Mobile Application Store Collection Service Design Document

Date:
Author: Lisa "Lex" Loren
Reviewers: Frank O'Connor

Introduction

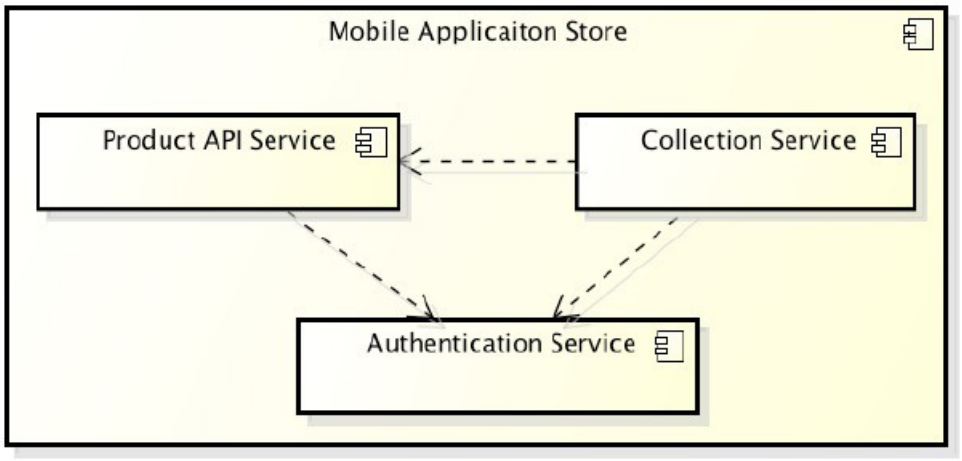
This document defines the design for the Mobile Application Store Collection Service.

Overview

The Mobile Application Store provides developers and content producers with a way to distribute downloadable applications, wallpapers, and ringtones to consumers. The store makes both free and premium content available for download.

The Collection Service offers administrators an easy to way to offer targeted collections of content for download. It makes the Mobile Application Store easier for consumers to navigate, and provides them with new ways of finding content.

The Collection Service relies on the Product API for some information. Although the Collection Service is only loosely coupled to the Product API Service, queries and calls for product information must be executed on the Product API at runtime.



[Figure: Relationship between the Product API Service and the Collection Service.]

Requirements

The Collection Service should support two types of Collections: Static and Dynamic. Both types of Collections should be able to contain other Collections as sub-collections.

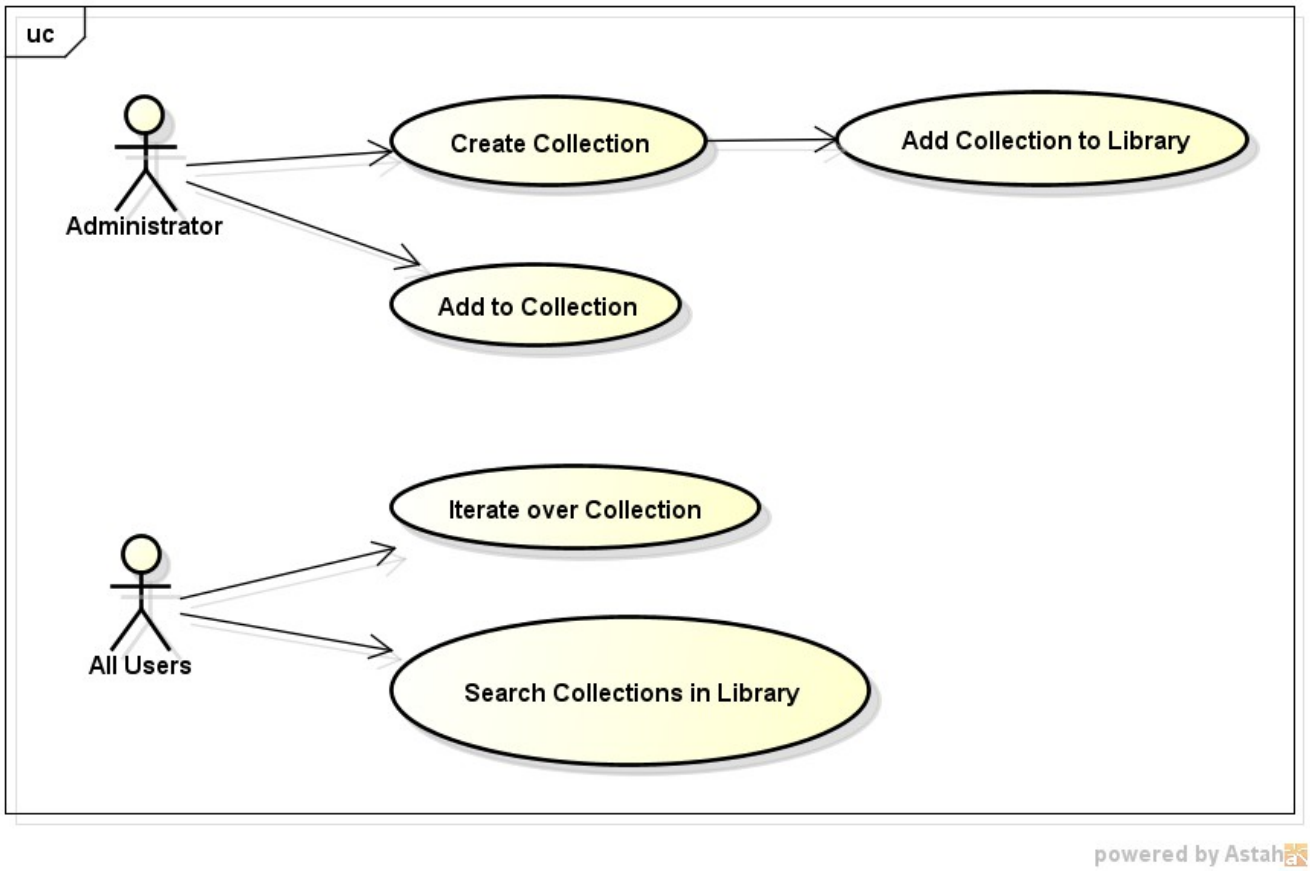
Static Collections should contain lists of product IDS, and return references to corresponding products.

Dynamic Collections should contain a query. When asked for their contents, they should execute the query and return the results.

Administrators should be able to create new Collections, and add to them. (This should be a restricted interface.)

The Collection service must offer all users a way to search Collections, and to iterate through a Collection and any sub-collections it may have.

Use Cases



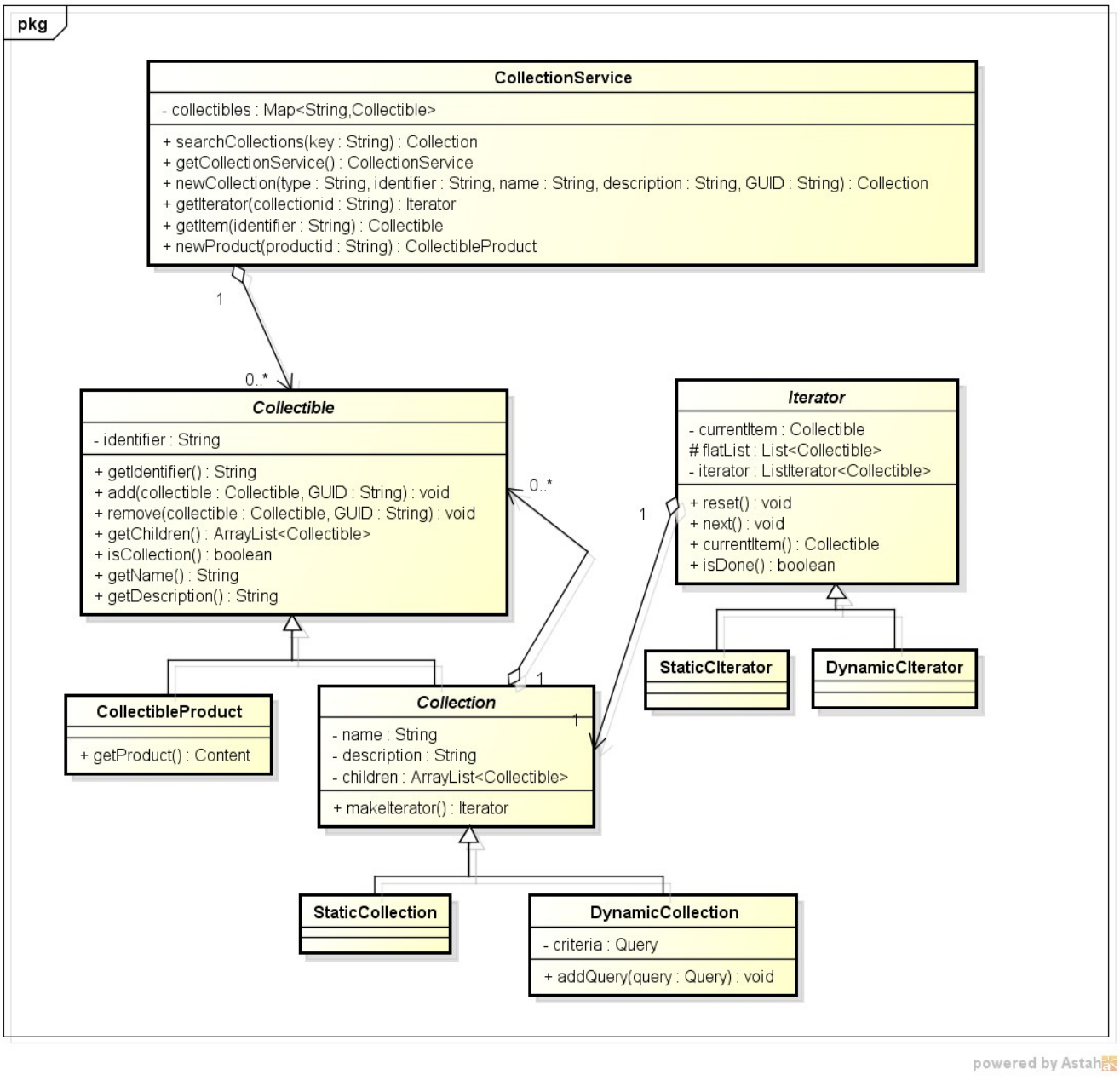
[Figure: Administrators and Users interact with the Collection Service.]

Administrators create Collections, which are then added to the Collection library. (See diagram below.)
Administrators add products to Collections.
All users (administrators/consumers) search the Collection library.
All users iterate over a particular collection, receiving a reference to each product contained within it (and its sub-collections).

Implementation

Class Diagram

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



[Figure: class diagram for the Collection Service.]

Class Dictionary

CollectionService

The CollectionService is a singleton object, holding collections in memory. It features two examples of the Factory pattern. The newCollection method creates a Collection subtype suitable to the information it is passed. The getIterator method creates an Iterator subtype specific to the Collection matching the identifier it receives. Throws a CollectionException.

Method	Signature	Description
newCollection	(type: String, identifier : String, name : String, description : String, GUID : String) : Collection	Factory method for creating collections. If the type field is "dynamic, this will become a Dynamic Collection. If it is "static", it will become a Static Collection. Throws a CollectionException.
getCollectionService	() : CollectionService	Returns the CollectionService Singleton.
searchCollections	(key : String) : Collection	Returns a Collection containing all collections with a name or description matching the search key.
getIterator	(collectionid: String) : Iterator	Returns an Iterator suitable for traversing the Collection with an identifer matching the input String. If the input is an empty String, returns an Iterator which traverses the entire Collection.
getItem	(identifier : String) : Collectible	Returns the Collectible with an identifier that matches the given identifier.
newProduct	(productid : String) : CollectibleProduct	Creates a new CollectibleProduct from the given productid. Thriws a CollectionException if an item with given ID already exists.

Association Name	**Type**	**Description**
collectibles	Map<String, Collectible>	Map of all current Collectibles, indexed by identifier.

Collectible

The Collectible class is an abstract class embodying the composite pattern. Some Collectible subtypes

can have child Collectibles. Others can be children, but not parents.

Method	Signature	Description
getIdentifier	() : String	Returns the identifier of the Collectible.
add	(collectible : Collectible, GUID : String) : void	Adds the specified collectible as a child of the current collectible. Throws an exception if the parent cannot accept children.
remove	(collectible : Collectible, GUID : String) : void	Removes a child collectible. Throws an exception if the specified collectible is not a child of the current collectible.
getChildren	() : ArrayList<Collectible>	Returns a list of child collectibles. Note that, unlike an Iterator, this method does not obscure the tree structure of children.
isCollection	() : boolean	Returns true if the collectible is a collection - if it can have other collections added to it. Returns false if the collectible does not support children.
getName	() : String	Returns the name of the Collectible.
getDescription	() : String	Returns the description of the Collectible.
addQuery	(query: Query) : void	If the collectible is a Dynamic Collection, sets this query as its criteria. Throws a CollectionException.

Property Name	**Type**	**Description**
identifier	String	A unique identifier for the collectible.

Collection

The Collection class is an abstract class which can contain other Collectible types.

Method	Signature	Description
makeIterator	() : Iterator	Returns an Iterator for the Collection.

Property	Type	Description
name	String	The name of the collection.
description	String	The description of the collection.
children	ArrayList<Collectible>	All children of the collection.

StaticCollection

The StaticCollection class is a subclass of the Collection class. It contains a static list of Collectibles. Its makeIterator methods creates a StaticCIterator instance.

DynamicCollection

The DynamicCollection class is a subclass of the Collection class. It contains a Query object, called at runtime, which returns a list of products. A DynamicCollection object can reference children of type Collection, but no other Collectibles. Its makeIterator methods creates a DynamicCIterator instance.

Association Name	Type	Description
criteria	Query	The Query object to be searched for.

CollectibleProduct

A collectible-type wrapper for a product ID. (This corresponds to the "identifier"of the collectible class.) This Collectible cannot have children. The add(), remove(), getChildren(), and addQuery() methods inherited by this class all throw CollectionExceptions.

Method	Signature	Description
getProduct	() : Content	Returns a product with an ID match the identifier.

Iterator

The Iterator is an abstract class designed to let users move through a Collection in an organized fashion. When initialized, an Iterator subclass converts the collection's children into a flat List of collectibles. During initialization, it will create the Iterators of any child collections to add their products to it list.

Method	Signature	Description
next	() : void	Sets currentItem() to the next item in the list.
currentItem	() : Collectible	Returns the current item.

isDone	() : boolean	Returns false if there are more items to iterate through.
reset	() : void	Resets the iterator.

Association Name	**Type**	**Description**
currentItem	Collectible	The current Content item or Collection.
flatList	List<Collectible>	A list of all children and grandchildren.
iterator	ListIterator<Collectible>	Used by the iterator to move through the list of children and grandchildren.

StaticCIterator

The StaticCIterator class is a subclass of the Iterator class specialized for StaticCollections.

DynamicCIterator

The DynamicCIterator class is a subclass of the Iterator class specialized for DynamicCollections. When initialized, it runs the DynamicCollection's Query and converts results to CollectibleProducts. which are returned along ith any child collections.

Implementation Details

All data having to do with Collections is stored in-memory, in a singleton instance of the CollectionLibrary class. The CollectionLibrary provides methods for searching collections and for adding collections. Collections are stored in a HashMap, with the product ID serving as key, so that they can be easily referenced by other Collections.

Individual collections are subclasses of the Collection class, and come with iterators that are subclasses of the Iterator class. Any type of collection can contain child collections, which in turn can be of any Collection type.

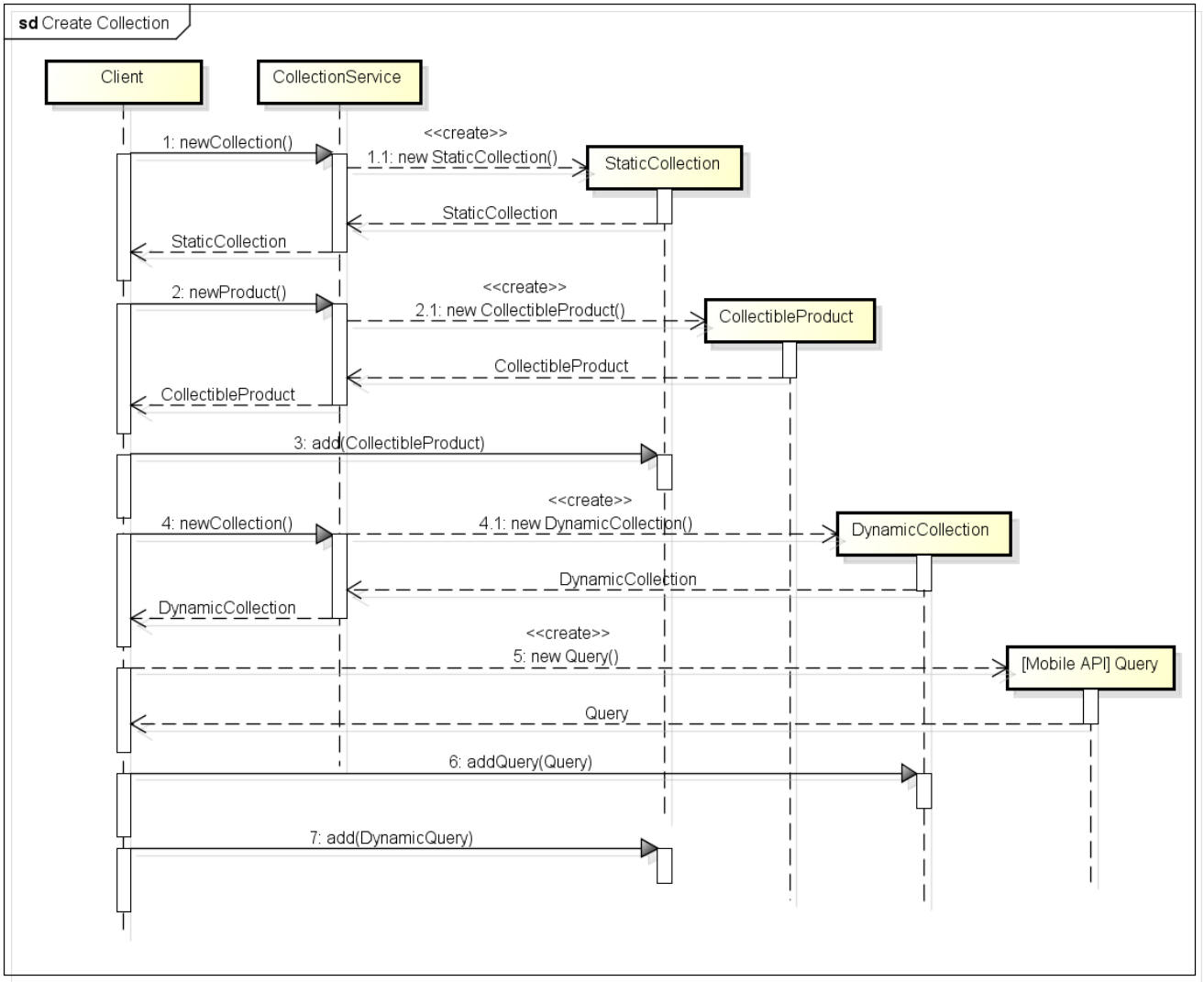
Use of Factory Pattern:
The newCollection() method of the CollectionService class implements the factory pattern. It accepts a String argument, type, which determines which subclass of Collection is constructed.

Use of Composite Pattern:
The Collectible class and its subclasses implement the composite pattern. An object of the collection subclasses, StaticCollection and DynamicCollection, may contain any number of Collectible objects, including other instances of their own classes.

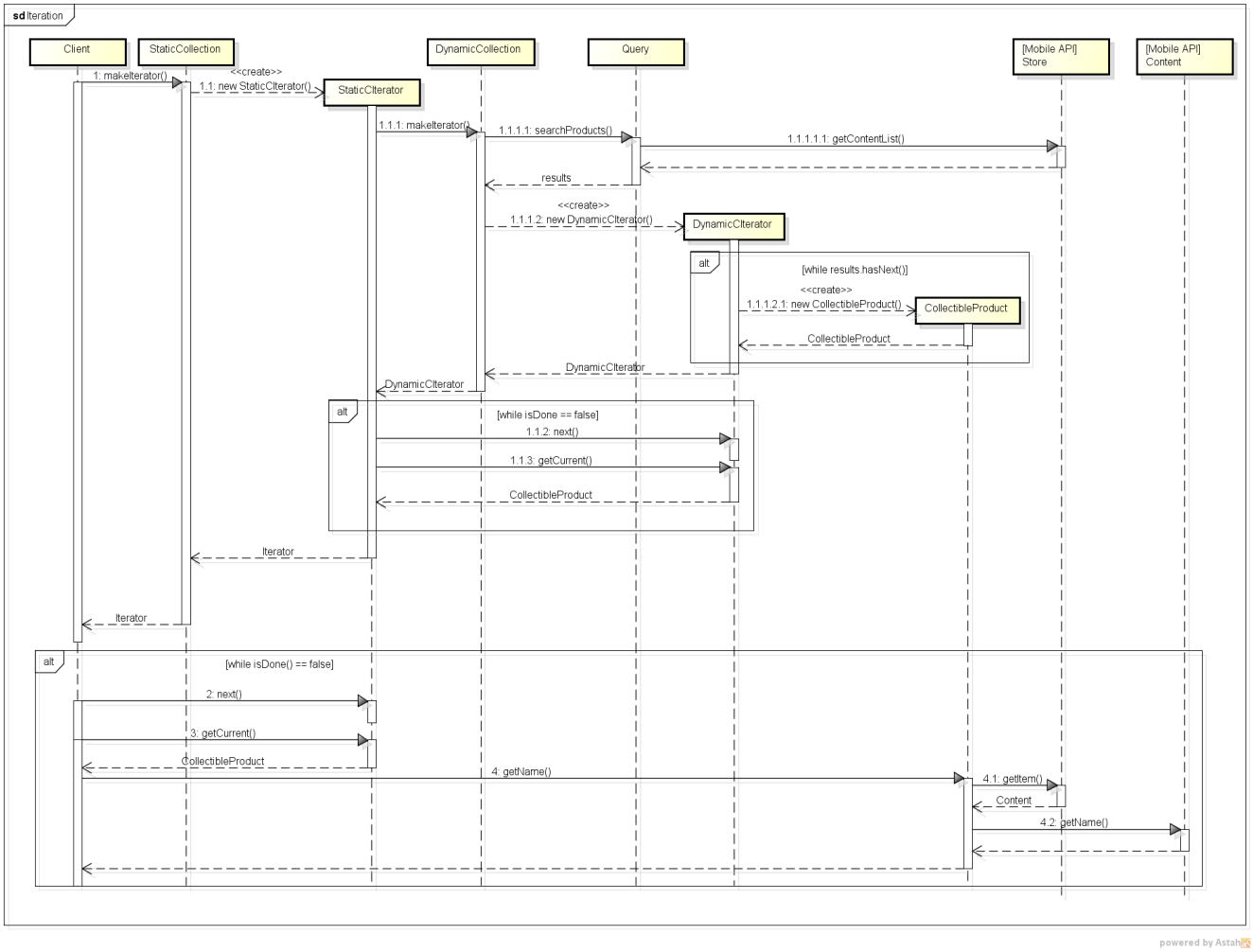
Iterator Pattern:
The Iterator class subclasses, StaticCIterator and DynamicCIterator, implement the iterator pattern. More specifically, they allow for depth-first iteration, and vastly simplify the process of exploring the collection for end users.

State Transtion Diagrams

The following diagrams illustrate the way the Collection Service interact with the Product API.



[Figure: This illustrates the process of creating a Static Collection, adding a CollectibleProduct to it, and creating and adding a Dynamic Collection to it.]



[Figure: Building off the previous diagram, this shows the process od creating an Iterator that encompasses both Static and Dynamic Collections. It then shows iteration, complete with calls to the Product API for product information.]

Testing

Usage: java TestDriver countries.csv devices.csv products.csv collections.csv

The testing suite for this service consists of three classes: TestDriver, Importer, and ImportException. TestDriver primarily handles file I/O and client output, while Importer communicates more directly with the Collection Service.

Note: products.csv is very similar to the one provided, but has been converted to ASCII. The provided products.csv contained hex characters that interfered with Java's Scanner.

Risks

The Authentication Service is as-yet unimplemented. Implementation will require alterations to the bodies of many methods, as well as additional error handling.