CSE 460/598 Programming Assignment

Computer Science Program

Arizona State University, Tempe, AZ

Kranthi Kumar Rekala

[krekala@asu.edu](mailto:krekala@asu.edu)

Spring 2015

**Assumptions**

This type of Model is generally used when the communicating parties are unaware of each other namely Publisher and Subscriber. The Following are some of the key assumptions:

* The delivery of messages is generally not guaranteed in a Pub Sub Model, However, here we assume that there is no such problem
* The messages are delivered synchronously between Broker-Sub and Pub-Sub
* The Publisher and Subscriber are unaware of each other and Broker communicates with each of them using appropriate events.

1. Publisher-subscriber

Brief description

* The Publish/Subscribe pattern structure is an enhancement to the Observer pattern structure we have discussed in the class, but it has several important differences.
* In Observable patterns, the Observer structure is a part of service which has some state and based on it Observer candidates would receive notifications.
* With publish/subscribe pattern, subscribers register with an intermediate service named Broker and are responsible for delivering specialized events. Publishers also deliver events using sane service.

There are many implications of this structure, including the following:

* Subscribers can subscribe a single time, yet they can receive events from many publishers.
* This technique, called anonymous publishing, is useful for systems that do not have a static set of objects responsible for events in the system.
* Publishers do not have to re-create event publishing and subscriber tracking mechanisms for each class that delivers events.
* The application and system can change around an event service without a change to the event service itself.

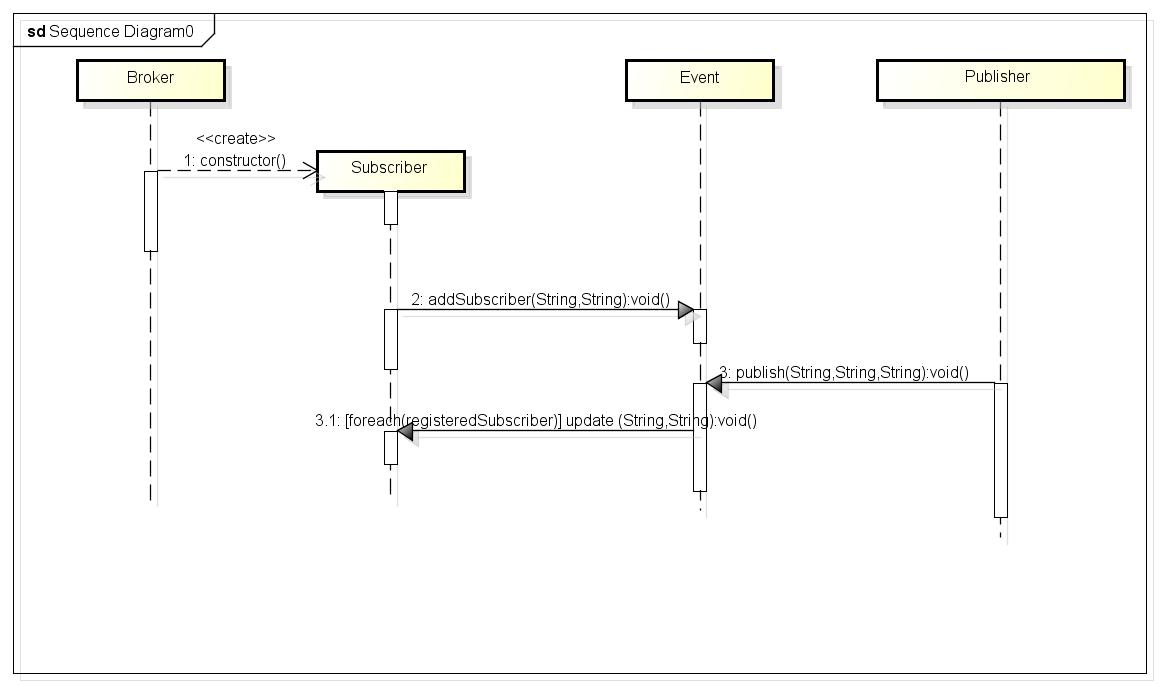
1. Class diagrams

The Approach of breaking the problem into the PubSub model gives us a clue that the communication should happen between Pub and Sub with the help of a broker who

Acts as a middle-man. Also, there are some set of specified events generally (Publish, Subscribe) in our case that would trigger subsequent methods defined. Basically, the communication happens with the help of dispatching and adding/removing events.

Hence , as per my discussion above there are 4 Java classes in my code Publisher, Subscriber, Broker and Main class named PubSub. An Event interface is defined as per the pattern and this is "realized" by both Publisher and Subscriber so as to use and implement the Event class methods. All the local variables of each class are declared as private and the methods are defined as public .The class diagram drawn as part of his project will give us a much clear picture.

1. Sequence diagrams

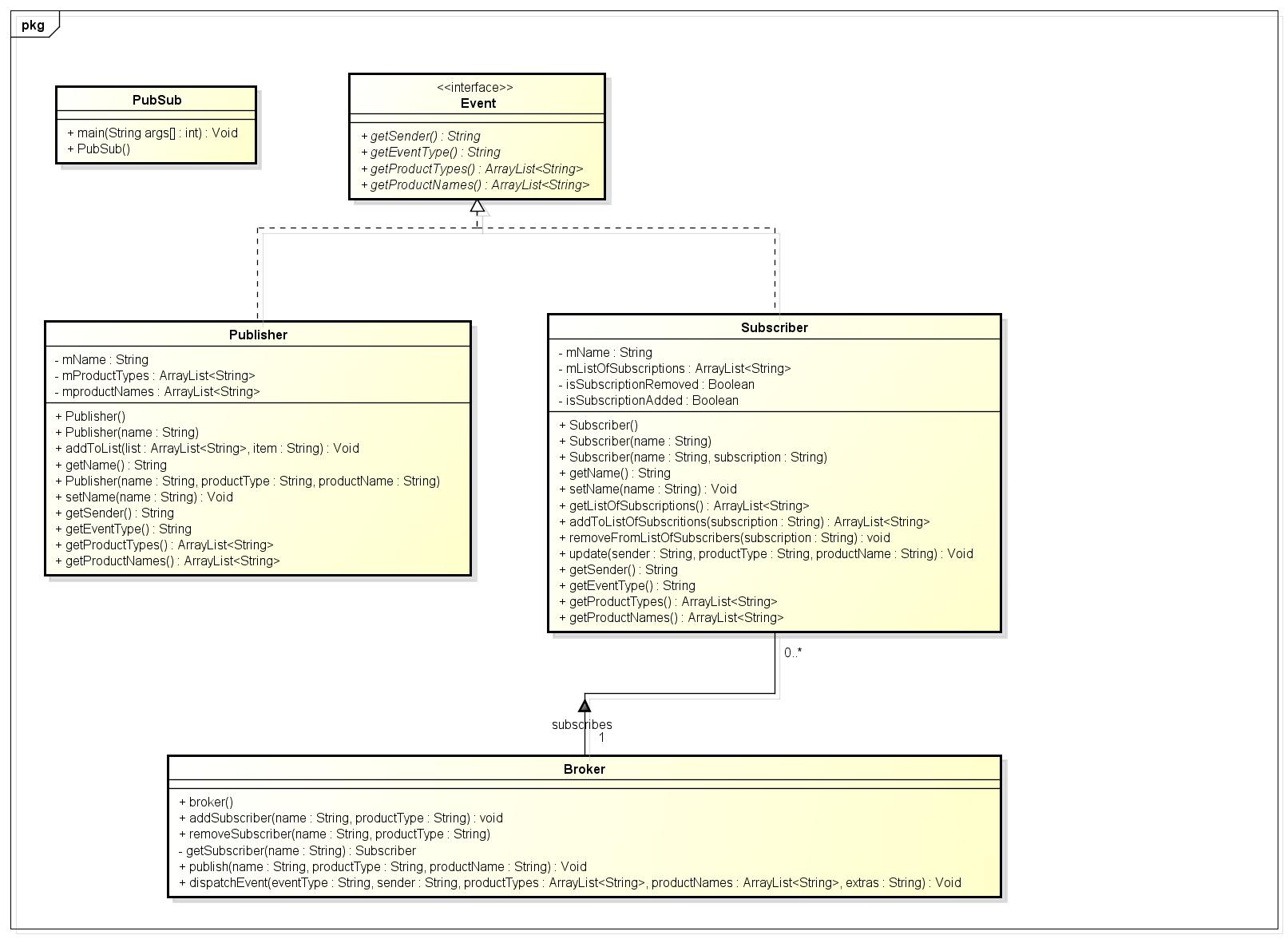


The collaborations between the components in the Publish/Subscribe pattern are nearly similar to the collaborations in the Observer pattern. A client registers a subscriber with an event service. As part of the registration through the addSubscriber operation on the event service, the client identifies the events that they want to receive through a string-based identifier for the subscriber Web Service. Once registered, the subscriber receives events with a topic that matches the identifier.

Publishers create the data surrounding an event and publish the event through the publish operation on the event service and deliver the message. Once delivered, the event service searches through the subscribers to locate all of those interested in the event and delivers the event to the subscribers.

The set of Events are laid down in the sequence diagram as per the ordering of the Events.

1. Forward engineering



The Above class diagram was developed after a couple of iterative improvements to the design. Using the above class diagram which confirms to the PubSub Model and Astah software I have generated the code for this design. It gave me all the required methods and variables mentioned in the diagram with their proper return types as well. Also, the realization of the class to interfaces are also given by Astah.

The Sequence Diagram was used in order to design the flow of Events and the subsequent Actions

1. Brief description

The Following is a brief description on how the process of subscription, un subscription and publishing happens, there are lot of details that can be included, but I am providing a concise description of the functioning of my model

Subscribe:

The process of subscribe starts once the input set of Events has a "subscribe" string in it. Subsequent tokens as per my design are passed as arguments to the subscribe constructor via the broker methods dispatch events method which in turn initialize the values of the subscriber and the product to which it’s subscribed. Multiple subscriptions are allowed. The value of the product to which this subscriber is subscribed is maintained in a list called mListOfSubscriptions and when a publish event is encountered, all the subscribers which confirm to the Event are notified. All the subscribe events are dispatched by the broker to the

Unsubscribe:

Unsubscribe has a similar working as above, but the values will be removed from subscriber list to avoid notifying of subsequent publish events

Publish:

The process of publish starts once the input set of Events has a "publish" string in it.The Subsequent tokens as per my design are passed as arguments to the publish constructor via the broker methods dispatch Events which initialize the values of the publisher and their subsequent values. These values are looked up in the list of subscriptions for that particular products and the availability of them are published onto the console.

1. Readme

PubSub.java is the name of Main Class

Appendix

Astah files and completed Java source code.

Include any other supporting material you may have.

References:

<http://flylib.com/books/en/4.10.1.105/1/>

<http://wikipedia.com>

[http://www.youtube.com/PubSub design pattern](http://www.youtube.com/PubSub%20design%20pattern)