FFTRADER _ FUNCTIONAL SPECIFICATIONS

Date: 29/07/2011

Version: 1.0

Purpose: The software FFTrader is a client-server platform that must allow traders to consult special trading data , to modify these data and to place orders on markets.

Table Of Contents

1.Client Side

1.1 Dojo App

The Client is being written in javascript using Dojo framework.

The client use the Dojo implementation of Comet to push data to the server.

1.2 DataGrid

The grid is made with the following fields: *Ticker,Type,Maturity,Exchange,Bid,Ask,Last,Open*

Definition of the fields :

Ticker:
Type:
Maturity
Exchange:
Bid:
Ask:
Last:
Open:

1.3 GUI

1.3.1 Push of the prices to the grid

When 3 mandatory fields are non-void , the grid must send data to the server .The server will check if the data refers to an existing InstrumentID and if this is the case , it will push the prices back to the client and then to the fields of the grid :

1/2) The user fills 3 mandatory fields

Ticker	Type	Maturity	Exchange	Bid	Ask	Last	Open
AAA	BBB		QQQ				

2/2) The prices are being pushed on the right side without intervention of the user

Ticker	Type	Maturity	Exchange	Bid	Ask	Last	Open
AAA	*BBB*		*QQQ*	*1.2*	*1.2*	*1.3*	*1.4*

Q: what happens if the user wants to edit the Maturity fields after he edits the 3 mandatory fields ? What happens if he wants to make some corrections to a mandatory field ? The grid should start requesting to the server the updated information at each change of the state of the cells ? Not only when 3 mandatory fields become non-void .Then we have something like:

(JavaScript)onUpdateField()->...->(Java)uid getPriceRequest(...)->sendPrices(uid)
After an instrument is recognized as valid, i.e when the client receives the id > 0, the Javascript changes the property to false (not implemented yet, I'll do). If the client wants to change a field, he has to remove the line (which results in canceling the subscription of the user to that instrument) and add another one or edit a blank line.

The only thing that you are requested to implement is the Javascript function PutPriceRequest(ticker, type, maturity, exchange) which will return an integer: > 0 in case of success, 0 in case of failure of the Servlet function with the same name

1.3.2 Submitting an order

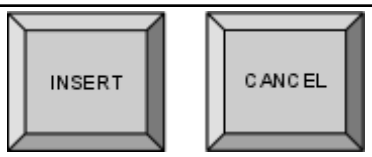
The datagrid must be provided with a "Submit Order" button No. The Bid and Ask fields should do that instead of the button (I'll do that):

Ticker	Type	Maturity	Exchange	Bid	Ask	Last	Open
AAA	BBB		QQQ	*1.2*	*1.2*	*1.3*	*1.4*

When the client click the "Submit order" button , a new window will appear:

Ticker	Type	Maturity	Exchange	Quantity	Price	Side	Order Type	Account
AAA	BBB		QQQ					

Message(Free Text)



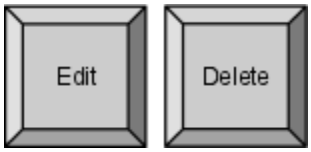
The 4 first fields : Ticker,Type,maturity and Exchange are not editable .The other fields are editable.

When the client clicks the “Insert” button , the data is sent to the server including the username , The server returns the Instrumentid.
The “cancel” button cancels the insertion of the Order.

1.3.3 Summary page

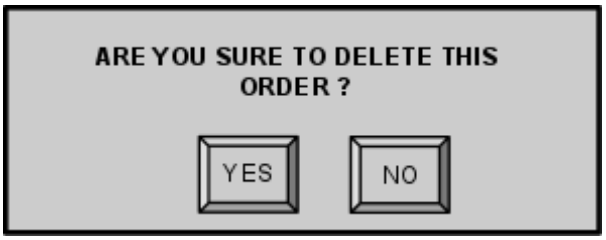
The summary page displays in a vertical format with a vertical scrollbar all the orders pending and executed. These data being constantly updated by the server:

Ticker	Type	Maturity	Exchange	Quantity	Price	Side	Order Type	Account	Username	Inserted time	modified time	status
AAA	BBB		QQQ	122	1.2	1	1	1	Trader1	12::12::12	12::12::13	accepted
AAA	CCC		RRR	132	0.9	2	2	2	Trader2	01::12::12	01::12::13	pending



When the user clicks on the “Edit” button , this launch a window which is identical to the “Submit Order” Windows from 1.3.2 except that the title is now “Modify Order” instead of “Insert Order”. The ability to use this button is conditioned by the value of the ‘Status’ field and by the role of the user (based on its username) . If the username is not a ‘Master’ role then it is not possible to use the edit button. (see 3.2/Conditional Access) **No. The non master role has the rights to edit its own orders**
OK

When the use clicks on the “delete” button , this opens a windows asking for a confirmation “Yes/No”:



2.Server Side

2.1 Application server

The application server is a Glassfish server.The choice of a java application server (vs fast_cgi’s) has been made for sake of performance , scalability and reliability.

The App server must implement a PUSH mechanism with the client using the Grizzly implementation of COMET Push Technology (see 2.3).

The server must perform polling of data the clients have subscribed (see 2.8) .
Q:I have understood this like that :

1) The server will be constantly polling the updated data for the InstrumentId that have been “registered” Yes

2) When a client needs updated information about an instrumentID it will get data from the data already polled by the server (e.g.g will not directly initiate a connection to IB servers) OK The connection to IB has to be unique and always up

The servlets must implement the following methods:

int **getPriceRequest**(string Ticker,string Type,string Maturity,string Exchange)

return/args	Ticker	Type	Maturity	Exchange
Instrumentid				

return the updated value of an Instrumentid

Q: can you confirm and explain what these functions and the others will do ? I'll implement it. The function will create an instance of an IB Instrument and subscribe for receiving market data. If the IB API will answer with an exception (=instrument not found or prices not subscribed), the function will return 0 and the client will do nothing. If the subscription is OK, the function will look into the db table instrument and will add the instrument if it's missing, and will add the subscription of the user to that instrument. In this second case, the function will return the instrument id (auto-generated by the mysql table). Something like this. What I need is that this function exists and is triggered when the user updates the datagrid, and the function contains these input values.

double **sendPrices**(int Instrumentid,int Pricetype)

return/args	Instrumentid	Pricetype
Price?		

return the realtime prices of an Instrumentid. **I'll implement it. This function is triggered when the IB API sends a price for a subscribed instrument**

double **getOrder**(int Instrumentid,string Username,int Quantity,double Price,string Side)

return/args	Instrumentid	Username	Quantity	Price	Side
?					

place an order **I'll implement it. The function will place an order to the IB API**

2.2 IB API

The IB (Interactive Broker) API must be instantiated uniquely for the whole application using an EWrapper interface.

The IB API class must be instantiated during the startup of Glassfish.

The IB API must authenticate with a running Gateway IB in order to communicate with IB servers
Q:can you confirm ?
The IB API gets access to the IB servers and gets granted access through an already authenticated IB gateway.The usernames are only internal to FFTrader and will not be used by the IB gateway or by the IB servers? Exactly. I have sent you the authentication details (for a paper = test account) for the IB Gateway

The Gateway must be re-authenticated every day by mean of a cron job or whatever similar task
.Q:Apparently the authentication of the gateway must be done manually with a prompt for the login and password. Should it be started manually every day or should we implement a cron task to do it automatically (a robot that would enter automatically the data in the fields) ? Not possible. The login to IB gateway prints 2 codes (in a not machine-readable format) and requires looking for the respective code-answers from a card. I'll need to reboot the IB gateway daily, and we could think at a button or similar that reconnects the servlet after this.
OK they have an OTP or something like that ,I was almost sure that IB was not giving access with a static login/password.

2.3 GRIZZLY COMET API

2.3.1 Comet

The comet support must be enabled in the http listeners bound to the virtual servers where the JFFTrader component are being deployed

2.3.2 CometD and the Bayeux Protocol implementation

The Bayeux protocol and its implementation made by the CometD project will be used for creating Bidirectional communications channel (see 2.8)

2.4 JERSEY RESTFul service

Q: 1) What is RESTFul supposed to do ? It only works for persistence of data (e.g. when the session has expired) or is it being used all the time to handle connections to the db?
2) How will the work be divided between RESTFul and Glassfish?
3) Does the Glassfish server have R/W access to the db ?

Concerning Jersey, I have read that this is the easiest way to connect a database to a Dojo DataGrid, I have used NB wizard for creating the RESTful web service, and this works well.

2.5 SQL database

The SQL database contains a table named FFTrader with the following fields :

Instrument id (key)	Ticker	Type	Maturity	Exchange	?	?	?	?

The prices and the data that are constantly changing in general (e.g. volatile data) are not to be stored in the db. The db is mainly used for data persistence.

2.6 GUI

2.6.1 Administration / Control Panel

The server must provide routines and interfaces for managing the platform , the orders and the users. This should be achieved through a web interface .

//TODO : describes WEB ADMIN interface

2.7 Management

The server must be running on a Debian 5 O.S - 'Lenny'
The rc.d script (/etc/init/rc.d) must be configured to enable the startup of :

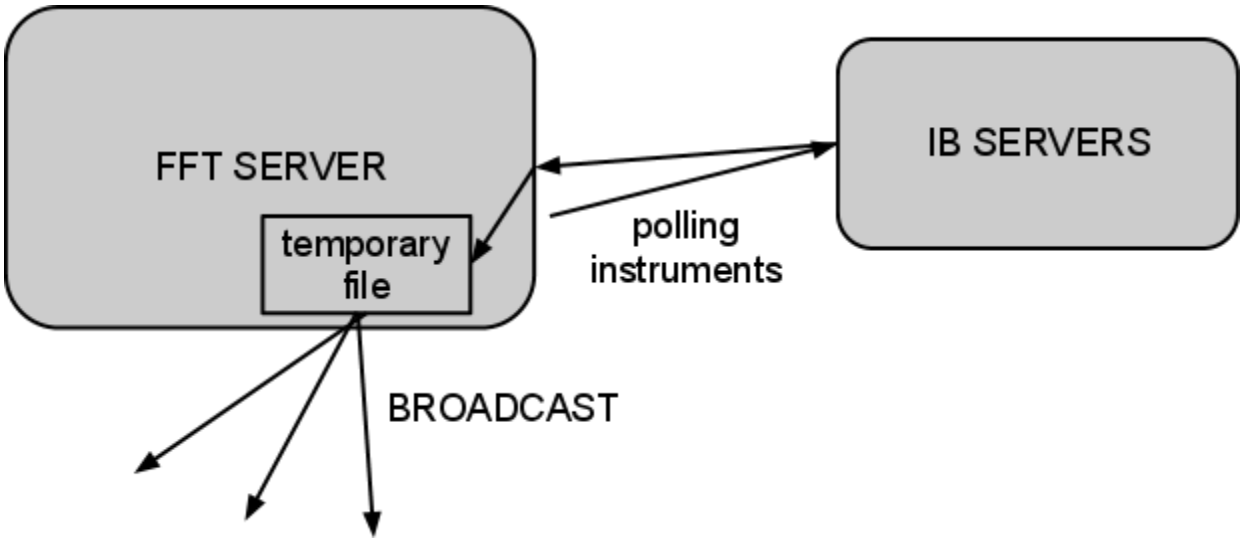
- OpenDS/LDAP
- Glassfish
- SQL
- The IB Gateway

2.8 Bidirectional communications and flux of information

The server side will implement the following model for pushing data :

- **by default the server do not poll any information to the IB servers**
- **when a client starts requesting a subscription to an instrumentID the server applies the following algorithm:**
  - **1) if the instrumentID is not broadcasted (e.g. no client is being subscribed to it) then the server include this instrumentID to be polled from ib servers and to be broadcasted , the client is being authenticated then if this authentication is successful , he is being given access to broadcasting of this instrumentID**
  - **2) if the instrumentid is already being broadcasted then the client is authenticated and if the authentication is successful , he is given access to the broadcasting of the instrumentID**
- **the broadcasting process and the polling process are asynchronous , e.g are not sequential to each others**

This is illustrated by the following figure:



3.Security

3.1 Authentication of the clients

The authentication of the clients must be done by mean of an OpenDS server that will be used as an interface to an LDAP directory .

The software should allow the filtering by I.P address of the client

The openDS server (which is standalone) may be integrated into the Glassfish server by the “standard” procedure with the creation of a web.xml document .(see https://www.opends.org/wiki/page/GlassfishApplicationServer.) but the server must be able to retrieve the session info.

The SSO support must be enabled in the http listeners bound to the virtual servers where the JFFTrader component are being deployed

3.2 Confidentiality

The data transferred between the clients must be transmitted via a SSL channel to be established between the browser and the server.

The SSL support must be enabled in the http listeners bound to the virtual servers where the JFFTrader component are being deployed

3.2 Conditional access

The conditional access of the clients will be as follows :

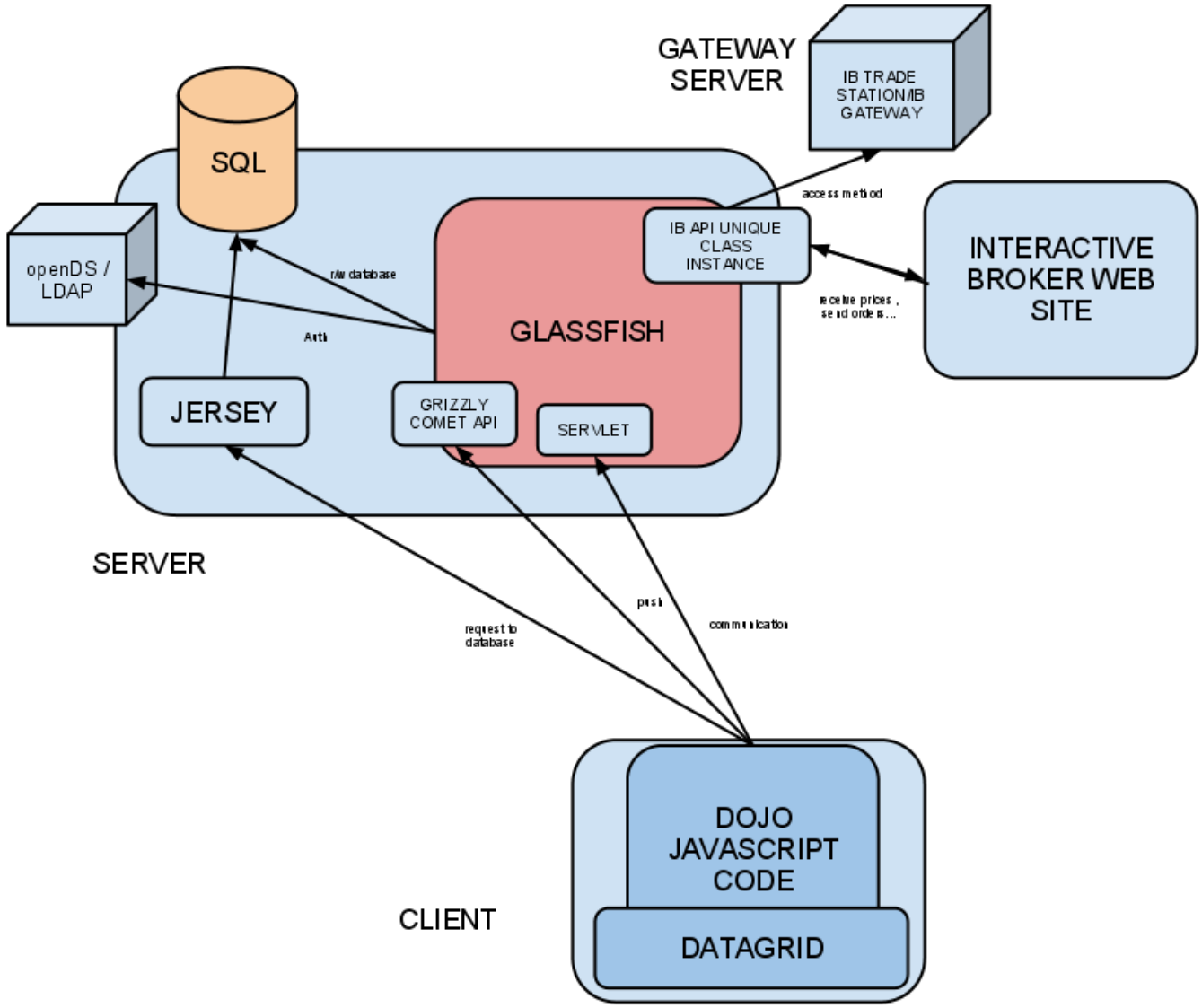
Type/Function	Read Informations	Write Informations	Edit Orders	Read/Write Other Traders Orders
Master	Yes	Yes	Yes	Yes
Trader	Yes	Yes	Yes	No

4.Architecture

4.1 Ecosystem

The Architecture of FFTrader is being described by the following picture , the main components are :

- *An OpenDS server that implements a LDAP server*
- *An application server , Glassfish ,used to process the orders from the clients and to perform bidirectionals pushes.*
- *A Sql database to store the Persistent Data*
- *A Jersey component used to get RESTFul services*
- *A Grizzly Comet component used for bidirectionals pushes client <->server*
- *The InteractiveBroker gateway*
- *The interactive Broker API Component*



4.2 General

The application JFFTrader must be started with glassfish . It will be working as such :

the main JFFTServer initialize at startup :

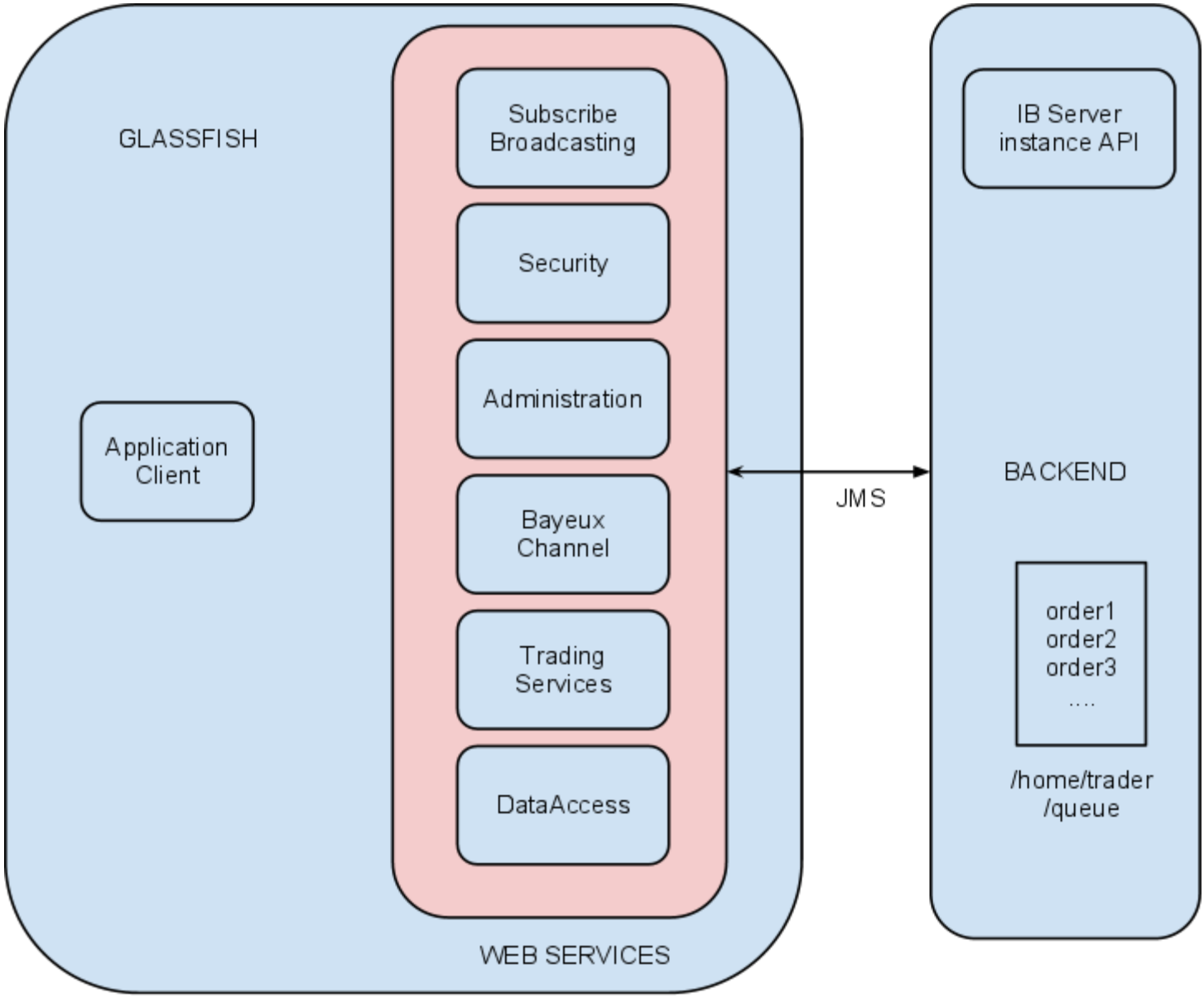
- Test the presence of the OpenDS and SQL servers
- Instantiate the IB API
- Initiate communication with IB servers
- Retrieve list of InstrumentID subscribed by clients

- Starts polling **Should the server starts polling even if there is no client connected ?In my mind : yes this is the philosophy of broadcasting**
- Starts Broadcasting service
- Starts Subscription service
- Starts CometD service

4.3 Backend

The components will be divided in 3 distinct parts :

1. A web Client Application hosted by Glassfish
2. Web services that will provide methods (EJBs , ...) to the client also ran into Glassfish
3. A backend ‘server’ that will process all the orders that will not be run into Glassfish and that will communicate with the other applications by messaging via JMS , the overall connection being made by JCA connectors

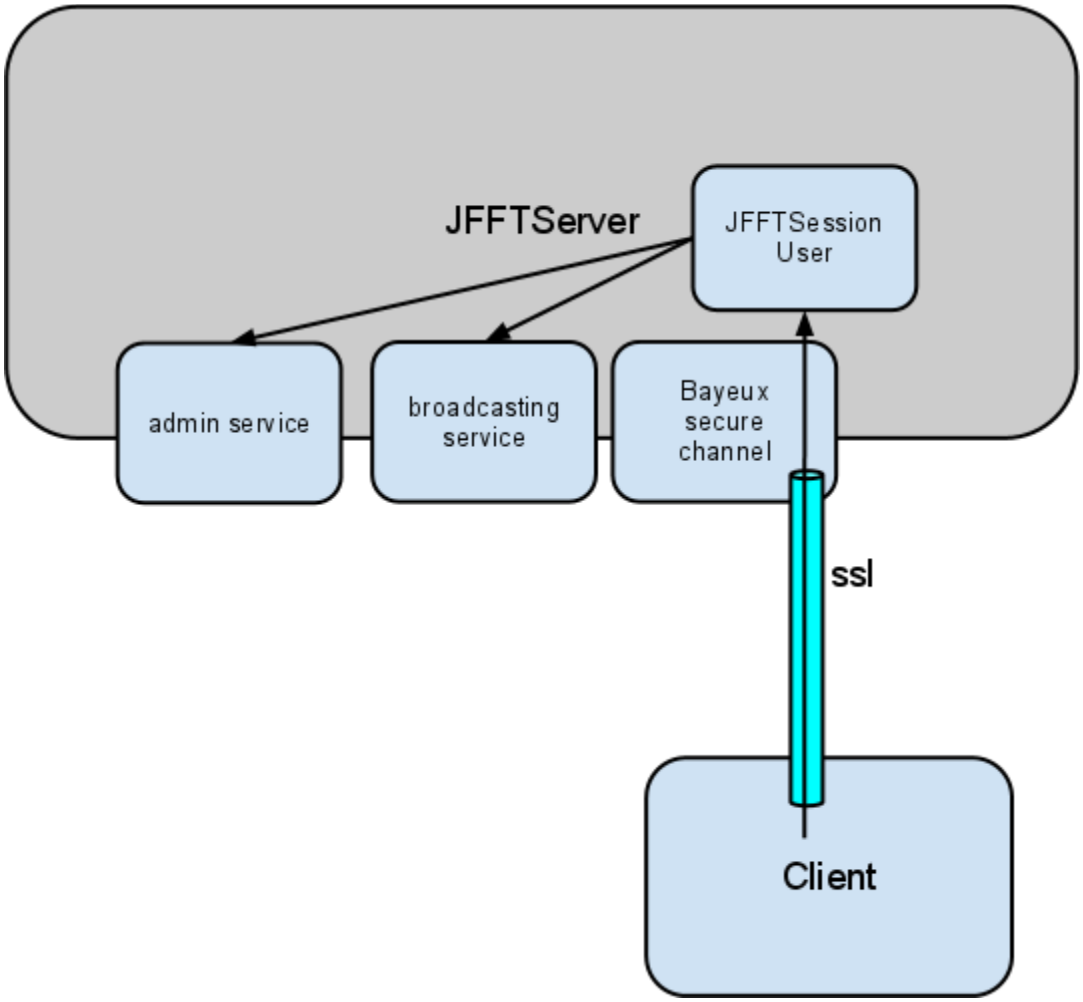


-
- Starts Bayeux bidirectional channel support (including SSL)
- Starts Admin service
- Other init tasks (to be completed)
- Wait for a client to connect

When a client connects :

- The client authenticate via openDS using the web.xml info
- The server first allocate to the client a specific Bayeux communication channel
- Then instantiate an instance of SessionUser with the authentication info included
- the client and the server can talk to each other (**the format has to be defined**)

The following figure illustrate this:



5.Design

5.1 Client

5.2 Server

5.2.1 Classes

The FFTraderServer application will be divided into the following components :
//TODO: Update this with the new definitions and new classes from the javadoc

class User
role: model an user of the platform FFTRader

class SessionUser
role: manage the session of the users , authenticate to OpenDS

class SSL
role: open , close , manage the SSL channels

class Tools
role: provide miscellaneous tools for getting information, debugging,provide Log , etc...

class DBRequest
role: provide the routines fro the db management

class FFTrader
role: provide the routines for managing the request to the IB servers and the processing of the trade orders

class Admin
role: provide the routines for managing the users , the orders and for the management of the platform

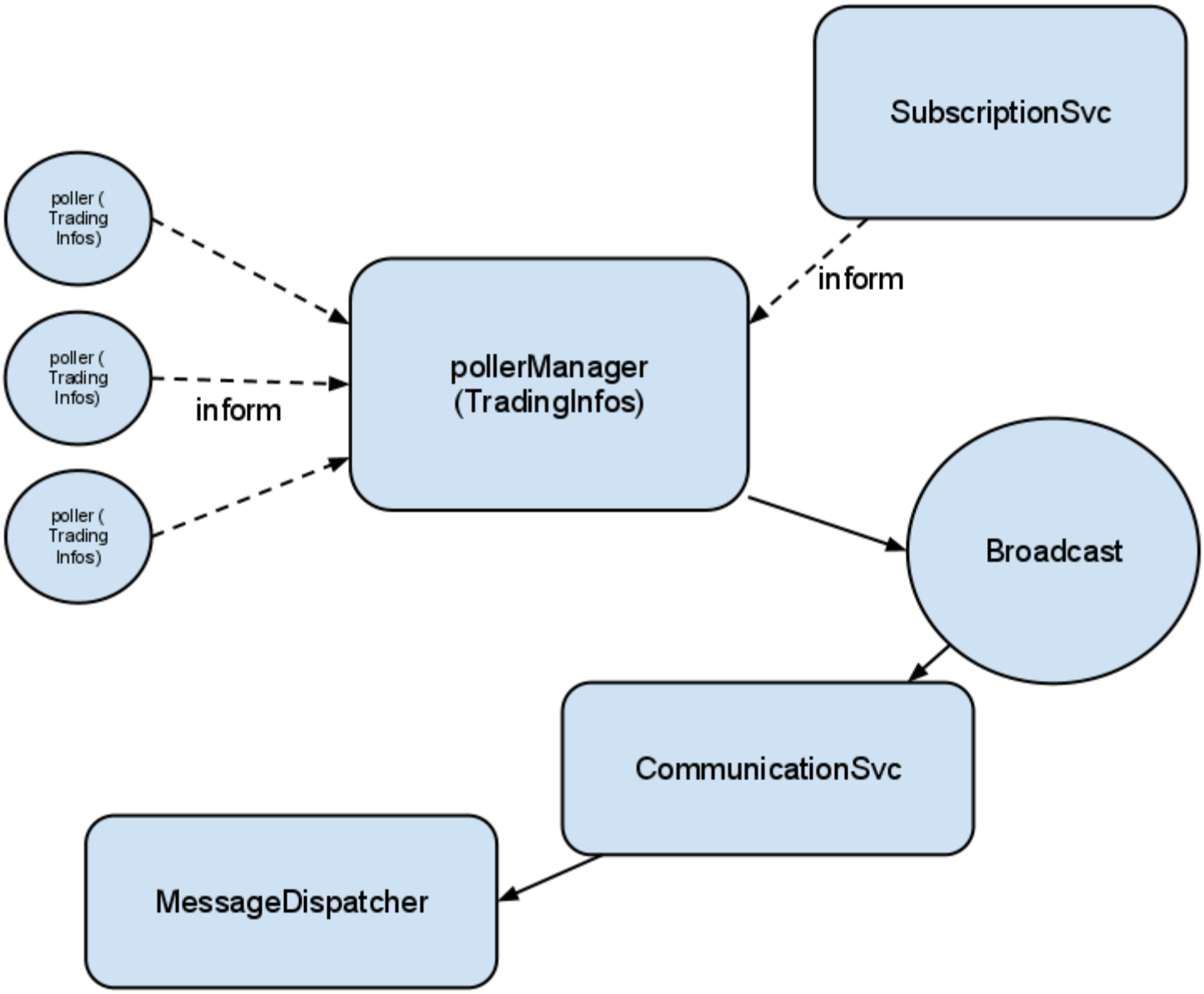
class ManagePush
role: manage the bidirectional pushes , based on Comet

class Instrument
role: model an instrument

class JFFTServer
role: provide the server functionality and the main[] entry point

5.2.2 polling of Trading Informations

The following figure represents the way Trading infos are being polled and dispatched. The polling manager listens to the pollers and is being informed when new trading infos are coming , he then gets them , sends them to the broadcaster who sends them to the message dispatcher .



5.2.2 processing of Trading Orders

The following figure represents how the Orders are being processed .

5.2.3 Message Dispatcher

The configuration service allows two choices :
1) the use of a message Dispatcher , directly after the broadcaster , that sends the message to the channels opened by those to whom the message has to be delivered
2) the sole use of a broadcaster , the client being responsible for filtering the messages he wants to read

5.2.4 Initializer Service

The initializer service starts with Glassfish , reads the configuration files then starts and configure the servers and the services .

5.2.5 Global view