit {
    def held() : Int
}
```

3.7 Holdables

A collection of held things. For example our Inventory system.

```
trait Holdables {
    def findHoldableById(id: Id): Result[Holdable]
}
```

3.8 Reservable

A reservable is something that goes out of availability.

```
class Reservable extends Identifiable with SingularUnit{
    def reserved() : Result[Int]
}
```

3.9 Reservables

A collection of reservable things.

```
trait Reservables {
    def findReservableById(id: Id): Result[Reservable]
}
```

3.10 NonSellable

A non sellable is something that isn't available yet. For example, boxes arrived in loading dock for a sku.

```
class NonSellable extends Identifiable with SingularUnit{
    def nonSellable() : Result[Int]
}
```

3.11 NonSellables

A collection of non sellable things.

```
trait NonSellables {
    def findNonSellableById(id: Id): Result[NonSellable]
}
```

3.12 InventoryItem

A inventory item combines several concepts into one that provides the various values that make up the inventory calculation.

```
trait InventoryItem extends
    OnHand with
    Reservable with
    Holdable with
    NonSellable{}
```

3.13 InventoryItems

A collection of inventory items.

```
trait InventoryItems extends
    OnHand with
    Reservables with
    Holdables with
    NonSellables {

    def findInventoryItemById(id: Id): Result[InventoryItem]
}
```

3.14 Adjustments

Adjustments are designed to model the ledger which is used to modify inventory items. The adjustments are applied by a warehouse to inventory items.

```
object Adjustments {
    sealed trait Adjustment
    case class OnHand(id: Id, amount: Int) extends Adjustment
    case class Held(id: Id, amount: Int) extends Adjustment
    case class Reserved(id: Id, amount: Int) extends Adjustment
    case class NonSellable(id: Id, amount: Int) extends Adjustment
    case class And(left: Adjustment, right: Adjustment)

    type Ledger = Seq[Adjustment]
}
```

3.15 Warehouse

A Warehouse is a set of inventory items. It has a location and shipping restrictions. It can return inventory items and also apply adjustments to the inventory.

The implementation to this will integrate with a 3rd party WMS and our own inventory tracking database.

```
trait Warehouse extends
  Identifiable with
  Locatable with
  HasShippingRestrictions with
  InventoryItems {

      def apply(adjustments: Adjustments.Ledger) : Result[Unit]

  }
```

3.16 WarehouseManagementSystem

This is to model the integration with a WMS. At the highest level it supports an update process which is automatic or manual. It also returns a sequence of warehouses the WMS manages.

```
trait WarehouseManagementSystem extends Identifiable {
    def warehouses() : Seq[Warehouses]
    def update(Warehouse): Future[Result]
}
```

4 Models

Given the concepts above we can define some models like SKU, and WMS

4.1 InventorySku

An inventory Sku represents an inventory item. It also computes available for sale of a sku in a particular warehouse.

```
case class InventorySku( item: InventoryItem, restrictions: ShippingR
    HasShippingRestrictions with InventoryItem {

    def id() = item.id()

    def availableForSale(): Int =
        math.max(item.onHand() - item.reserved() - item.held(), 0)

    def onHand() = item.onHand()
    def held() = item.held()
    def reserved() = item.reserved()
    def nonSellable() = item.nonSellable()
    def unit() = item.unit()
    def shippingRestrictions() = restrictions
}
```

4.2 Inventory

An inventory is a collection of warehouses. It will implement the algorithm that finds skus in warehouses and computes availble for sale.

```
class SearchContext(
    l: Option[Location],
    w: Option[Id],
    f: InventoryItem => Boolean) {
        val location = l
```

```
        val warehouseId = w
        val filter = f
}

case class Inventory(inventoryId: Id, warehouses: Seq[Warehouse]) {
    def id() = inventoryId

    def findInventoryItems( id: Id, c: SearchContext)(implicit ec: Exe
        warehouses.map{_.findInventoryItem(id)}
    }

    def findShippingRestrictions(id: Id)(implicit ec: ExecutionContext
        Result.good(Seq.empty) //TODO: Implement

    def findSkus(id: Id, c: SearchContext)(implicit ec: ExecutionConte
        val restrictions = Seq.empty
        for {
        items      findInventoryItems(id, c)
        } yield items flatMap {
            case Xor.Right(item)       Result.good(InventorySku(item, r
            case Xor.Left(failures)       Result.failures(failures)
        }
    }
}
```

4.3 Discussion

The idea behind the inventory model is to model the half object of the Sku. WMSes do not keep track of held items from a storefront.

If we trust the on hand amount, we can only hold it on the other side via a holdable. The other concept is a reservable and this is to model the order reserve transaction which would send the reserve to a WMS.

We can then have implementations of a warehouse that polls from a WMS creating cached inventory items in a DB or somewhere else which regularly updates. We can also implement an holdable that gets its value from a DB stored by the inventory system. Combining the cached and held value gives you the SKU and amount for sale.

Filtering based on shipping restrictions from and to. We can also do more fancier things like filtering based on attributes for display purposes.

Notice that you cannot simply find one SKU in the inventory system, but it returns multiple SKUs as a SKU can be in multiple warehouses.

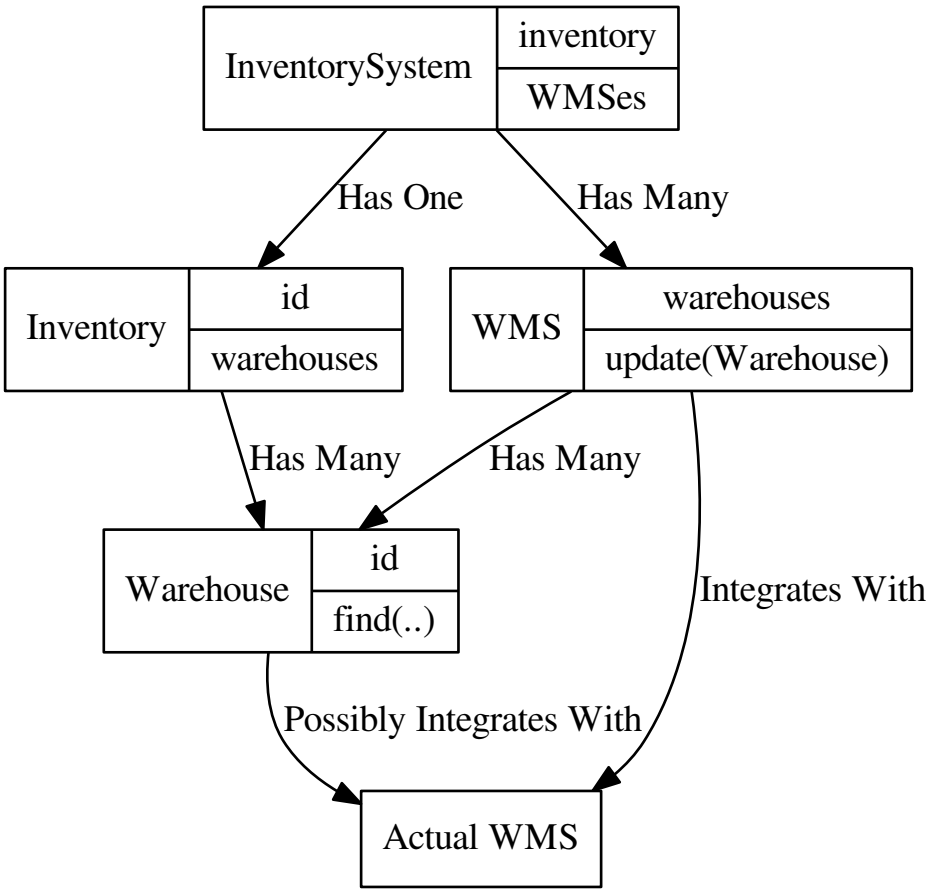
AFS becomes an accumulation over the SKUs

5 Architecture

The inventory system will be composed of an inventory and a set of WMSes.

```
class InventorySystem(i: Inventory,
    ws: Seq[WarehouseManagementSystem]) {

        var inventory = i;
        val wmses = ws

        def afs(sku: Id) = //stuff
        def afs(sku: Id, c: SearchContext) = //stuff
}

class InventorySystems(syss: Seq[InventorySystems]) {
    def systems = syss

    def findSystem(id: Id) = //gets system by id
}
```

5.1 Diagram



6 TODO

- Todo, flush out shipping restrictions
- Figure out how to incorporate catalog and filtering