Process MeNtOR 3.o

Uni-SEP

Pub-Sub System

Design Document

Version:	1.0
Print Date:	11/1/2005 8:54 AM
Release Date:	
Release State:	Initial/Core/Final
Approval State:	Draft/Approved
Approved by:	
Prepared by:	Abigail Garces, Fahreen Bushra, Firas Aboushamalah, Wafiq Syed
Reviewed by:	
Path Name:	
File Name:	T3-ReqmtsModel.doc
Document No:	

Document Change Control

Version	Date	Authors	Summary of Changes
	4/4/19	F.A./W.S.	Added Class Diagrams
	4/4/19	A.G.	Added Architecture
	4/5/19	F.B.	Added Pseudocode Modifications to Diagrams
	4/6/19	F.A./W.S./A.G./F.B.	Formatting

Document Sign-Off

Name (Position)	Signature	Date
Abigal Garces	A.G.	April 7/19
Fahreen Bushra	F.B.	April 7/19
Firas Aboushamalah	F.A.	April 7/19
Wafiq Syed	W.S.	April 7/19

Contents

1 Introduction

1.1 Overview

In this project, a prototype publish-subscribe (pub-sub) system will be implemented which allows for the creation of channels on a system composed strictly of subscribers and publishers. Simply, the publisher posts an event on a shared medium (the channel), which is inherently accessible by the subscriber depending on their relationship to the channel. If they are subscribed to the channel, they will receive a notification that an event has been posted, and may exclusively handle the event based on the state associated with them. Similarly, the channel which the publisher posts messages to is determined solely on their strategy. In this document, the resources and their references are cited accordingly.

The second sections deals primarily with the design decisions that were made throughout the process of implementing the system. This section aims to describe in detail the significant design choices and modularization criteria that are attributed to the pub sub system. The third section strictly highlights the package level component diagram of the system. This diagram provides an overview of the crucial factions of the system and how they interact with each other dynamically. The last section of the document is detailed with class diagrams of components that were altered in the process of assembling the system. These classes include: AbstractEvent, IState, IStrategy and AbstractChannel. The diagrams contain the UML Notations and a table for each class detailing its respective attributes and methods.

1.2 Resources - References

http://agilemodeling.com/artifacts/classDiagram.htm

https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/

http://www.gatherspace.com/software-requirement-specifications/

https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern
https://docs.oracle.com/cd/B10501_01/appdev.920/a96590/adg15pub.htm

2 Major Design Decisions

Major Design decisions

Majority of the software code was provided. Below is an overview of the code that was implemented and modified, along with the design patterns observed.

Singleton Design pattern

By implementing a singleton design pattern, the system is able to contain reference points to the unique objects that are the collection of AbstractChannel type entities and the methods provided which manipulate them. Because the singleton design pattern ensures that the ChannelPoolManager class is limited to the number of objects it can create. However, by implementing the design patterns in the following classes: ChannelAccessControl, ChannelCreator, ChannelDiscovery, ChannelEventDispatcher and SubscriptionManager, we are able to expand on the unique object by subclassing without specifically altering the client program. This helped simplify the structure of the program, allowed the creation of a specific number of objects and provided a more secure implementation of the data transported.

Factory Method Design Pattern

The Factory Method design pattern is followed in classes PublisherFactory, StrategyFactory, SubscriberFactory, StateFactory, and EventFactory. The Factory Method design pattern creates an instance of any of the subclasses derived from the same Abstract class. To specify an example, the factory method pattern was followed to implement the event factory (I1). All types of EventType extend the AbstractEvent class. EventTypeC was created and accordingly, EventType was modified to include TypeC as one of the enum data type variables. All instances are created through the “Factory,” which creates a concrete instance based on the condition of the call made to the factory. For instance, StrategyFactory has the public method createStrategy(StrategyName strategyName), and will create a specific strategy based on the strategy name. This is achieved via a switch case.

Publisher and Subscriber id

In order to distinguish between Publisher and Subscriber objects, the object’s hashcode is used as a unique reference.

Access.java

In the pubSubServer, an Access class was created in order to provide public access to protected/private methods. The Access.java class provides public access to the following methods:

- ChannelPoolManager.getinstance()
- ChannelEventDispatcher.getinstance()
- ChannelAccessControl.getinstance()

This allows the methods of the classes listed above to remain protected, while providing third-party access.

3 Architecture

The software architecture style employed in this system is the Publish Subscribe pattern. The publishers publish categorized events into classes and subscribers receive notifications of posted events in the channel they've subscribed to.

Below is the package level component diagram of the system.

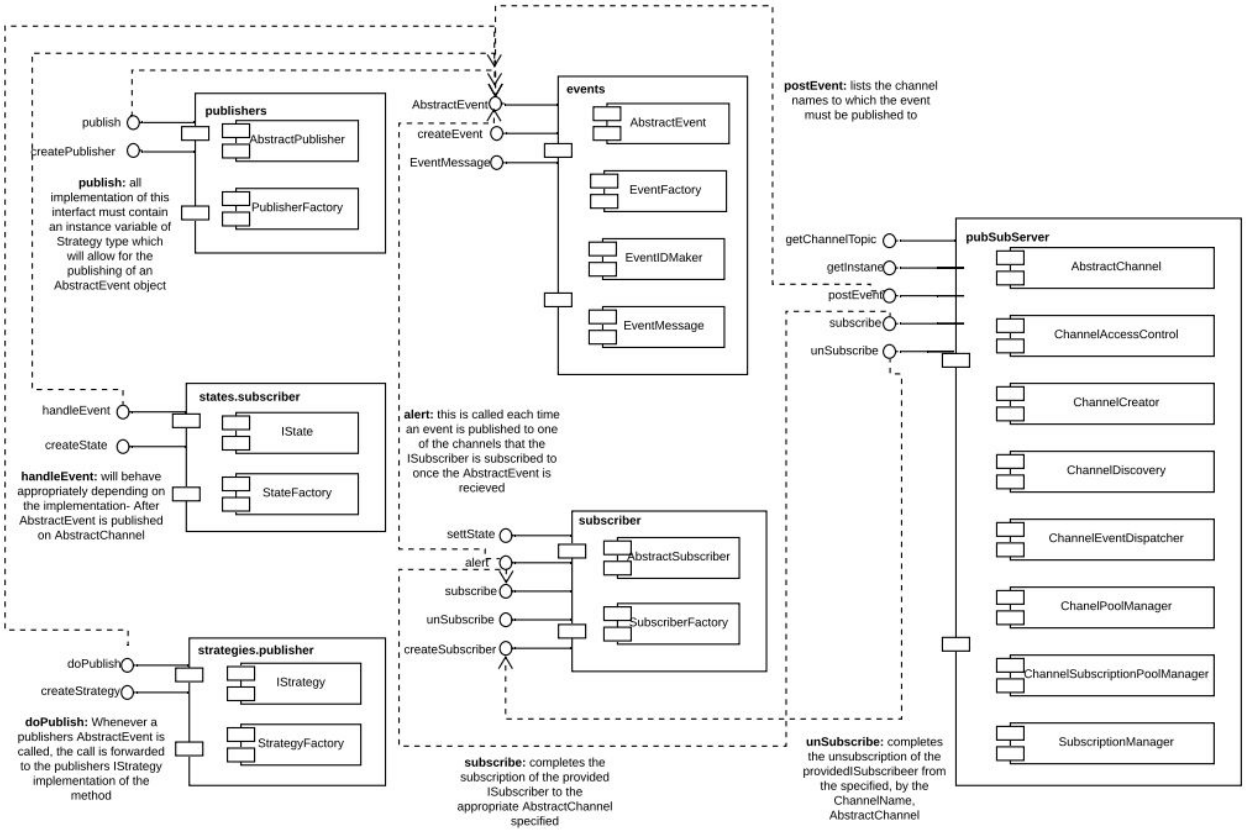

Figure 1. Component Diagram.

4 Detailed Class Diagrams

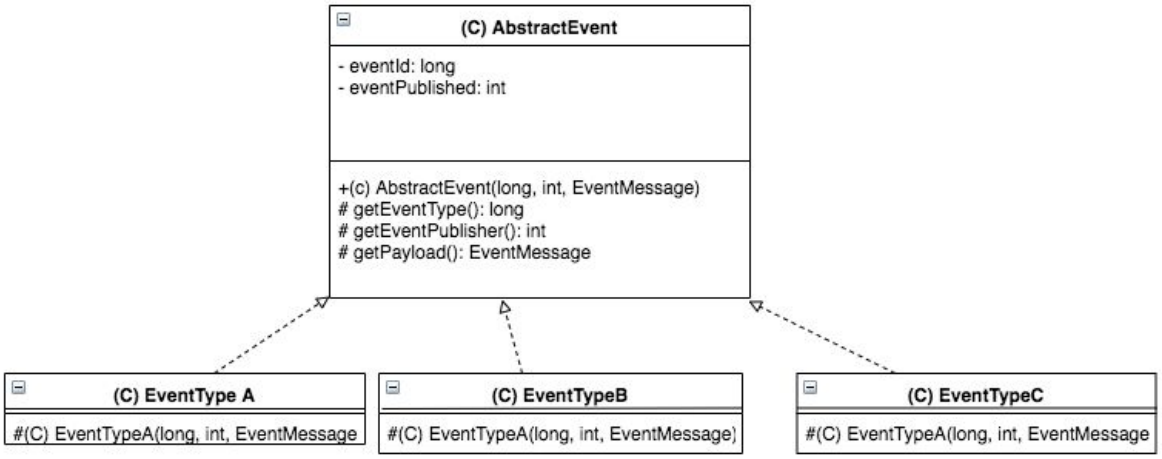

Figure 2. Class Diagram for I1.

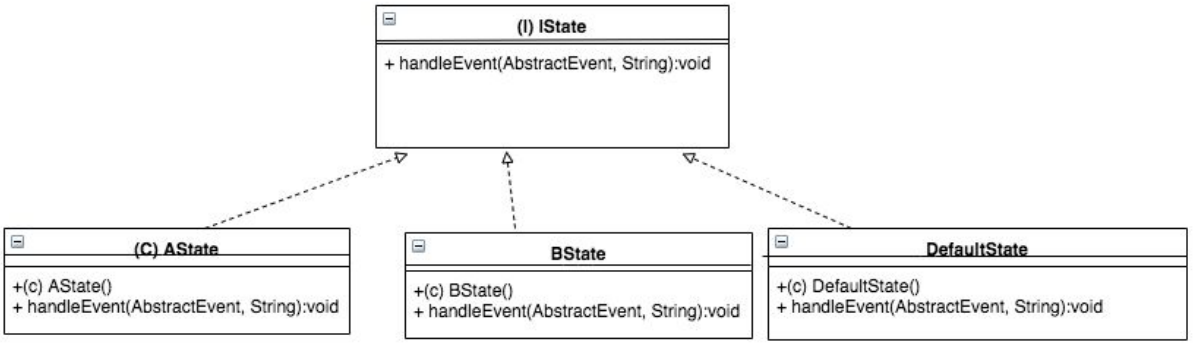

Figure 3. Class Diagram for I2.

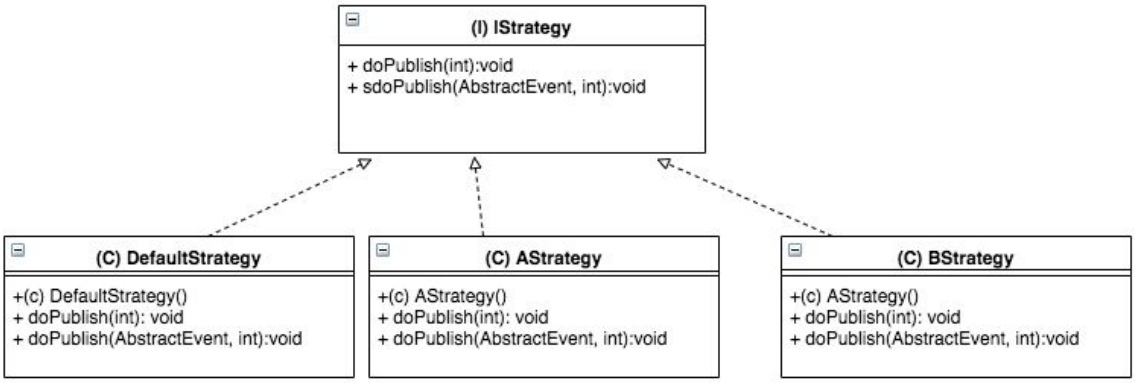

Figure 3. Class Diagram for I4.

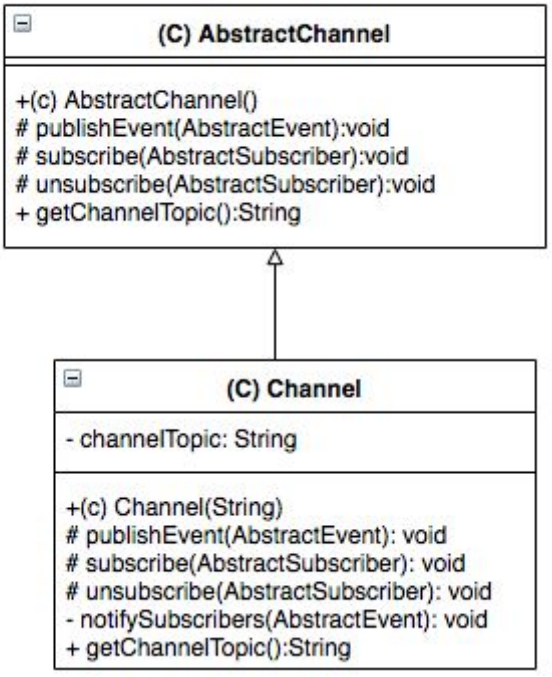

Figure 4. Class Diagram for I5.