CS2212B - Meeting Minutes (Group 1)

Date:	Duration:	Members:	Meeting Summary:
Jan.16	1 hr	Seif, Pouya, Sarah, Din, Tyler	Group introduction.
Jan. 23	2 hrs	Seif, Pouya, Sarah, Din, Tyler	Familiarize ourselves with software requirements and other related documents. Made a general plan on overall execution strategies. Specific roles assigned to complete SRS.
Jan. 30	2.5 hrs	Seif, Pouya, Sarah, Din, Tyler	Work session. Each person is working on their assigned section to complete SRS.
Feb. 06	2 hrs	Seif, Pouya, Sarah, Din, Tyler	Work session. Each person is working on their assigned section to complete SRS.
Feb. 13	2 hrs	Seif, Pouya, Sarah, Din, Tyler	Work session. Each person is finalizing their assigned section to complete SRS. Worked on case, sequence, and collaboration diagrams.
Feb. 20	1 hr	Seif, Pouya, Sarah, Din, Tyler	(Reading Week) Online work session. Group met to proof read the SRS document for a final draft.
Feb. 24	1 hr	Seif, Pouya, Sarah, Din, Tyler	Reviewed SRS.

Publish-Subscribe System
Requirements Model

Version:	1.0
Print Date:	Feb. 25, 2019
Release Date:	TBA
Release State:	Initial
Approval State:	Draft
Approved by:	-
Prepared by:	Seif, Pouya, Din, Sarah Tyler
Reviewed by:	Seif, Pouya, Din, Sarah Tyler
Path Name:	~/Group1_srs
File Name:	Group1_srs
Document No:	1

Document Change Control

Version	Date	Authors	Summary of Changes
1.0	Feb. 03/19	Seif, Pouya, Din, Sarah, Tyler	Initial Write
1.5	Feb. 24/19	Seif, Pouya, Din, Sarah, Tyler	Reviewed

Document Sign-Off

Name (Position)	Signature	Date
Seif	Seif	Feb. 24
Pouya	Pouya	Feb. 24
Din	Din	Feb. 24
Sarah	Sarah	Feb. 24
Tyler	Tyler	Feb. 24

Contents

1 Introduction

1.1 Purpose

This document details the requirements of a prototype system that is composed of publishers and subscribers. Publishers publish events to designated channels, which then notify the subscribers of the channel that an event has been posted. Subscribers can subscribe to multiple channels which each have assigned topics. The channel is filtered by the publishers with the use of topics and subjects. Topics within the channel are governed by rules and policies based on the initial system software design. Once subscribers have been notified of an event, they can choose to respond to it.

1.2 Overview

The aim of the project is to implement a prototype system which allows the publishers to generate events and post them on a shared medium (i.e. the *channel*), where many subscribers will have access to it. The publishers do not know, and do not need to know, who the subscribers of the channel are, and who consumes the published events. Publishers just generate events that are sent to a channel. The channel on which a publisher publishes its events is determined by the strategy the publisher is associated with. Similarly, subscribers subscribe to specific channels, in order to get access to the events posted in those channels. A channel knows its subscribers and *notifies* them when a new event is *posted* in the channel. Upon receipt of an event, subscribers may perform an action, depending on their *state.*

The objectives of the project include a) publishing an event; b) channel notifying the subscribers of the event being published; c) subscriber handling an event; d) controlling access to a channel; e) subscriber subscribes to a channel.

Non-functional requirements:

NFR1: The system has to be able to handle at least 100 channels

NFR2: The system has to be able to handle at least 200 publishers and 200 subscribers at various configurations

1.3 References

Eclipse:	http://www.eclipse.org/downloads/index.php
Java Server:	http://java.sun.com/j2ee/1.4/download.html#sdk
GANTT:	http://wiki.phprojekt.com/index.php/Gantt-diagram

2 Business Scenario Model

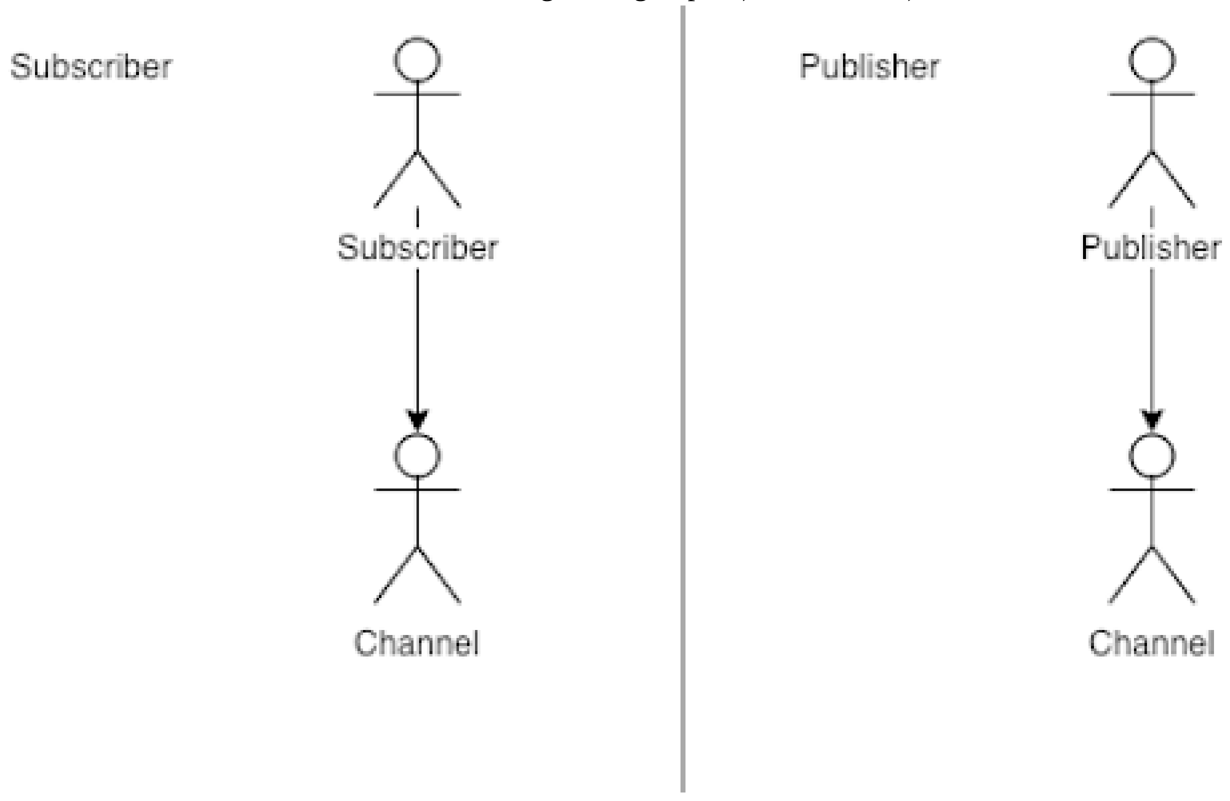
2.1 Actors

2.1.1 Overview

The actors in our system include end users of the software (subscribers), the publishers, and the channel system. The actors in our software system use a common medium (aka the channel) to communicate to one another. Subscribers exist in an environment that allows them to subscribe to multiple channels at any given time. As a subscriber, they are notified once an event has been published to the channel, assuming that the subscriber is registered to the channel and not blocked. Publisher actors have the ability to publish an event to a given channel for distribution to subscribers. In addition, the channel is able to block and unblock subscribers.

2.1.2 Actor Diagram

The figure below represents the actors in our system. Based on their interactions with the system, the actors are characterized into three general groups: a) Subscriber b) Publisher



2.1.3 Actor Definitions

Publisher

Description	An entity that publishes/posts events to one or more channels. A publisher publishes an event to a channel based on a strategy, or to a default channel if the no specific strategy is selected.
Aliases	Publisher
Inherits	None
Actor Type	Active
Contact Person	
Contact Details	

Subscriber

Description	An entity that handles events published on a channel. A subscriber is subscribing to one or more channels and is notified of published events. A subscriber has a "state" which determines how it handles an event.
Aliases	Subscriber
Inherits	None
Actor Type	Active
Contact Person	
Contact Details	

Event

Description	An entity that denotes a piece of information posted on a channel. An event has an event ID, a reference to its publisher, and a payload which is an *EventMessage* object. The *EventMessage* has a header and a body.
Aliases	None
Inherits	None
Actor Type	Active Actor
Contact Person	
Contact Details	

2.2 Use Case Descriptions

This section documents the complete business scenarios within the scope of this project.

2.2.1 XXXX-0001 Normal Operation Scenario with Channel

In this business scenario, one Publisher uses the system to publish an event to a specific channel, and the subscriber gets notified if he/she are registered to the channel and are not blocked by the publishing party.

Description:
This scenario pertains to the publishing of an event to a channel by the publishing party. This process starts off when a subscriber registers to the channel of their choice, making them part of a list of subscribers for a specific channel. Then the publisher has the ability to publish an event to one or more channels of their choice. The subscriber is then notified of a new event if they are initially registered to the channel and are not blocked by the publisher.

Actors:
The actors associated with this business scenario are as follows:

1. Publisher (Channel publisher)
2. Subscriber (Channel subscriber)

Preconditions:
Before this scenario can be performed:

1. Channel must be on and functioning.
2. Subscriber must be initially registered to the channel.
3. Publisher creates an event and publishes it to one or more channels.
4. Subscribers are notified of the new event if they are registered and not blocked by the publisher of the event.

Postconditions:

1. The channel will have performed its intended functionality.

Alternative Courses:
None.

Extends:
None.

User Interfaces:
None.

Constraints:
None.

Questions:
None.

Notes:
Every effort has been made to optimize the system by utilizing the lightest and most efficient tools and resources. No alternatives have been listed at this current time. Details for the systems use cases are provided below.

Authors:
Seif, Pouya, Din, Sarah, Tyler

2.2.2 UC1: Publishing an Event

In this use case, one Publisher publish an event using a specific strategy designated for that publisher which determines the channel that the event is published to.

Description:
This scenario pertains to the publishing of an event to a channel by the publishing party. The publisher has a designated strategy which determines the channel that the event will be published to, and the channel is created if necessary. The first variation of this scenario involves the Publisher specifying which event to publish and relies on the given strategy to determine which channel to post it to. In the second variation of this scenario, the Publisher relies on the strategy to generate the event and decide which channel to post it to.

Actors:
The actors associated with this business scenario are as follows:

1. Publisher
2. Channel

Preconditions:
Before this scenario can be performed:

1. Publisher and event must both exist
2. If channel does not exist it must be created

Postconditions:

1. There should be an event in the queue of the specified channel
2. Channel must exist (existed before is created as part of this event publication)

Alternative Courses:
None.

Extends:
None.

User Interfaces:
None.

Constraints:
None.

Questions:
None.

Notes:
Every effort has been made to optimize the system by utilizing the lightest and most efficient tools and resources. No alternatives have been listed at this current time. Details for the systems use cases are provided below.

Authors:
Seif, Pouya, Din, Sarah, Tyler

2.2.3 UC2: Channel Notifies Subscribers Event Has Been Published

In this use case, and event has been posted to the Channel and Subscribers of the Channel must be notified.

Description:
This scenario pertains to notifying Subscribers of a Channel that an event has been published. Every channel has a topic, associated with a list of Subscribers and a queue of events. A Subscriber is notified only if they are not blocked on the Channel.

Actors:
The actors associated with this business scenario are as follows:

1. Channel
2. Subscriber

Preconditions:
Before this scenario can be performed:

1. Event must be posted on a channel.
2. Channel has Subscribers.

Postconditions:

1. Subscribers who are not blocked on the Channel must be notified.

Alternative Courses:
None.

Extends:
None.

User Interfaces:
None.

Constraints:
None.

Questions:
None.

Notes:
Every effort has been made to optimize the system by utilizing the lightest and most efficient tools and resources. No alternatives have been listed at this current time. Details for the systems use cases are provided below.

Authors:
Seif, Pouya, Din, Sarah, Tyler

2.2.4 UC3: Subscriber Handling an Event

In this use case, a Subscriber handles an event based on its current state.

Description:
This scenario pertains to a Subscriber handling an event it has been notified of. The event is handled depending on its current state.

Actors:
The actors associated with this business scenario are as follows:

1. Subscriber

Preconditions:
Before this scenario can be performed:

1. Subscriber must exist.
2. Subscriber must have a state.
3. Subscriber must subscribe to the Channel the event is posted in.

Postconditions:

1. Subscriber handles event according to the logic dictated by its state.

Alternative Courses:
None.

Extends:
None.

User Interfaces:
None.

Constraints:
None.

Questions:
None.

Notes:
Every effort has been made to optimize the system by utilizing the lightest and most efficient tools and resources. No alternatives have been listed at this current time. Details for the systems use cases are provided below.

Authors:
Seif, Pouya, Din, Sarah, Tyler

2.2.5 UC4: Controlling Access to a Channel

In this use case, Subscribers are blocked or unblocked from accessing a Channel.

Description:
This scenario pertains to the blocking or unblocking of specific Subscribers from a Channel, and the mechanism for checking whether or not a Subscriber is blocked. This is handled by the Channel Access Control Module, which provides services for blocking or unblocking a subscriber and checking whether or not a subscriber is blocked. Blocking and unblocking is the responsibility of the Administration Server, which has references to the Channel Access Control Module and Channel Pool Manager Module for obtaining a list of all available channels and their subscribers. The Subscriber is blocked or unblocked and returns true if it exists or null if it does not exist.

Actors:
The actors associated with this business scenario are as follows:

1. Channel
2. Subscriber

Preconditions:
Before this scenario can be performed:

1. Channel exists

Postconditions:

1. Subscriber must be added to list of blocked subscribers of channel (blocking case)
2. Subscriber must be removed from list of blocked subscribers of channel (unblocking case)

Alternative Courses:
None.

Extends:
None.

User Interfaces:
None.

Constraints:
None.

Questions:
None.

Notes:
Every effort has been made to optimize the system by utilizing the lightest and most efficient tools and resources. No alternatives have been listed at this current time. Details for the systems use cases are provided below.

Authors:
Seif, Pouya, Din, Sarah, Tyler

2.2.6 UC5: Subscriber Subscribes to a Channel

In this use case, a Subscriber requests to subscribe or unsubscribe from a channel.

Description:
This scenario pertains to a Subscriber subscribing or unsubscribing from a designated channel. A subscribe or unsubscribe request is handled by the Subscription Manager Module, which then utilizes the subscribe and unsubscribe services of the channel to perform the operation.

Actors:
The actors associated with this business scenario are as follows:

1. Subscriber
2. Channel

Preconditions:
Before this scenario can be performed:

1. Channel exists

Postconditions:

1. Subscriber is added to list of subscribers for given channel (subscribe case)
2. Subscriber is removed from list of subscribers for given channel (unsubscribe case)

Alternative Courses:
None.

Extends:
None.

User Interfaces:
None.

Constraints:
None.

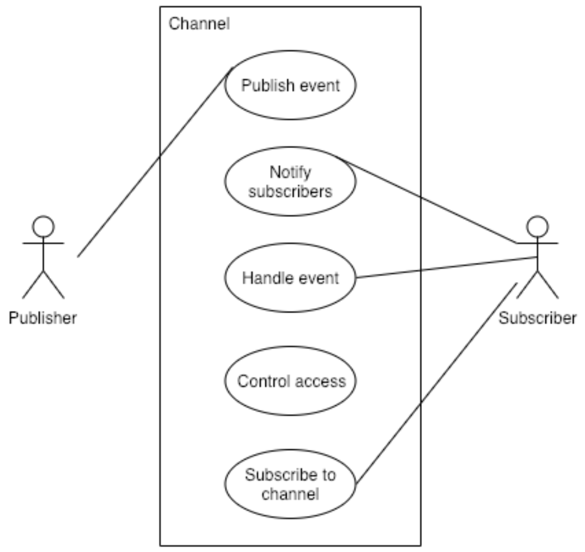
Questions:
None.

Notes:
Every effort has been made to optimize the system by utilizing the lightest and most efficient tools and resources. No alternatives have been listed at this current time. Details for the systems use cases are provided below.

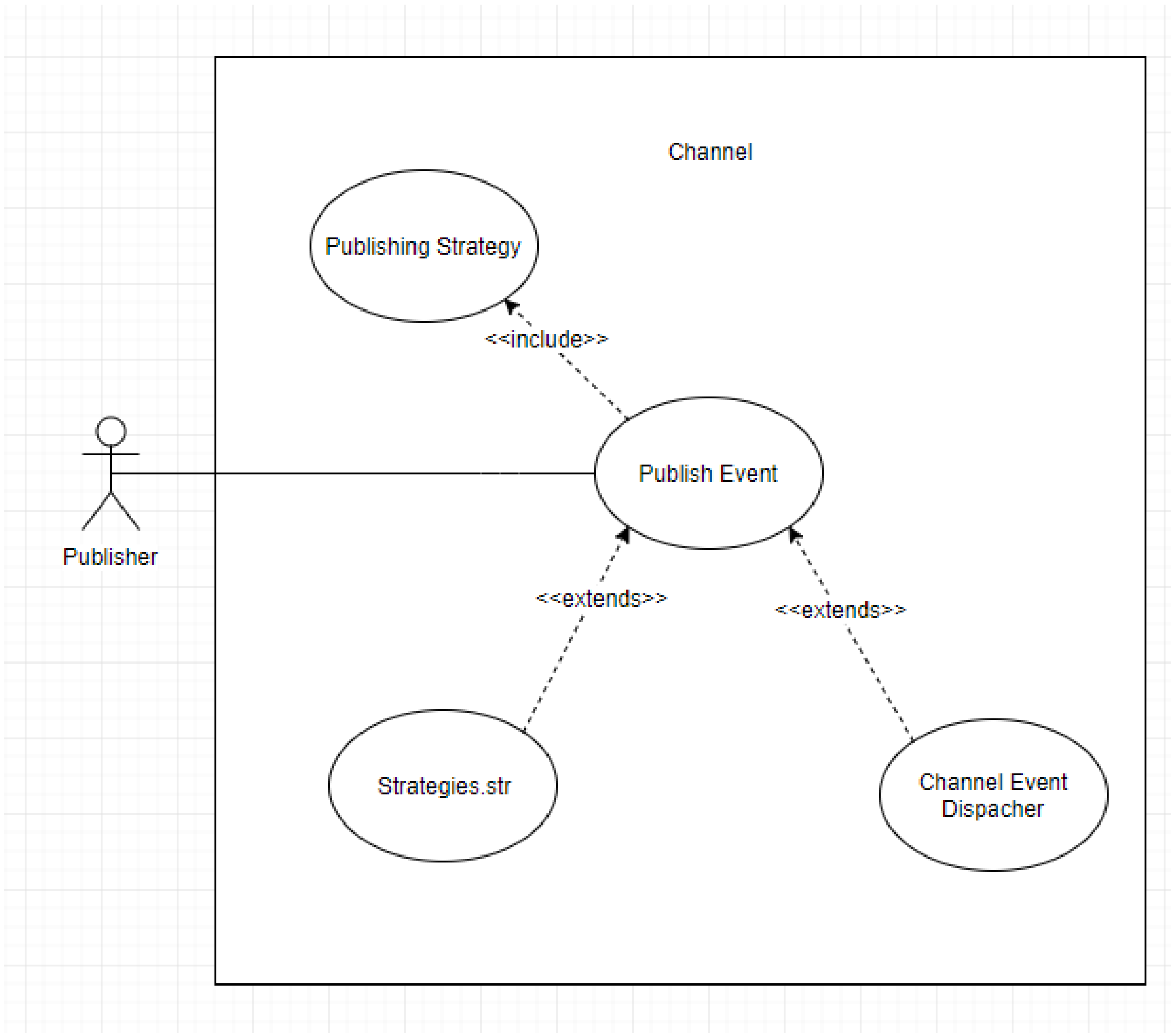
Authors:
Seif, Pouya, Din, Sarah, Tyler

2.3 Use Case Diagrams

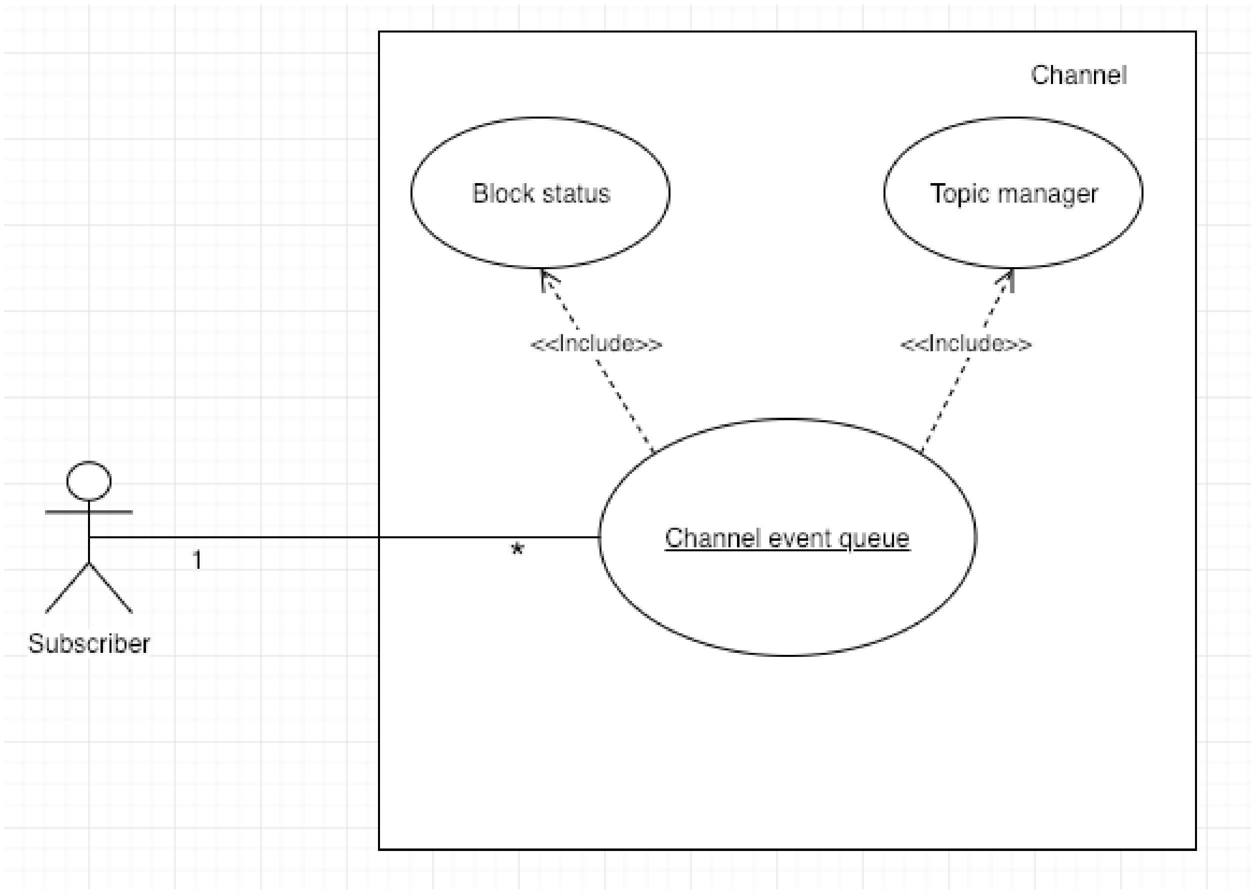
This section presents the business scenarios of the subject area in a graphical form.



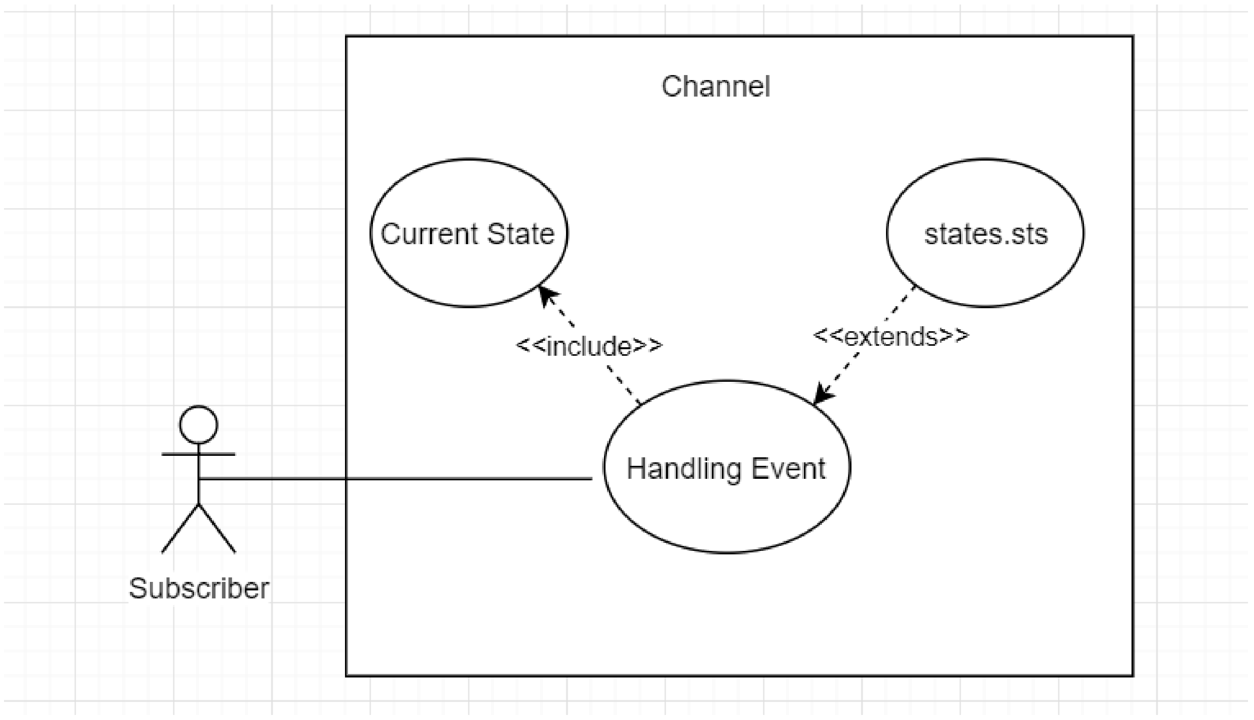
Use Case 1


Channel
Publishing Strategy
<<include>>
Publisher
Publish Event
<<extends>>
<<extends>>
Strategies.str
Channel Event Dispacher

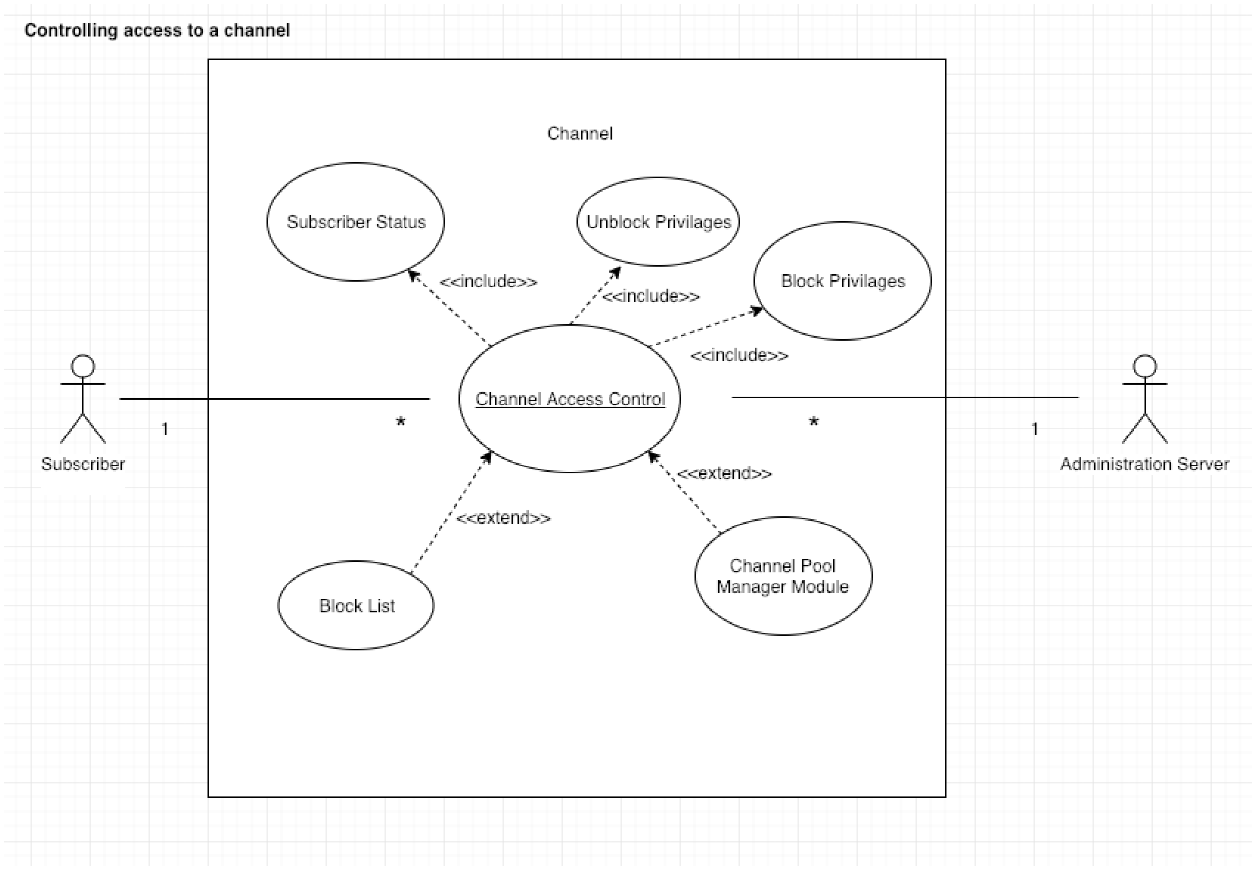
Use Case 2


Channel
Block status
Topic manager
<<Include>>
<<Include>>
Channel event queue
1
*
Subscriber

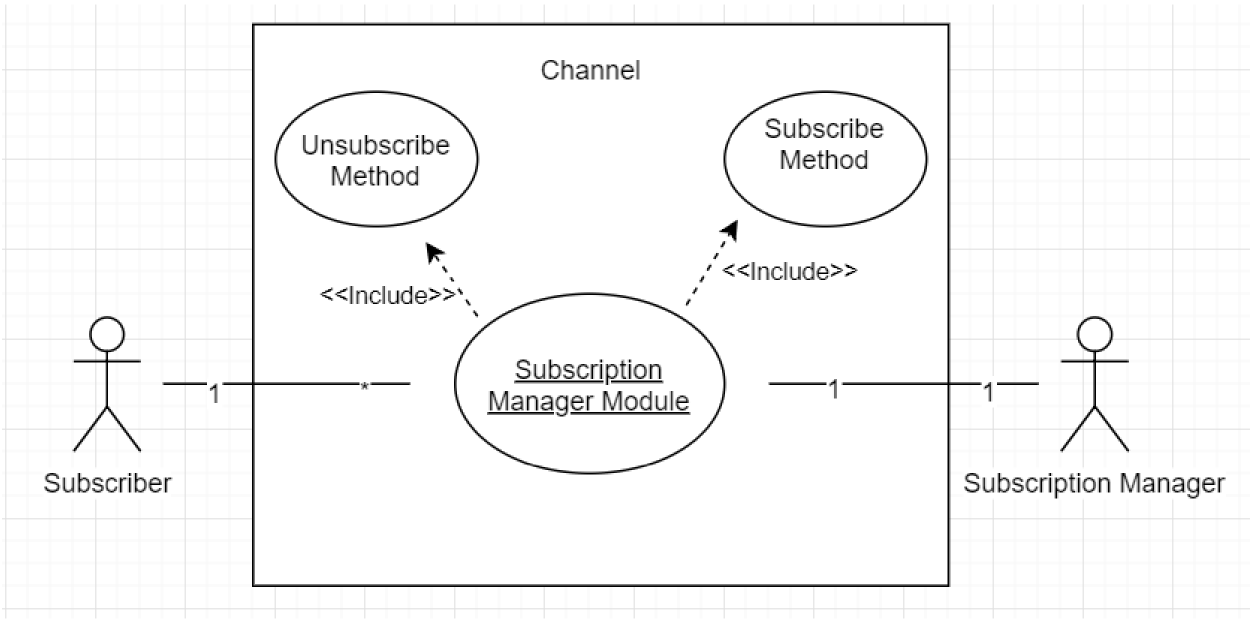
Use Case 3


Channel
Current State
states.sts
<<include>>
<<extends>>
Handling Event
Subscriber

Use Case 4



Use Case 5



3 Domain Model

3.1 Domain Model Class Diagram

The domain model class diagram for the pub-sub system appears below:

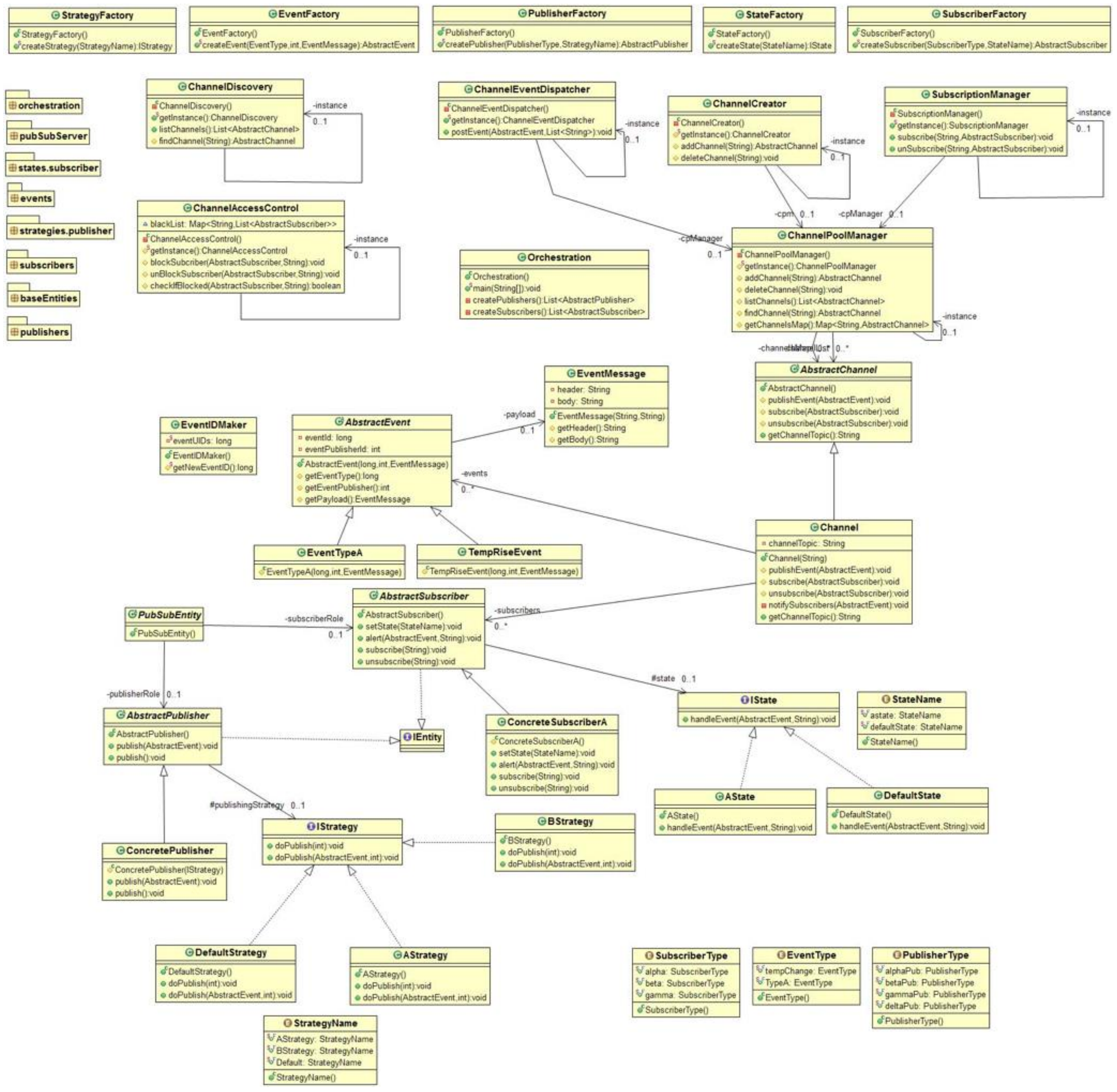


Figure 3.1 Domain Model Class Diagram

3.2 Domain Model Class Diagram

3.2.1 Channel

Description	This object represents an abstraction of a communication medium. It maintains a list of subscribers, and a queue of events. Once an event is added to the queue, the channel notifies its subscribers. There may be more than one channels in the system.
Attributes	
Sub-classes	AbstractChannel, ChannelPoolManager, SubscriptionManager, ChannelCreator, ChannelEventDispatcher, ChannelDiscovery, ChannelAccessControl
Responsibilities	This object is responsible for utilizing ChannelDiscovery and ChannelAccessControl for managing any related activity from the publisher and the events they publish.
Business Rules	

3.2.2 Subscriber

Description	This object represents an entity that handles events published on a channel. A subscriber is subscribing to one or more channels. A subscriber has a "state" which determines how it handles an event.
Attributes	
Sub-classes	AbstractSubscriber, PubSubEntity, ConceteSubscriberA, SubscriberFactory
Responsibilities	This object is responsible for setting up a subscriber class, setting state, alerting the subscriber when an event has been posted by the publisher. The subscriber also has the ability to register, subscribe, and unsubscribe to s channel.
Business Rules	

3.2.3 Publisher

Description	This object represents an entity that publishes/posts events to one or more channels. A publisher publishes an event to a channel based on a strategy, or to a default channel(s) if the no specific strategy is selected.
Attributes	
Sub-classes	AbstractPublisher, PubSubEntity, ConcretePublisher, PublisherFactory
Responsibilities	This object is responsible for setting up a publisher class and the ability to publish an event on one or more channels.
Business Rules	

3.2.4 Strategy

Description	This entity represents an abstraction of the rules that a publisher follows when publishing to a channel.
Attributes	
Sub-classes	IStrategy, DefaultStrategy, AStrategy, BStrategy, StrategyName, StrategyFactory
Responsibilities	This entity is responsible for deciding which channel an event is published to and the ability to specify rules when designing an event.
Business Rules	

3.2.5 State

Description	This entity represents a set of states for subscribers and the events pertain to them. This dictates how the channel handles a subscriber.
Attributes	
Sub-classes	IState, AState, DefaultState, StateName, StateFactory
Responsibilities	This entity is responsible for providing guidelines on the state of a subscriber. This will determine how a subscriber will handle an event created by the publisher.
Business Rules	

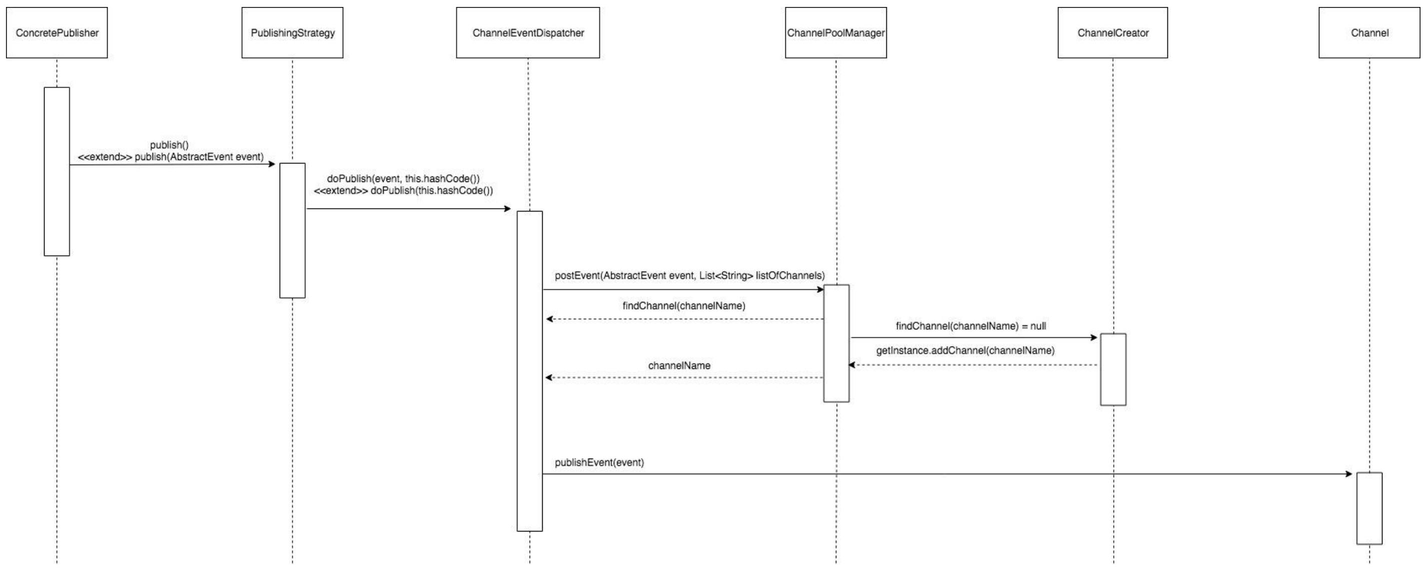
3.2.6 Event

Description	An entity that denotes a piece of information posted on a channel. An event has an event ID, a reference to its publisher, and a payload which is an *EventMessage* object. The *EventMessage* has a header and a body.
Attributes	
Sub-classes	EventIDMaker, AbstractEvent, EventTypeA, TempRiseEvent, EventMessage
Responsibilities	This entity is responsible for creating an event that will be published to a channel and interacted with by the subscribers.
Business Rules	

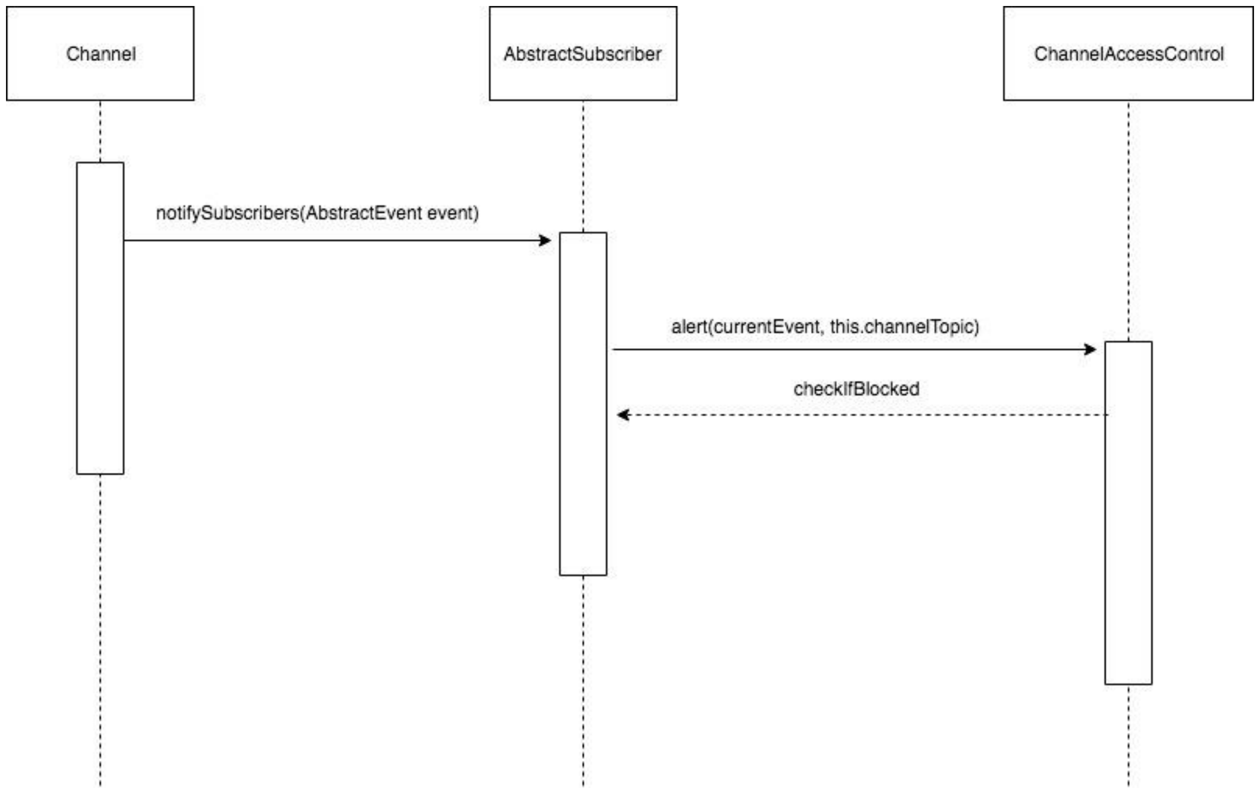
4 Interaction Diagrams

4.1 Sequencing Diagrams

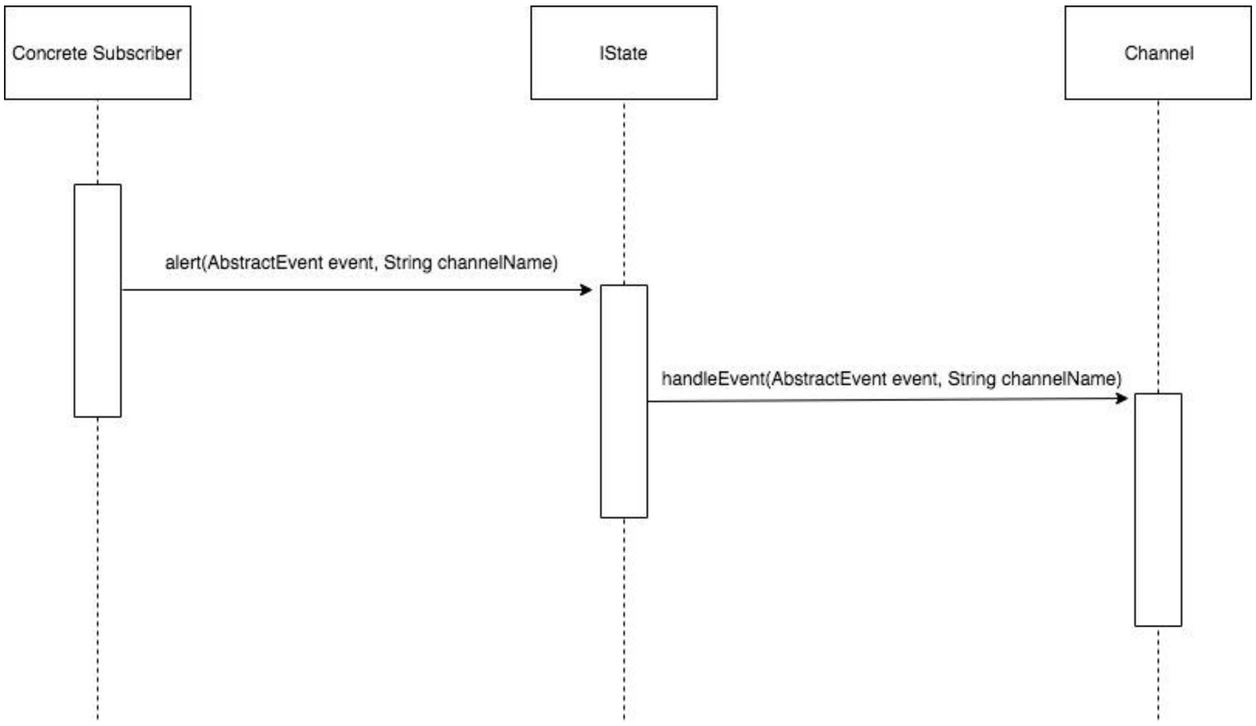
4.1.1 UC1


ConcretePublisher
PublishingStrategy
ChannelEventDispatcher
ChannelPoolManager
ChannelCreator
Channel
publish()
<<extend>> publish(AbstractEvent event)
doPublish(event, this.hashCode())
<<extend>> doPublish(this.hashCode())
postEvent(AbstractEvent event, List<String> listOfChannels)
findChannel(channelName)
findChannel(channelName) = null
getInstance.addChannel(channelName)
channelName
publishEvent(event)

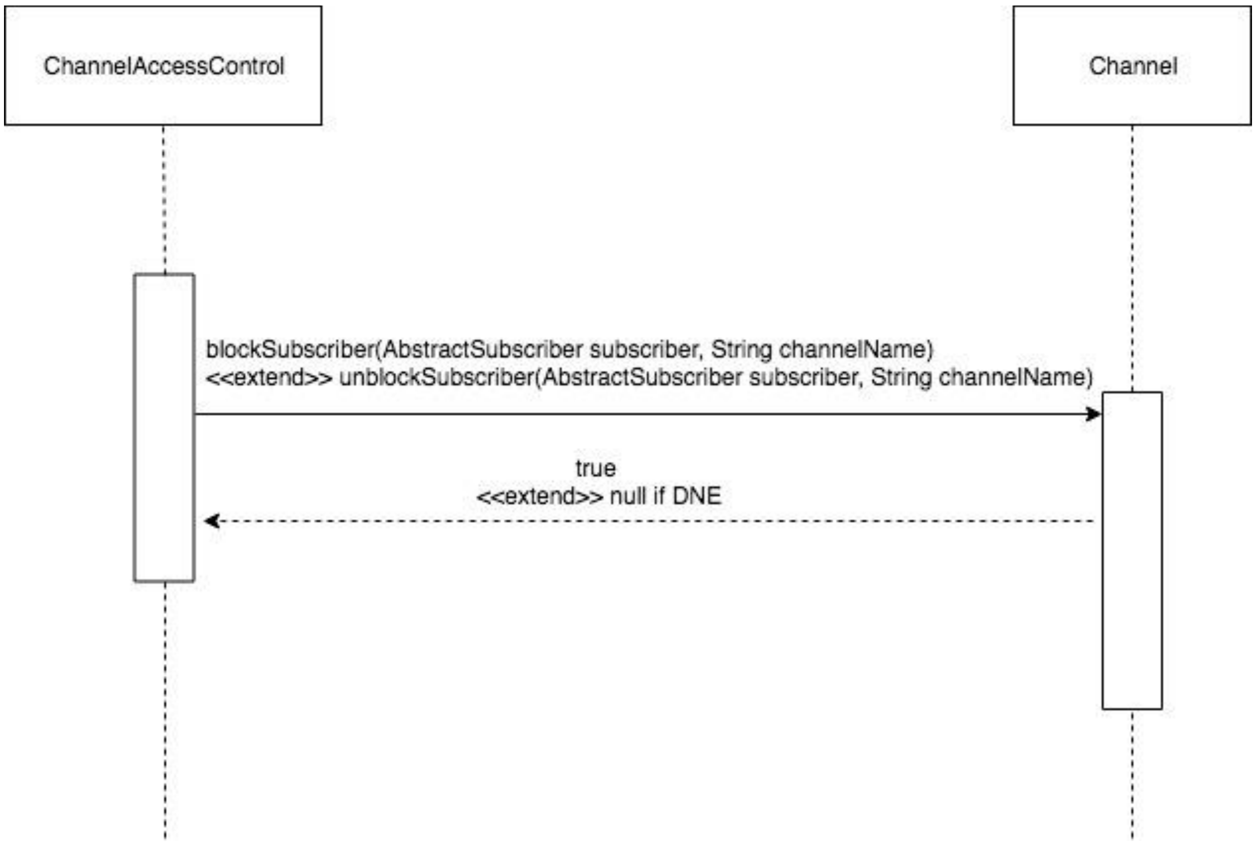
4.1.2 UC2


Channel
AbstractSubscriber
ChannelAccessControl
notifySubscribers(AbstractEvent event)
alert(currentEvent, this.channelTopic)
checkIfBlocked

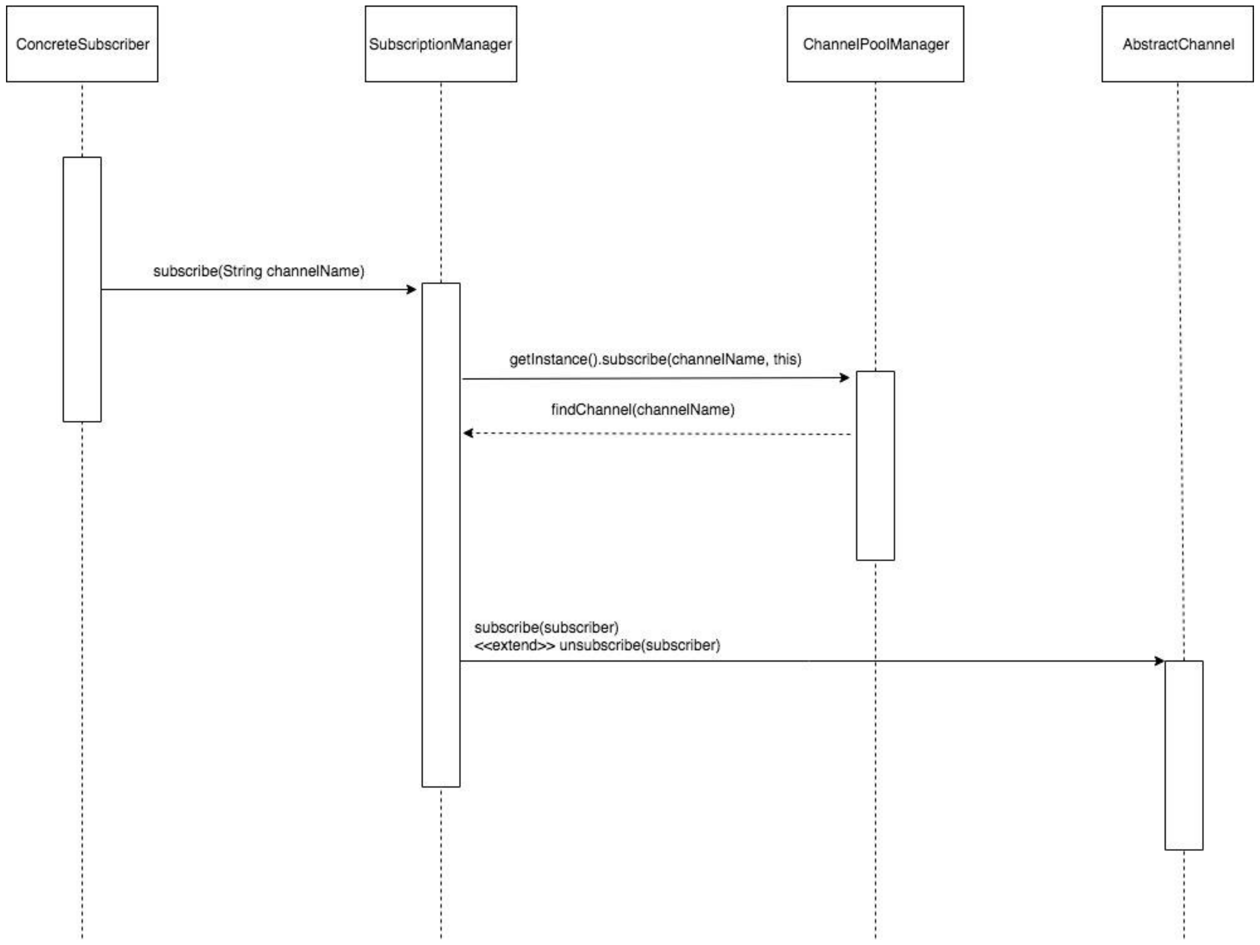
4.1.3 UC3


Concrete Subscriber
IState
Channel
alert(AbstractEvent event, String channelName)
handleEvent(AbstractEvent event, String channelName)

4.1.4 UC4

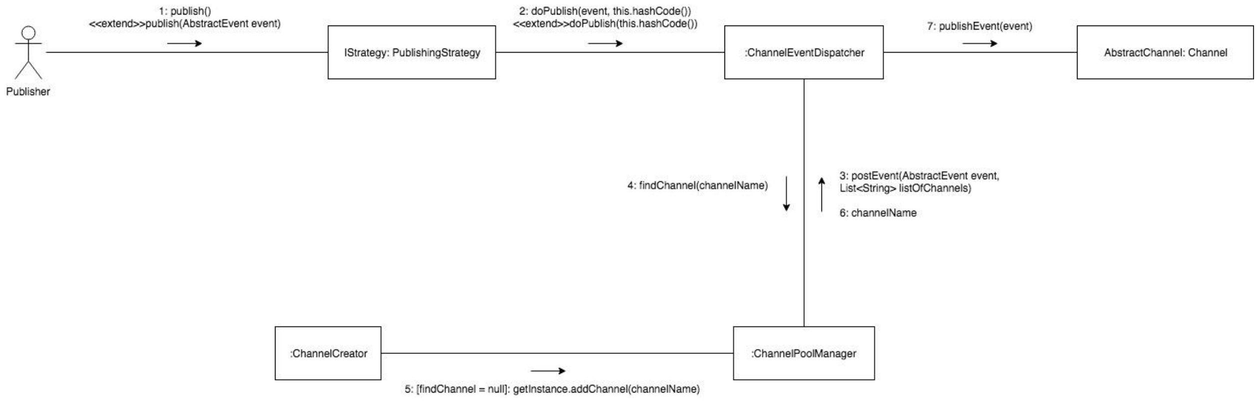

ChannelAccessControl
Channel
blockSubscriber(AbstractSubscriber subscriber, String channelName)
<<extend>> unblockSubscriber(AbstractSubscriber subscriber, String channelName)
true
<<extend>> null if DNE

4.1.5 UC5

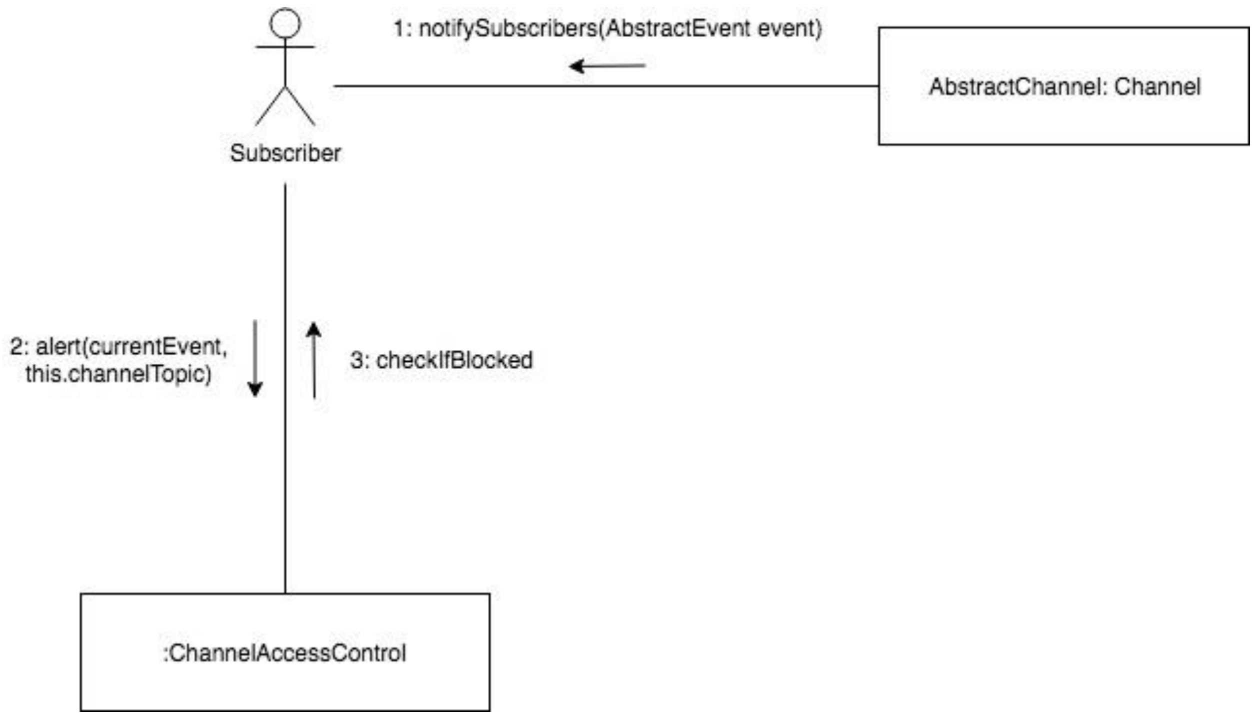


4.2 Collaboration Diagrams

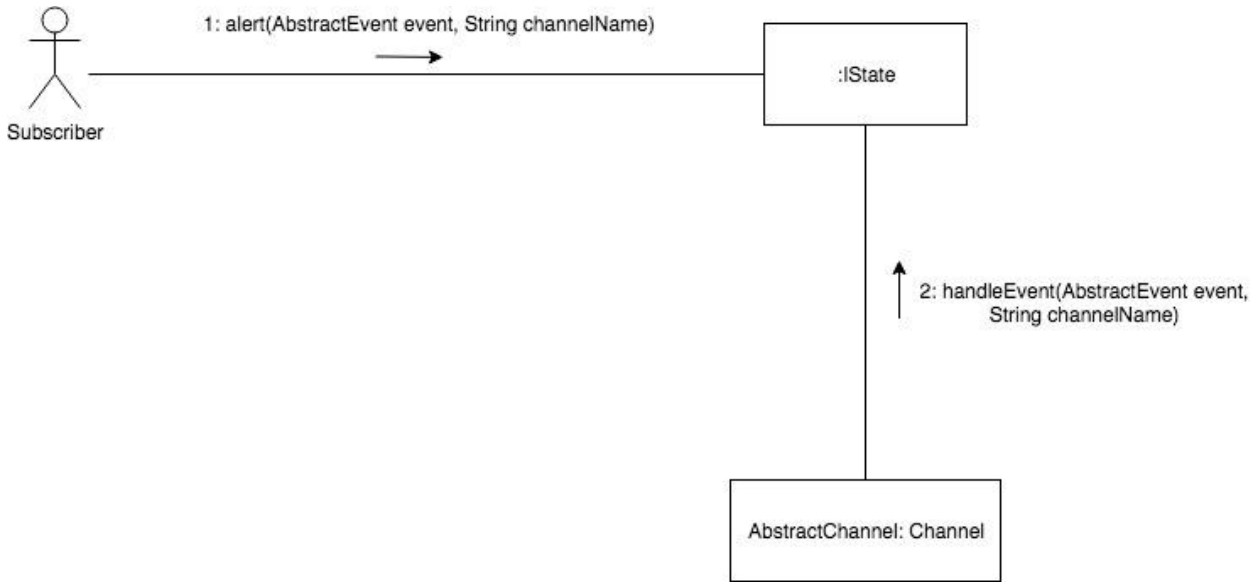
4.2.1 UC1


1: publish()
<<extend>>publish(AbstractEvent event)
Publisher
IStrategy: PublishingStrategy
2: doPublish(event, this.hashCode())
<<extend>>doPublish(this.hashCode())
:ChannelEventDispatcher
7: publishEvent(event)
AbstractChannel: Channel
4: findChannel(channelName)
3: postEvent(AbstractEvent event,
List<String> listOfChannels)
6: channelName
:ChannelCreator
:ChannelPoolManager
5: [findChannel = null]: getInstance.addChannel(channelName)

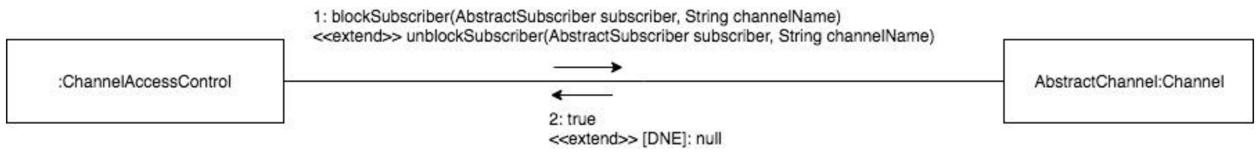
4.2.2 UC2


1: notifySubscribers(AbstractEvent event)
AbstractChannel: Channel
Subscriber
2: alert(currentEvent, this.channelTopic)
3: checkIfBlocked
:ChannelAccessControl

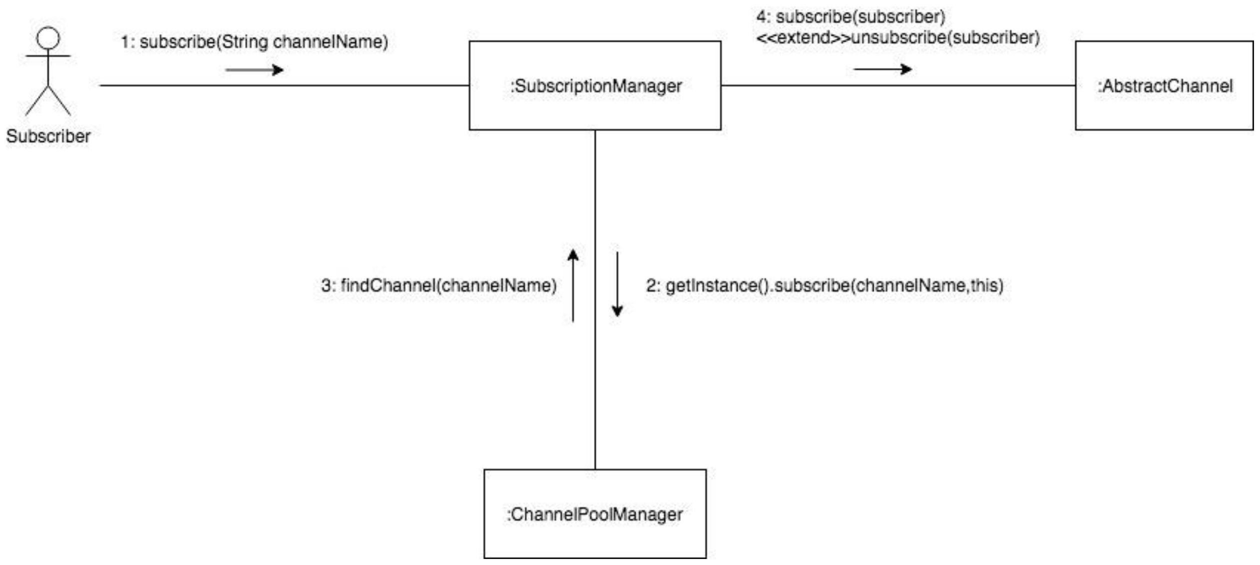
4.2.3 UC3


1: alert(AbstractEvent event, String channelName)
:IState
Subscriber
2: handleEvent(AbstractEvent event, String channelName)
AbstractChannel: Channel

4.2.4 UC4


1: blockSubscriber(AbstractSubscriber subscriber, String channelName)
<<extend>> unblockSubscriber(AbstractSubscriber subscriber, String channelName)
:ChannelAccessControl
AbstractChannel:Channel
2: true
<<extend>> [DNE]: null

4.2.5 UC5


1: subscribe(String channelName)
Subscriber
:SubscriptionManager
4: subscribe(subscriber)
<<extend>>unsubscribe(subscriber)
:AbstractChannel
3: findChannel(channelName)
2: getInstance().subscribe(channelName,this)
:ChannelPoolManager

5 Non-Functional Requirements Specification

5.1 Overview

The non-functional requirements of the system are comprised of utilities, environments, and other specifications that are necessary for the smooth operation of the system as a whole. This includes user interfaces, development environment, capacity specifications, network and operational parameters.

5.2 Enabling Technologies

5.2.1 Targeting Development Environment

The system should be developed in a Windows or Mac environment and using Java. The Integrated Development Environment (IDE) used is Eclipse.

5.3 Capacity Planning

5.3.1 Permanent Storage

In order to be reliable we will be using multiple disks to backup code and other relevant information. This ensures that if one of the disks fails the other disk functions as a single hard drive until the faulty one is replaced.

5.4 Network

The system should have connectivity to the Internet with bandwidth sufficient enough to carry signals.

5.5 Workstations

There are no minimum system requirements and configurations for the computer used for development. A hard disk space of 1GB and RAM of 2GB is recommended.

5.6 Operational Parameters

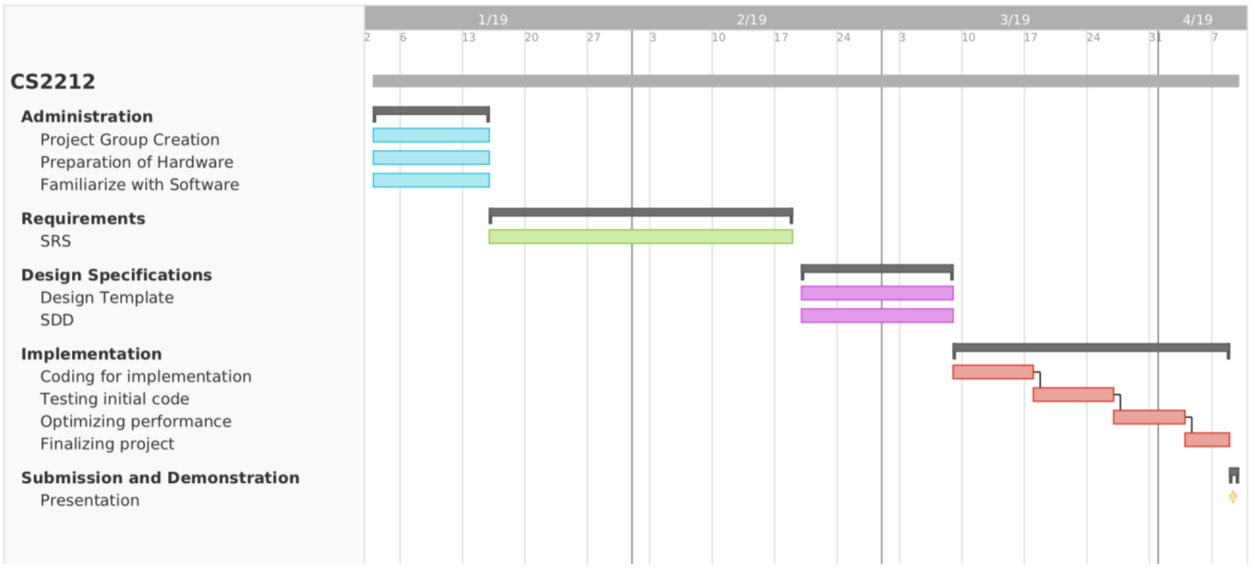
5.6.1 Usability

The system should be learnable and usable by the average computer user. The interface will be designed in such a way that is meaningful making it easy to access all the functions of the system.

5.6.2 Reliability

The backup and recovery functions of the system will consist of snapshots which will be taken at regular intervals and stored somewhere external to the system.

6 Activities Plan



7 Domain Dictionary

All terminology used in this document has been described either in this document or in a referred document.