Technical solutions for integrated trading on Spot, Futures and Bonds stock markets

Vlad Diaconita, Ion Lungu, Adela Bara
Department of Economic Informatics
The Academy of Economic Studies
Piata Romana Street, nr. 6, BUCHAREST
Research Financed by UEFISCSU, GRANT code 820/2007
diaconita.vlad@csie.ase.ro; ion.lungu@ie.ase.ro; bara.adela@ie.ase.ro

Abstract: - This article is an extended version of a paper presented in the WSEAS MCBE09 conference [10] in which we will present in more detail practical solution for building an integration tier between an online trading platform and two stock exchange markets, in a SOA like architecture. Our solution is constructed using XML, Java and PL/SQL and not third-party costly solutions. This open source approach, even if it is more difficult to develop and implement at first, helps a company to control the solution. The system will not be tied to SOA vendors that are usually keeping their software as secret as possible, demanding that they will develop, an additional costs, all the future developments. We are using XML for communication, not only because of the constraints given by the systems we integrate, but also because it is the natural choice in a SOA like solution. XML it's being used to enable web services and similar, often custom, RPC functionality to allow greater access to data across multiple systems within an organization and allowing the possibility of future systems to be created from collections of such RPC functionality [5, 6].

Keywords: - XML, integration, SOA, PL/SQL, Java, Threads, Spot Market, Futures Market

1. Stock markets

A stock exchange is a corporation or mutual organization which provides "trading" facilities for stock brokers and traders, to trade stocks and other securities. Stock exchanges also provide facilities for the issue and redemption of securities as well as other financial instruments and capital events including the payment of income and dividends. The securities traded on a stock exchange include: shares issued by companies, unit trusts, derivatives, pooled investment products and bonds. To be able to trade a security on a certain stock exchange, it has to be listed there. Usually there is a central location at least for recordkeeping, but trade is less and less linked to such a physical place, as modern markets are electronic networks, which gives them advantages of speed and cost of transactions. Trade on an exchange is by members only. The initial offering of stocks and bonds to investors is by definition done in the primary market and subsequent trading is done in the secondary market. A stock exchange is often the most important component of a stock market. Supply and demand in stock markets are driven by various factors which, as in all free markets, affect the price of stocks (see stock valuation).

There is usually no compulsion to issue stock via the stock exchange itself, nor must stock be subsequently traded on the exchange. Such trading is said to be off exchange or over-the-counter. This is the usual way that derivatives and bonds are traded. Increasingly, stock exchanges are part of a global market for securities.

In this article we will talk about integrating trading on three different stock exchanges:

- Spot market
- Bonds market
- Future market

Historically, the markets, which as noted, encompasses the totality of stock trading on all exchanges has been slow to respond to technological innovation, thus allowing growing pure speculation to continue.

Conversion to all-electronic trading could erode/eliminate the trading profits of floor specialists.

In the case presented in this paper, the spot and bonds market runs a different platform from the futures market belonging to different organizations. Beside this, trading on different markets posses economic constraints and difference to which we will advance solutions.

2. Problem formulation

The intermediary, the broker, the initiator of the integration, has a RDBMS based operational system that must be integrated with two systems, the systems of two different stock exchange markets. The integration has to be build so a user that connects to the broker's system must be able to make an order on any of the two markets. The system must receive, validate and send the order to the corresponding market, and receive the confirmation or the rejection of that order, all in real time.

The components to be integrated are the following:

- The online trading platform informatics system (OTPIS) built on a three tier web architecture which enables users to connect, using a user and a password, to see the market picture and to trade, for the money they made available, on one or several stock exchange markets.
- The integration tier of the on-line platform side which consists of several applications enabling the OTPIS to communicate with the stock exchanges.
- The integration tier of the stock exchange side. Every stock exchange distributes a client application, a gateway that must be used by the integration team of the broker to connect their system to the stock exchange system. In this case study we will to the gateways: GW1 and GW2.
- The Stock Exchange systems, on which the offer and demand meet and the trades are taking place. We will refer to two systems: SE1 and SE2.

3. Problem Solution

The OPTIS is build around an Oracle solution, a 10G Database Server and a 10G Application Server. To integrate the systems, we will build the integration tier of the OPTIS by using the existing procedures and by writing new procedures to enable them to exchange information. The architecture of the solution is shown in Fig. 1.

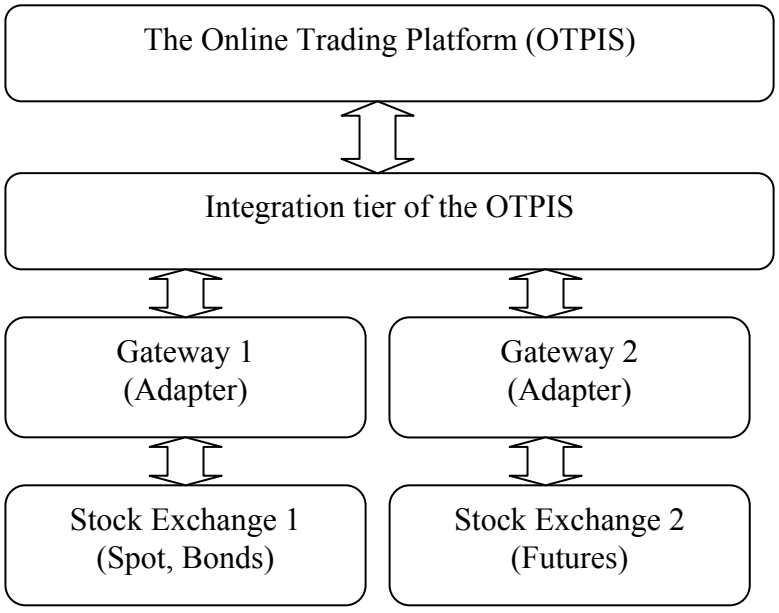


Fig. 1: SOA style Integration Architecture

We will build the integration tier using PL/SQL and Java procedures to make the applications communicate and to manipulate relational and the XML data.

Oracle has implemented XML through the XML DB component of the database, which is a standard feature in Enterprise Edition as well as Standard Edition. The SQL/XML standard has been under development to provide a mechanism that allows the users to generate XML from a relational query and, conversely, provide the ability to deliver SQL data from XML documents. Oracle XML DB in Oracle Database 10*g* Release 2 implements the SQL2003 standard and features from the upcoming SQL/XML standard release. A new data type, XMLType, was introduced starting the 9i release of Oracle Database that allowed an XML document in the database to be accessible in SQL and at the same time allow XML developers the ability to use XML standards on a document. This data type tells the database that the content is in XML format and allows us to perform queries on an XML document. Using XMLType rather than a relational or CLOB implementation provides a layer of separation between the application and storage model. This separation can allow data to move to a different storage model without being tied to a CLOB or relational model. XMLType can be used to create a table, column, or view. It can also be used as a data type for parameters and variables. Built-in XML methods operate on the content of a document by allowing the user to create, extract, and index XML data. Indexing can be performed using b-tree, text indexing, and function-based indexes. In effect, XMLType data combined with XPath access can be used to look inside a document. This functionality is provided through PL/SQL and Java APIs. XMLType can be used in PL/SQL, in Java using JDBC, and in the Oracle Data Provider for .Net. When passing queries we use optimization techniques like: table partitioning, indexing, using hints and using analytical functions (described in detail in [9]) instead of data aggregation in some reports in order to reduce as much possible the response time and to prevent deadlocks.

3.1 Gateway Client 1: Stocks and bonds/bills market

Between the GW1 and the integration tier the messages have the following format [8]:
|start sequence | message body |checksum| end sequence

1. The start sequence is a 9 byte sequence (0xEF 0x81 0x86 0xE2 0x86 0xA6 0xEF 0x81 0x86)
2. Message Body - the message payload; it has variable length
3. Checksum – the MD5 hash computed over the message body - 16 bytes
4. The end sequence is a 9 byte sequence (0xEF 0x81 0x85 0xE2 0x86 0xA4 0xEF 0x81 0x85)

UTF-8 character set is used to convert between bytes and character sequence representation of the message body. The message body should not contain the start or end sequences in order to prevent a misinterpretation of the real message. There are two types of messages that flow between the gateway and the client:

- outgoing messages : messages that are sent from the gateway client to the central system through the gateway
- incoming messages: messages received from the central system by the gateway client through the gateway

At application level the message body is interpreted as an XML formatted text. The XML message structure is fully described using an XML Schema file named gateway-messages.xsd. Every outgoing message will be parsed and validated against the provided XML Schema. In the event of an invalidated message the gateway will send an error message to the client formatted against this specification.

Any outgoing message has a client sequence and an inner content that represents the actual command with its parameters. An outgoing engine message has the following general structure:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<m:outgoingEngineMessage xmlns:c="http://..." xmlns:m="
http://..." xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <content xsi:type=…>
   …
   </content>
   <csq>10001</csq>
</m:outgoingEngineMessage>
```

Content is the body of the actual command that is sent to the central system. The client sequence (csq) can be used to identify an incoming message as being the response for this outgoing message. The central system will not

modify the content of this field. Client sequence is managed by the gateway client.

Every incoming message has a header and an inner structure that embeds a command confirmation, a report, a market data event or other information. We can call these embedded structures Data Transfer Objects. An incoming message has the following general structure:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<m:incomingEngineMessage xmlns:c=" http://..." xmlns:m="
http://..." xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <content xsi:type=…>
      …
   </content>
   <csq>10003</csq>
   <err>false</err>
   <id>213</id>
   <kmsg>to ADMIN: Hello</kmsg>
   <ktime>20070827174849441</ktime>
</m:incomingEngineMessage>
```

Based on the *id* of the message, the client knows what to expect in the content. *Time,* having the *yyyyMMddHHmmssSSS,* format is the one at which the message was generated by the stock market's system. In the kmsg field, there is a comment, a message or a error description. The Client sequence (csq) is used to match an incoming message to a command; it is 0 in case of an unsolicited messages. In content we have the actual message content, one or more data structures packed together but it also can be NULL.

The table below lists all the possible incoming messages a client should expect to come unsolicited from the gateway [8]:

Message id	**Event**	**Message content type / comments**
800	Operational events and/or trading activity	TickersPack
802	Trade confirmation	HalfTrdDto The gateway will receive all trade notifications regardless they were performed by other users from the same participant
378, 373, 305, 309, 147, 381, 376, 719, 354, 555, 558	Operations performed on an order by another user (change, delete, etc). Note that order fills will not be transmitted this way.	OrdDto 719 arrives in case of order activation for contingent orders 354, 555 or 558 arrive in case batch operations are performed upon orders by system administrators The gateway will receive all order operations confirmations regardless they were performed by other users from the same participant
801	A text message sent by another user or other text announcements.	MailDto
102	The gateway was disconnected from the central system.	Null
603	The central system responds to a heartbeat message sent by the gateway. Note that this heartbeat is not related to the HeartBeatCmd.	Null. ktime will be a yyyyMMddHHmmssSSS timestamp representing the time of the central system.
762	Start init cache. This message arrives after a successful login or after the order book	SmkDto In this case only sym and mkt fields are relevant and they can come in 4 different combinations:

	cache was reloaded on the communication server.	• sym=*,mkt=* • sym=defined, mkt=* • sym=*, mkt=defined • sym=defined, mkt=defined After this message the client should expect a series of 175 type messages (MboDto payload will have the symbol and market field from the range definded in the abolve 4 categories) that could be used to initialize the cache of the client. For example if sym=* and mkt=* the client should expect the 175 type messages for every symbol-market it is subscribed to.
765	End init cache. This message marks the end of the cache initialization process.	Null
761	The order book cache was invalidated from the central system and no longer consistent.	SmkDto. See the comments for 762 above. When this message arrives the client should invalidate it's own cache as it is no longer consistent. For example if sym=* and mkt=REGS the client should invalidate the order books for every symbol trades in REGS market.
175	After a successful login or after the order book cache was reloaded on the communication server.	MboDto
790	When a market or symbol-market parameter is added, changed or deleted	ParamsShortDto

Table 1: Types of possible unsolicited incoming messages

3.2. Gateway Client 2 : Futures market

Communicating through the second gateway client is similar to the first; the difference is at the messages format and complexity. Exchanging information with the gateway client 2 is simpler; we don't use any embedded complex data structures, only straight XML. The incoming messages has the following structure

```
<market t='type'>
  <info>
    <label>label1</label>
    <value>value1</value>
  </info>
  ...
    <label>labeln</label>
    <value>valuen</value>
  </info>

</market>
```

Where type is the kind of message being received, and the content is send in (label,value) pairs. For example, to receive the market picture, we use the 112 type messages that describe the market at connect time and 212 messages that contain the updates. For knowing if an outgoing message, an order, was accepted or not, we have to analyze the 400 and 401 messages.
As an example, a message of type 112 has the following XML structure:

```
<market t='112'>
<info><label>503</label><value>DERRC</value></info>
<info><label>504</label><value>MAR09</value></info>
<info><label>510</label><value>0</value></info>
<info><label>511</label><value>0</value></info>
<info><label>525</label><value>0.003</value></info>
<info><label>505</label><value>0.024</value></info>
<info><label>520</label><value>0</value></info>
 <info><label>530</label><value>0</value></info>
<info><label>531</label><value>0.0</value></info>
<info><label>500</label><value>Active</value></info>
```

<info><label>540</label><value>0</value></info>
<info><label>550</label><value>2</value></info>
<info><label>542</label><value>0</value></info>
</market>

In the case of GW2 there are fewer outgoing message types. The most use are: new order and cancel order.

For example, a order is sent to the market using an XML structure like this one:

```
<m t='AddFuturesOrder'>
        <ls>31875</ls>
        <ct>DESIF2</ct>
        <sd>MAR09</sd>
        <ac>99</ac>
        <cc>990001</cc>
        <vo>5</vo>
        <pr>0.714</pr>
        <vl>Day</vl>
        <bs>Buy</bs>
        <ot>Limit</ot>
        <ap>0</ap>
</m>
```

In this case, we have a buy order for 5 contracts at 0.714, on DESIF5 with the MAR09 settle date.

We receive the accept/reject confirmation by a XML message of type 400:

400 – OrderInfo

500 Accept or reject(values: OK = accepted, Wrong = rejected)

610 Local order number for new orders or the market order number for deleted orders

615 Comment, explanation for rejection or the server time in case of acceptation

If the previous order was accepted we will receive the following message:

<market t='400'>

<info><label>500</label><value>OK</value></info>

*<info><label>610</label><value>*31875*</value></info>*

<info><label>615</label><value>15:46:38</value></info>

</market>

If the order has been rejected we will receive the following message:

<market t='400'>

<info><label>500</label><value>Wrong</value></info>

<info><label>610</label><value>31875</value></info>

<info><label>615</label><value>You can not cancel order while status<>Active.:17:45:16</value></info>

3.3. Putting it together

In OPTIS, we'll have an integration point, a table VALIDATE_ORDERS {Id_robot, Ord, Oper, Enter_Date, Market_Date, Market_Order, Status, Stare, XML_String}, which contains all the orders that were received from clients at meet the conditions to be sent to the market. Next, we will build a java application that reads the orders from this table and sends them to the markets. The same java application receives the messages and sends them to PL/SQL procedures which converts them back to relational. The message flow is portrayed in figure 2:

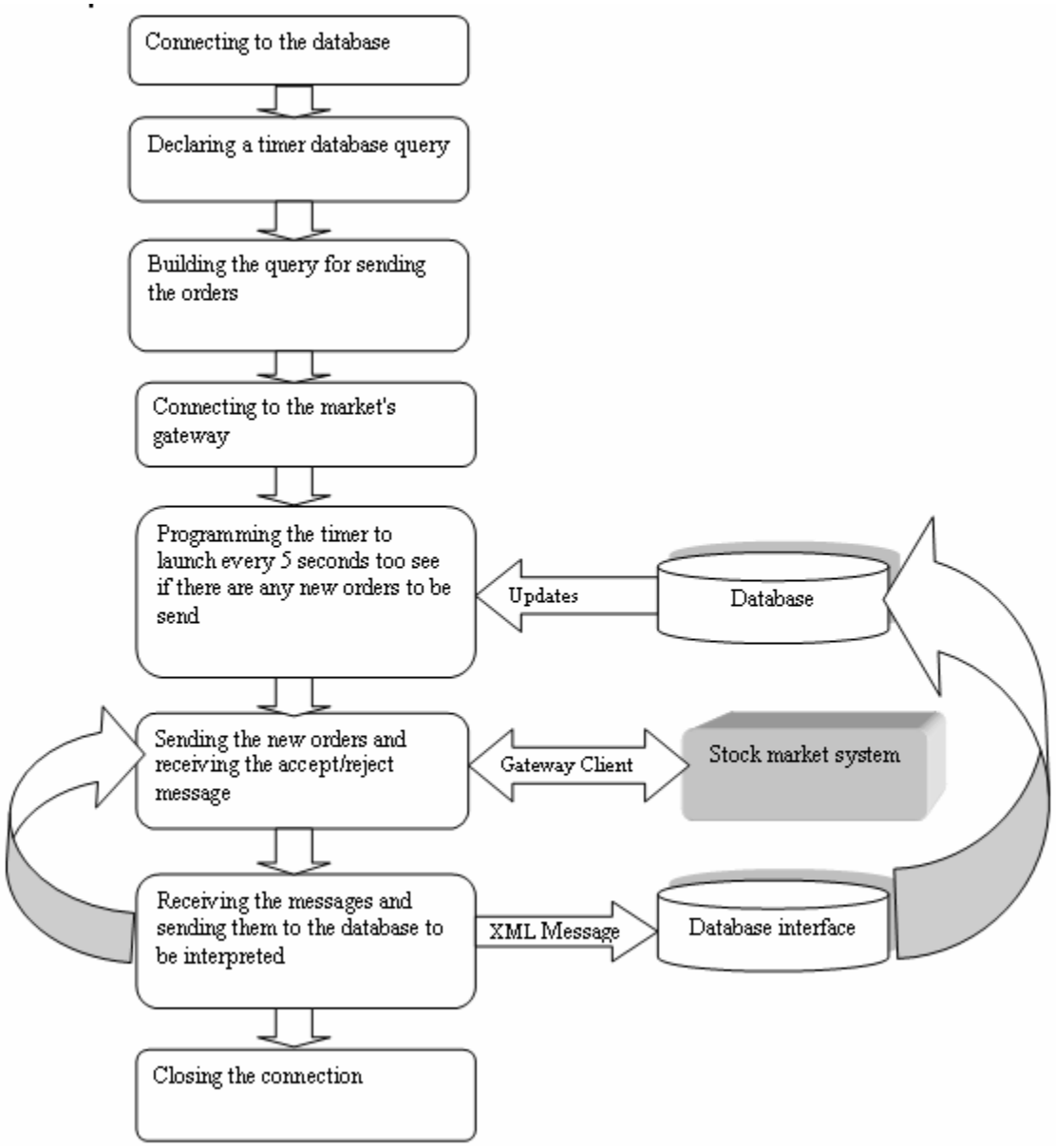


Fig 2. Message flow between the systems

Using the gateway client the system can issue commands as outgoing message any time after login. Any command that arrives at the central system will have a response that maps to an incoming message containing a confirmation that states the command was executed as expected or an incoming message with the error flag marked to true that states that the command could not be executed. In case of an error should contain further explanations about the encountered error.
Also by using the gateway client the system can issue report requests as outgoing messages. Any report request that reaches the central system will have a response as an incoming message containing a page of records or an incoming message with the error flag marked to true. In case of an error message the content of the incoming message should contain further explanations about the encountered error. The number of records in a response page is limited. In order to receive all the pages of a report the gateway client has to issue further report requests, until the last page of records is received.

For the OPTIS integration tier, to connect to GW2 we use a simple Socket UTF communication. We build a separate thread in the application that assures the

exchange between the database and the GW2. This thread is for the incoming messages which are read from the socket.

```
   private static class ConexiuneSibex extends Thread {
      public Connection conn;
      public Statement StmtPersoane;
      ConexiuneSibex(Connection conn) {
         this.conn = conn;
          }
      public void run() {
       try{
            kin = new BufferedReader(new
InputStreamReader(System.in));
            s = new Socket("127.0.0.1",9191);
            dos = new DataOutputStream(s.getOutputStream());
            dis = new DataInputStream(s.getInputStream());
            System.out.print(dis.readUTF());
            dos.writeUTF("user_gw2");dos.flush();
            dos.writeUTF("pass_gw2");dos.flush();
         while (1==1){
            try
            {
            String t=dis.readUTF();
            ConexiuneBd c = new ConexiuneBd(conn,t);
            c.start();
             }
            catch(Exception e){
                  e.printStackTrace();
            }

         }
         }
       catch(Exception e){ e.printStackTrace(); }
      }
   }
```

We use a different thread to pass the XML to the database so the communication with the gateway is not affected by the procession time of the database.

```
   private static class ConexiuneBd extends Thread {

      public Statement StmtPersoane;
      Connection conn;
      String t;
      ConexiuneBd(Connection conn,String t) {
                this.conn = conn; this.t = t;  }
      public void run() {
       try{
       PreparedStatement       callstmt=conn.prepareCall("{call
parce_xml(trim(?))}");
       try{
           callstmt.setString(1,t);
            callstmt.executeUpdate();
            }
            catch(Exception e){e.printStackTrace();}
            callstmt.close();
          }
        catch(Exception e){e.printStackTrace();}
       }
   }
```

In the database tier, we need a procedure that receives the XML, deserializes it and uses the information correspondingly. In this example, we use a cursor to deserialize the XML and obtain the (label, value) pairs used to update the market picture.

```
create or replace procedure parce_xml(pstring varchar2) is
cursor c is
select
extractvalue(value(p), '/info/label') label,
extractvalue(value(p), '/info/value') value
from
table(
xmlsequence(
extract(
xmltype(pstring),'//info'))) p;
 cursor cc2(qserialno number) is
    select  'D' from futures_deal_ag t where
      t.serialno = qserialno;
[…]
  if pstring like '%<market t="112">%' then
    v:=true;

    for r in c loop
     if r.label=503 then pcontract:=r.value;  elsif r.label=504
then psettledate:=r.value;
     elsif r.label=510 then
pbestbid:=to_number(r.value,'99999.9999');
[…]
     end if;
    end loop;
   begin
    insert into
futures_market(CONTRACT,SETTLEDATE,BESTBID,BES
TASK,SETTLE,OPEN,HIGH,LOW,CHANGE,TRADES,CO
NTRACTS,OPENINT,STATUS,CATEGORY,DATA_FOLD
ER)    values
(pcontract,psettledate,pbestbid,pbestask,psettle,popen,phigh,pl
ow,pchange,ptrades,pcontracts,popenint
    ,pstatus,'MARKETS' ,vdata_folder);
    exception
     when dup_val_on_index then null;
[...]
end;
```

When connecting to GW1 we use a different approach. Even if the sent and received messages are in XML

format, we use some java packages (ro.java.gw) to ease the conversion [3].

For a sell/buy order the message format is the following:

Field name	Description
sym	Symbol code
mkt	Market code
stt	Settlement term type: 1 = standard, 2 = non standard
clr	Standard settlement date; if stt = 1, clr is one of the standard settlement terms as defined at the symbol level; if stt = 2, clr should be 0
std	Non standard settlement date; if stt = 1, std should be 0; if stt = 2 std should be the actual settlement date (yyyyMMdd)
trm	Order term; 0 = Fill or Kill, 1 = Day, 2 = Open, 3 = Good Till Date
opd	Open date; should be 0 unless trm = 3 else it should be the date until this order should live (yyyyMMdd)
ref	Comment
acc	Account number
prc	Price; -1 = Market, 0 = Unpriced, a positive numeric for Limit orders
size	Volume
ver	Special volume restriction; 0 = NONE
dcv	Disclosed volume; 0 unless the order is hidden else it should be the disclosed volume of the hidden order
tpa	Trigger price type; 1 = None, 2 = Stop, 3 = If Touched
tgp	Trigger price; a positive number unless tpa = 1 else 0 (zero)
ssl	Short sell mark (used only for sell order); 1=Yes, 0=No

```
import import ro.arena.gw
[…]
iem =
client.sendMessage(getAddSellOrder(pACC,pSym,pMkt,pPR
C,pVol,pOrd,pGTD));
```

```
[…]
public static OutgoingEngineMessage getAddSellOrder(int
pAcc,String pSym, String pMkt, BigDecimal pPrc, int
pSiz,String pOrd, int pGTD) {
AddOrderSellCmd buy = new AddOrderSellCmd();
buy.sym = pSym; buy.mkt = pMkt;
buy.stt = OrdDto.ORDER_STT_STANDARD;buy.clr = 3;
      if (pGTD>0)
       {
        buy.trm=3;  buy.opd=pGTD;
        }
        else
         {
        buy.trm = OrdDto.ORDER_TRM_DAY;
          }
         buy.ref = pOrd;buy.acc = pAcc;buy.prc = pPrc;buy.siz
= pSiz;buy.ver = OrdDto.ORDER_VER_NONE;
buy.dcv = 0;buy.tpa = OrdDto.ORDER_TPA_ORDINARY;
buy.tgp = new BigDecimal("0");
 return new OutgoingEngineMessage(buy,
GatewayClient.getNextClientSequence());
           }
```

This function gets appealed from this context, first we send the order to the market and then we receive the confirmation. After we receive the confirmation we must update the state of the order so the client will know if his order was accepted or rejected.

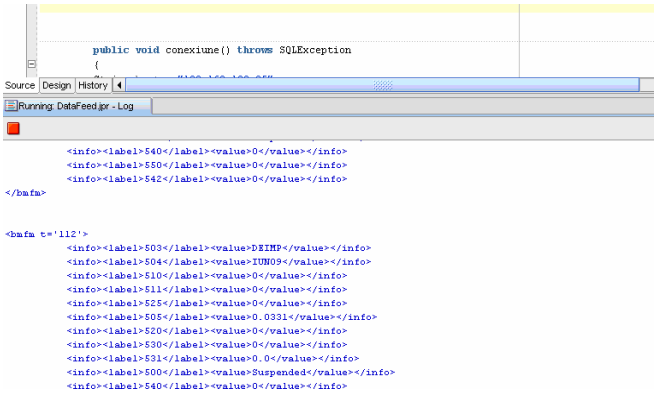
```
 while (this.CrsPersoane.next())
{
 String pSym=CrsPersoane.getString("sym");
 int pVol=CrsPersoane.getInt("vol");
 int pACC=CrsPersoane.getInt("acc");
 int pOper=CrsPersoane.getInt("oper");
 int pSDE=CrsPersoane.getInt("sde");
 String pOrd=CrsPersoane.getString("ord");
 int pOrd_Arena=CrsPersoane.getInt("ord_a");
 String pMkt=CrsPersoane.getString("mkt");
 int pGTD=CrsPersoane.getInt("gtd");
 BigDecimal pPRC=CrsPersoane.getBigDecimal("prc");
 java.sql.Timestamp  Ts = CrsPersoane.getTimestamp("ts");

if (pOper==1) // New Order
     {
        if (pSDE==2) //Sell order
            {

   iem =
Client.sendMessage(getChgOrderCmd(pOrd_Arena,pSym,pM
kt,pPRC,pVol,pOrd,Ts,pGTD));
     try
      {
      OrdDto od=(OrdDto)iem.content;
      if (od!=null)
```

```
      modifica(iem.ktime,iem.kmsg,od.uti,od.ref);
      else modifica(iem.ktime,iem.kmsg,null,null);
      }
      catch (Exception ex)
        CallableStatement callstmt=conn.prepareCall("{call
update__order_state(?,?)}");
        callstmt.setString(1,pOrd);
        callstmt.setInt(2,2);
        callstmt.execute();
        callstmt.close();
              }

  }
```

Operational events are generated when an exchange entity is added or deleted or one of its properties is changed. For example when a trade is generated the message id is always 802 in this case and the content of the message will be a HalfTrdDto structure:

```
try {
client = new GatewayClient(gwHost, gwPort, new
GatewayClientListener() {
public void onMessageReceived(IncomingEngineMessage
iem)  {
  try
   {
   if (iem.id==802  )
     {
     HalfTrdDto htd=(HalfTrdDto)iem.content;
    if (htd.sde==1)
     {
      CallableStatement callstmt=conn.prepareCall("{call
arena.buy_trade_gw(?,?,?,?,?,?,?,?)}");
callstmt.setInt(1,htd.acc);
      callstmt.setString(2,htd.sym);
      callstmt.setString(3,htd.mkt);
      callstmt.setInt(4,htd.tck);
      if (htd.mkt.equals("ORDB"))
        callstmt.setBigDecimal(5,htd.dtp);
      else
        callstmt.setBigDecimal(5,htd.prc);
      callstmt.setInt(6,htd.ord);
      callstmt.setLong(7,htd.siz);
      callstmt.setTimestamp(8,htd.uti);
      callstmt.execute();
      callstmt.close();
    }
     [...]
   }

}
```

When we run the application, it connects to the gateway and XML is received from the gateway. The XML is deserialized and the data it contained is added in the *market* table.



4. Discussions and conclusions

The architecture shown in this paper is a primitive SOA because the integration logic occurs at the edges of the architecture, not the heart, like in the case of Enterprise Application Integration (EAI) [2, 4]. In SOA, the integrations are pushed outward, toward the applications themselves, leaving the bus to speak a standardized language.

This solution is not a true SOA solution; it only uses some of its principles, for example the existence of services for every action: enter an order, cancel an order, process conformation etc. The main problem is the lack of true loose coupling. A modern solution should by more configuration-based than code-based to assure louse coupling [7]. Most software systems in use today are code-based; Java EE 5 applications are a great example of this. In a Java EE 5 application, you write source code, compile it into an EAR or WAR file, copy that EAR or WAR file onto one or more Java EE 5 application servers, and then deploy those applications. Sometimes it's necessary to restart the Java server, depending on the nature of your deployment. Configuration-based systems work differently. There's nothing to compile or deploy. You simply change the configuration and activate those changes. Similarly, network routers and switches are configuration-based. As you make changes to their configuration, those changes take effect. There's no need for a longer software development life cycle to take place. Configuration and coding are two different strategies. Neither is superior to the other in all situations. There are times when the Java EE 5 approach is the most appropriate, and other times when the configuration-based approach is best.

In our solution, another problem can be the usually large size of the XML that can be particularly problematic for use in transmission across a network, where network bandwidth restrictions can cause significant delays in receiving the transmission. In a XML integration solution it's always a good idea to try and minimize the information that is being transmitted [1]. Given the fact that the XML is not dependent only of one of the systems, and its format is usually a compromise, the best way to overcome this problem is by assuring a good steady connection between the remote systems.

Our contribution in this project concerned remodeling the integration process between the intermediary system and the stock exchange systems. This required two main tasks. The first involved developing the interface between the database and the market exchanges, by using Java and PL/SQL like shown in this article. The second task involved re-routing several internal processes without changing too much of the existing software. For now, we can say this solution delivers and delivers well, by being powerful and reliable, but also opens the path for a full SOA integrated system.

References:

[1] Matjaz B. Juric, Ramesh Loganathan, Poornachandra Sarang,Frank Jennings: SOA Approach to Integration,XML, Web services, ESB, and BPEL in real-world SOA projects

[2] Diaconița Vlad, Botha Iuliana, Bara Adela, Lungu Ion, Velicanu Manole - Portal oriented integration in public institutions, Proceedings of the 7th WSEAS International Conference on ARTIFICIAL INTELLIGENCE, KNOWLEDGE ENGINEERING and DATA BASES (AIKED'08) University of Cambridge, Cambridge, UK, February 20-22, 2008, ISBN: 978-960-6766-41-1, ISSN: 1790-5109 (pag. 532-536)

[3] Diaconita, Vlad, Lungu Ion- TCP/IP XML Integration, The Proceedings of the 1st International Conference - Economics and Information Technology-e society Knowledge and Innovation, pg 51-58, 26-27 Sep 2008, ISBN 978-973-749-491-7

[4] Diaconita, V., Botha, I., Bara, A., Lungu, I., Velicanu, M.- Two integration flavors in public institutions, WSEAS TRANSACTIONS ON INFORMATION SCIENCE AND APPLICATIONS, Volume 5, Issue 5, May 2008, Pages 806-815 , ISSN: 1709-0832

[5] Curbera, F. Duftler, M. Khalaf, R. Nagy, W. Mukhi,N and Weerawarana, S.: Unraveling the web services web: An introduction to SOAP, WSDL, UDDI. IEEE Internet Computing, 6(2): 86-93, March-April 2002.

[6] Kohloff, Christopher and Steele, Robert: Evaluating SOAP for High Performance Business Applications: Real-Time Trading Systems, 2003,

[7]Jeff Davies, David Schorow, Samrat Ray, David Rieber- The Definitive Guide to SOA, Oracle Service Bus, SECOND EDITION, Apress 2008, ISBN-13: 978-1-4302-1057-3

[8] Arena Gateway 1.1.4 Documentation, 22.July,2008

[9] A. Bâra, I.Lungu, M. Velicanu, V. Diaconita, I. Botha – Improving query performance in virtual data warehouses, WSEAS TRANSACTIONS ON INFORMATION SCIENCE AND APPLICATIONS, May 2008, ISSN: 1790-0832 - Indexed by SCOPUS

[10] V. Diaconita, I. Lungu, A. Bara, A practical solution for stock market integration, Proceedings of the 10th WSEAS international conference on Mathematics and Computers in Business and Economy (MCBE'09 Prague) ISBN 978-960-474-063-5, ISSN 1790-5109. Indexed by ISTP/ISI Proceedings Web of Knowledge