**Marketplace Application**

- **Basic Functionality**
  - A multiagent system with independent buyers and sellers; the facilitator is an intermediary
  - All communications is via the facilitator
  - Each buyer and seller has three levels: basic, intermediate, and best
  - Depending on level, skills are either hard coded or use a rule-based system
  - The goal is to buy one guitar (hopefully for $100), one set of drums (for $200), and a second guitar (for $100)
  - In most cases, the buyer will actually pay more than these prices depending on relative skill levels

**Messages to Buy and Sell**

1. The Buyer asks the Facilitator to recommend one Seller for an item, P.
2. The Facilitator tells the Buyer the name of the Seller.
3. The Buyer asks the Seller (through the facilitator) if the Seller has an item P for sale.
4. The Seller either performs a make-offer to the Buyer (passing the item P, a unique item ID, and an initial asking price) or will deny that it has item P for sale.
5. The Buyer then can either accept the offer by echoing the offer back to the Seller or make a counter offer (with a different price) to the Seller.
6. The Seller can either accept the offer, make a counter offer, or reject the offer.
7. If the Seller accepts the offer, it will send a tell message to the Buyer. Then, the sales transaction will be complete.
8. If the Seller rejects the offer, the negotiation is over.

- There is no direct communications between the buyer and seller; this makes tracing the transactions easier
A First Example

• There is a single buyer and a single seller, both at the “basic” level
• Five second pauses allow watching the trace in real time
• At the basic level there is limited bargaining

• The facilitator trace on the next page shows three items purchased based on the initial offering price
• The marketplace trace shows the items were purchased for a total of $650, a very expensive price
The first guitar is sold for $175

The drums are sold for $300

The second guitar is sold for $175

The total price is $650

A Second Example
Both buyer & seller are in “intermediate” mode

The first guitar is sold for $125

The drums are sold for $250

The second guitar is sold for $125

The total price is $500

Ending Marketplace
The first guitar is sold for $115
The drums are sold for $240
The second guitar is sold for $115
The total price is $470

A Third Example
Both buyer & seller are in “best” mode

Marketplace Design
facilitator = FacilitatorAgent.getInstance();  // singleton
facilitator.reset();  // clear it out
facilitator.initialize();
facilitator.startAgentProcessing();

if (basicSellerCheckBoxMenuItem.isSelected()) {
basicSellerAgent = new SellerAgent();
basicSellerAgent.setTraceLevel(traceLevel);
basicSellerAgent.addCIAgentEventListener(this);
basicSellerAgent.initialize();
basicSellerAgent.startAgentProcessing();
}
if (intermediateSellerCheckBoxMenuItem.isSelected()) {
intermedSellerAgent = new BetterSellerAgent();
intermedSellerAgent.setTraceLevel(traceLevel);
intermedSellerAgent.addCIAgentEventListener(this);
intermedSellerAgent.initialize();
intermedSellerAgent.startAgentProcessing();
}
if (advancedSellerCheckBoxMenuItem.isSelected()) {
advancedSellerAgent = new BestSellerAgent();
advancedSellerAgent.setTraceLevel(traceLevel);
advancedSellerAgent.addCIAgentEventListener(this);
advancedSellerAgent.initialize();
advancedSellerAgent.startAgentProcessing();
}
if (basicBuyerCheckBoxMenuItem.isSelected()) {
basicBuyerAgent = new BuyerAgent();
basicBuyerAgent.setTraceLevel(traceLevel);
basicBuyerAgent.addCIAgentEventListener(this);
basicBuyerAgent.initialize();
basicBuyerAgent.startAgentProcessing();
}
if (intermediateBuyerCheckBoxMenuItem.isSelected()) {
intermedBuyerAgent = new BetterBuyerAgent();
intermedBuyerAgent.setTraceLevel(traceLevel);
intermedBuyerAgent.addCIAgentEventListener(this);
intermedBuyerAgent.initialize();
intermedBuyerAgent.startAgentProcessing();
}
if (advancedBuyerCheckBoxMenuItem.isSelected()) {
advancedBuyerAgent = new BestBuyerAgent();
advancedBuyerAgent.setTraceLevel(traceLevel);
advancedBuyerAgent.addCIAgentEventListener(this);
advancedBuyerAgent.initialize();
advancedBuyerAgent.startAgentProcessing();
}
void stopMenuItem_actionPerformed(ActionEvent e) {
    if (facilitator != null) {
        facilitator.removeCIAgentEventListener(this);
    }
    if (basicSellerAgent != null) {
        basicSellerAgent.stopAgentProcessing();
    }
    if (intermedSellerAgent != null) {
        intermedSellerAgent.stopAgentProcessing();
    }
    if (advancedSellerAgent != null) {
        advancedSellerAgent.stopAgentProcessing();
    }
    if (basicBuyerAgent != null) {
        basicBuyerAgent.stopAgentProcessing();
    }
    if (intermedBuyerAgent != null) {
        intermedBuyerAgent.stopAgentProcessing();
    }
    if (advancedBuyerAgent != null) {
        advancedBuyerAgent.stopAgentProcessing();
    }
    topTextArea.append("Ending Marketplace 
");
    traceTextArea.append("Ending Marketplace 
");
    this.setTitle("CIAgent Marketplace Application");
    startMenuItem.setEnabled(true);
    stopMenuItem.setEnabled(false);
}

public void processCIAgentEvent(CIAgentEvent event) {
    Object source = event.getSource();
    Object arg = event.getArgObject();
    Object action = event.getAction();

    if ((action != null) && (action.equals("trace"))) {
        if (((arg != null) && (arg instanceof String))) {
            if (source instanceof FacilitatorAgent) {
                traceFacilitator((String) arg);
            } else {
                trace((String) arg);  // display the msg
            }
        }
    }
}
package marketplace;
import java.util.*;
import javax.swing.*;
import ciagent.*;

public class FacilitatorAgent extends CIAgent {
    private static FacilitatorAgent instance = null;  // Singleton
    protected Random random = new Random();  // used to select agents
    protected Hashtable allAgents = new Hashtable();  // agent name is key, agent object is value
    protected Hashtable communities = new Hashtable();  // domain name is key, vector of agent object is value
    protected BuySellMessage msg;    // current message being processed

    protected FacilitatorAgent() {
        this("Facilitator");
    }
    protected FacilitatorAgent(String name) {
        super(name);
    }

    public void initialize() {
        setSleepTime(10 * 1000);  // sleep for 10 seconds
        setState(CIAgentState.INITIATED);
    }

    public void reset() {
        allAgents = new Hashtable();    // clear all agents
        communities = new Hashtable();  // clear all communities
    }

    public String getTaskDescription() { return null; }
    public void process() {}

    public void processTimerPop() {
        if (traceLevel > 0) trace("Facilitator: active 
    ");
    }

    public void processCIAgentEvent(CIAgentEvent e) {
        if (traceLevel > 0) {
            trace("Facilitator: CIAgentEvent received by "+ name + " from "
                    + e.getSource() + " with args " + e.getArgObject() + "\n");
        }
        Object arg = e.getArgObject();
        Object action = e.getAction();

        if ((arg != null) && (action.equals("processMessage"))) {
            msg = (BuySellMessage) arg;
            if (traceLevel > 0) {
                msg.display();  // show message contents
            }
            route(msg);
        }
    }
}
static public FacilitatorAgent getInstance() {
    if (instance == null) {
        instance = new FacilitatorAgent("Facilitator");
    } return instance;
}

public static synchronized void register(CIAgentEvent e) {
    if (instance == null) {
        instance = new FacilitatorAgent();
    }
    instance.allAgents.put(((CIAgent) e.getSource()).getName(), e.getSource());
    ((CIAgent) e.getSource()).addCIAgentEventListener(instance);
    instance.addCIAgentEventListener((CIAgent) e.getSource());
}

public synchronized void route(BuySellMessage msg) {
    CIAgent sender = (CIAgent) allAgents.get(msg.sender);
    // agent wants to say they can handle questions concerning content
    if (msg.performative.equals("advertise")) {
        trace("Facilitator: adding " + msg.sender + " to " + msg.content + " community \n");
        if (communities.containsKey(msg.content)) {
            Vector agents = (Vector) communities.get(msg.content);
            agents.addElement(sender);
        } else {
            Vector agents = new Vector();
            communities.put(msg.content, agents);
            agents.addElement(sender);
        } return;
    }
    // agent wants to remove themselves from the community
    if (msg.performative.equals("unadvertise")) {
        trace("Facilitator: removing " + msg.sender + " from " + msg.content + " community \n");
        if (communities.containsKey(msg.content)) {
            Vector agents = (Vector) communities.get(msg.content);
            agents.removeElement(sender);
            if (agents.size() == 0) {
                communities.remove(msg.content);
            }
        }
    } return;
    // agent wants facilitator to recommend an agent to process content
    // randomly pick one from the list of advertising seller agents
    if (msg.performative.equals("recommend-one")) {
        String item = msg.content;
        if (communities.containsKey(msg.content)) {
            Vector agents = (Vector) communities.get(msg.content);
            int num = agents.size();
            int index;

            if (num > 1) {
                double rand = random.nextDouble();
                index = (int) (rand * num);
            } else {
                index = 0;
            }
        }
    }
}
FacilitatorAgent - 5

BuySellMessage

```java
public class BuySellMessage {
    protected String performative; protected String content;
    protected String inReplyTo; protected String language;
    protected String ontology; protected String receiver;
    protected String replyWith; protected String sender;

    BuySellMessage(String Performative, String Content, String InReplyTo, String Language, String Ontology, String Receiver, String ReplyWith, String Sender) {
        performative = Performative; content = Content;
        inReplyTo = InReplyTo; language = Language;
        ontology = Ontology; receiver = Receiver;
        replyWith = ReplyWith; sender = Sender;
    }

    public void display() {
        System.out.println("performative: " + performative + "\n" + "content: " + content + "\n" + "inReplyTo: " + inReplyTo + "\n" + "language: " + language + "\n" + "ontology: " + ontology + "\n" + "receiver: " + receiver + "\n" + "replyWith: " + replyWith + "\n" + "sender: " + sender + "\n");
    }
}
```
public class BuyerAgent extends CIAgent {

    protected BuySellMessage msg;       // current message being processed
    protected BasicNegotiation current;
    protected Vector wishList = new Vector();
    protected BasicNegotiation pending = null;       // item waiting for seller
    protected Hashtable inventory = new Hashtable(); // items we have purchased
    protected long totalSpent = 0;                   // total money spent on items
    protected Hashtable negotiations = new Hashtable();  // transaction history

    public BuyerAgent() { this("Buyer"); }
    public BuyerAgent(String name) {  super(name);  }

    public void initialize() {
        // start a thread running and send an update message every interval secs
        trace(name + ": initialize() 
");
        msg = new BuySellMessage("register:", name, null, null, null, null);
        CIAgentEvent e = new CIAgentEvent(this, "processMessage", msg);
        FacilitatorAgent.register(e);  // add this agent to the Facilitator
        // add items to buy on wish list
        wishList.addElement(new BasicNegotiation("guitar", 100));
        wishList.addElement(new BasicNegotiation("drums", 200));
        wishList.addElement(new BasicNegotiation("guitar", 100));
        setSleepTime(5 * 1000);        // process every 5 seconds
        setState(CIAgentState.INITIATED);
    }

    public String getTaskDescription() {  return "Buying stuff...";  }
    public void process() {}

    public void processTimerPop() {
        if ((wishList.size() > 0) && (pending == null)) {
            current = (BasicNegotiation) wishList.firstElement();
            pending = current;               // we have a pending negotiation
            wishList.removeElementAt(0);
            trace(name + " is looking to buy " + current.offer.item + "\n");
            msg = new BuySellMessage("recommend-one", current.offer.item,
                null, "Seller", current.offer.item, name);
            CIAgentEvent e = new CIAgentEvent(this, "processMessage", msg);
            notifyCIAgentEventListeners(e);  // signal interested observer
            trace(name + " has purchased " + inventory.size() + " items for "
                + totalSpent + "\n");
        }
    }

    public void processCIAgentEvent(CIAgentEvent e) {
        if (traceLevel > 0) {
            trace(name + ": CIAgentEvent received by " + name + " from "
                + e.getSource() + " with args " + e.getArgObject());
        }
        Object arg = e.getArgObject();
        Object action = e.getAction();
        if ((action != null) && (arg instanceof BuySellMessage)) {
            msg = (BuySellMessage) arg;
            if ((traceLevel > 0) && (msg.getAction().equals("processMessage"))) {
                msg.display();  // show message contents
                processMessage(msg);
            }
        }
    }
}
public void processMessage(BuySellMessage msg) {
    if (msg.sender.equals("Facilitator")) {
        if (msg.performative.equals("tell")) {
            BuySellMessage answer = new BuySellMessage("ask", pending.offer.item,
            msg.replyWith, msg.content, pending.offer.item, name);
            CIAgentEvent e = new CIAgentEvent(this, "processMessage", answer);
            notifyCIAgentEventListeners(e);  // respond through Facilitator
            return;
        }
        if (msg.performative.equals("deny")) {
            trace(name + ": Seller denied our 'ask' about " + msg.content + "\n");
            wishList.addElement(pending);  // put back on wish list
            pending = null;                // try again
            return;
        }
        Offer offer = new Offer(msg);  // seller has agreed and sales transaction is complete
        trace(name + ": Offer from " + offer.sender + ": content is "+ offer.item + "\n" + offer.id + " at price of " + offer.price + " is complete \n");
        BuySellMessage answer = new BuySellMessage("tell", msg.content,
        msg.replyWith, msg.sender, offer.item, name);
        trace(name + ": " + offer.item + " purchased! \n");
        inventory.put(offer.id, current);
        totalSpent += offer.price;
        CIAgentEvent e = new CIAgentEvent(this, "processMessage", answer);
        notifyCIAgentEventListeners(e);  // respond through Facilitator
        return;
    }
    if (msg.performative.equals("accept-offer")) {
        trace(name + ": OK -- sale of item " + offer.item + " with id " + offer.id + " at price of " + offer.price + " is complete \n");
        BuySellMessage answer = new BuySellMessage("tell", msg.content,
        msg.replyWith, msg.sender, offer.item, name);
        trace(name + ": " + offer.item + " purchased! \n");
        inventory.put(offer.id, current);
        negotiations.remove(offer.id);
        totalSpent += offer.price;
        CIAgentEvent e = new CIAgentEvent(this, "processMessage", answer);
        notifyCIAgentEventListeners(e);  // respond through Facilitator
    }
    if (msg.performative.equals("make-offer")) {
        negotiate(offer, msg);
        return;
    }
    if (msg.performative.equals("reject-offer")) {
        trace(name + ": " + msg.sender + " rejected our last offer ");
        negotiations.remove(offer.id);  // remove from active list
        wishList.addElement(current);   // put back on wish list
    }
}


void negotiate(Offer offer, BuySellMessage msg) {
    if (offer.price < current.strikePrice) {

        // accept the offer -- by repeating it to the seller
        BuySellMessage answer = new BuySellMessage("make-offer",
            msg.content, msg.replyWith, msg.sender, offer.item, name);
        CIAgentEvent e = new CIAgentEvent(this,
            "processMessage", answer);

        notifyCIAgentEventListeners(e);   // respond through Facilitator
    } else {

        // make a counter offer
        current.lastOffer = offer.price - 25;  //
        BuySellMessage answer = new BuySellMessage("make-offer",
            offer.item + " " + offer.id + " " + current.lastOffer,
            offer.item, msg.sender, offer.item, name);
        CIAgentEvent e = new CIAgentEvent(this, "processMessage",
            answer);

        notifyCIAgentEventListeners(e);   // respond through Facilitator
    }
}

public class Offer {
    protected String sender; // who made the offer
    protected String item;   // what item is offered
    protected String id;     // the unique object id
    protected long price;    // the offered sales price

    Offer(String item) {
        sender = ""; this.item = item;
        id = ""; price = 0;
    }

    Offer(String sender, String item, String id, long price) {
        this.sender = sender; this.item = item;
        this.id = id; this.price = price;
    }

    Offer(BuySellMessage msg) {
        // split content into item, id, price
        StringTokenizer s = new StringTokenizer(msg.content, " "); //space delimited
        int num = s.countTokens();

        item = s.nextToken();
        id = s.nextToken();
        price = new Long(s.nextToken()).longValue();
        sender = msg.sender;
    }
}
public class SellerAgent extends CIAgent {
    private long seed = 0;
    protected BuySellMessage msg;               // current message being processed
    protected BasicNegotiation current;               // total money earned
    protected Vector inventory = new Vector();  // items we have for sale
    protected Hashtable negotiations = new Hashtable();  // transaction history

    public SellerAgent() {
        this("Seller");
    }

    public SellerAgent(String name) {
        super(name);
    }

    public String getTaskDescription() {
        return "Selling stuff...";
    }

    public void process() {}

    public void processTimerPop() {
        trace(name + " has " + inventory.size() + " items in inventory. \n");
        trace(name + " has earned " + income + ".\n");
    }

    public void initialize() {
        // start a thread running and send an update message every interval secs
        trace(name + " : initialize() \n");
        msg = new BuySellMessage("register:", name, null, null, null, null);
        CIAgentEvent e = new CIAgentEvent(this, "processMessage", msg);
        FacilitatorAgent.register(e);  // add this agent to the Facilitator
        inventory.addElement(new BasicNegotiation("guitar", 100));
        inventory.addElement(new BasicNegotiation("drums", 225));
        inventory.addElement(new BasicNegotiation("guitar", 100));
        // item and strike price

        // advertise all items for sale
        Enumeration enum = inventory.elements();
        while (enum.hasMoreElements()) {
            current = (BasicNegotiation) enum.nextElement();
            msg = new BuySellMessage("advertise", current.offer.item, null, null, current.offer.item, name);
            e = new CIAgentEvent(this, "processMessage", msg);
            notifyCIAgentEventListeners(e);  // signal interested observer
        }
        setSleepTime(15 * 1000);  // 15 seconds
        setState(CIAgentState.INITIATED);
    }
}
public void processCIAgentEvent(CIAgentEvent e) {
    if (traceLevel > 0) {
        trace(name + ": CIAgentEvent received by " + name + " from " + e.getSource() + " with args " + e.getArgObject());
    }
    Object arg = e.getArgObject();
    Object action = e.getAction();
    if ((action != null) && (action.equals("processMessage"))) {
        msg = (BuySellMessage) arg;
        if (traceLevel > 0) {
            msg.display(); // show message contents
        }
        processMessage(msg);
    }
}

String genId() {
    seed++;
    return name + seed;
}

public void processMessage(BuySellMessage msg) {
    if (msg.performative.equals("ask")) {
        String item = msg.content;
        // see if we have any items left
        if (itemInInventory(item)) {
            // start a new negotiation
            String id = genId();
            current = removeItemFromInventory(item);
            current.offer = new Offer(msg.sender, item, id, 0);
            current.lastOffer = current.strikePrice + 100;
            negotiations.put(id, current);
            BuySellMessage answer = new BuySellMessage("make-offer", item + " id + " + current.lastOffer, msg.replyWith, msg.sender, item, name);
            CIAgentEvent e = new CIAgentEvent(this, "processMessage", answer);
            notifyCIAgentEventListeners(e); // respond through Facilitator
        } else {
            // deny we have any to sell
            BuySellMessage answer = new BuySellMessage("deny", item, msg.replyWith, msg.sender, item, name);
            CIAgentEvent e = new CIAgentEvent(this, "processMessage", answer);
            trace(name + ": deny we have any " + item + " to sell to " + msg.sender + "\n");
            notifyCIAgentEventListeners(e); // respond through Facilitator
        }
    }
    return;
}

Offer offer = new Offer(msg);
trace(name + ": Offer from " + offer.sender + ": content is " + offer.item + " " + offer.id + " " + offer.price + "\n");
current = (BasicNegotiation) negotiations.get(offer.id);
if (current == null) {
    // we must have sold this item --- reject offer
    BuySellMessage answer = new BuySellMessage("reject-offer", msg.content,
        msg.replyWith, msg.sender, offer.item, name);
    CIAgentEvent e = new CIAgentEvent(this, "processMessage", answer);
    notifyCIAgentEventListeners(e); // respond through Facilitator
    return;
}
current.newOffer(offer);
// buyer has made an offer, we can accept, make a counter offer, or reject
if (msg.performative.equals("make-offer")) {
    negotiate(offer, msg);
}
// buyer has received and acknowledged our accept-offer
// remove item from active list, add $ to income
// unadvertise this item with the Facilitator
if (msg.performative.equals("tell")) {
    negotiations.remove(offer.id); // remove from negotiations list
    income += offer.price;
    BuySellMessage msg2 = new BuySellMessage("unadvertise", current.offer.item,
        null, null, current.offer.item, name);
    CIAgentEvent e = new CIAgentEvent(this, "processMessage", msg2);
    notifyCIAgentEventListeners(e); // signal interested observer
}

void negotiate(Offer offer, BuySellMessage msg) {
    if (offer.price > current.strikePrice) {
        // accept
        BuySellMessage answer = new BuySellMessage("accept-offer", msg.content,
            msg.replyWith, msg.sender, offer.item, name);
        CIAgentEvent e = new CIAgentEvent(this, "processMessage", answer);
        notifyCIAgentEventListeners(e); // respond through Facilitator
        return;
    }
    if (offer.price < current.strikePrice) {
        // reject
        rejectOffer(offer);
        return;
    }
    BuySellMessage answer = new BuySellMessage("make-offer", offer.item + " " + offer.id + " " + current.lastOffer, msg.replyWith, msg.sender, offer.item, name);
    CIAgentEvent e = new CIAgentEvent(this, "processMessage", answer);
    notifyCIAgentEventListeners(e); // respond through Facilitator
}
void rejectOffer(Offer offer) {
    BuySellMessage answer = new BuySellMessage("reject-offer", msg.content, msg.replyWith,
            msg.sender, offer.item, name);
    CIAgentEvent e = new CIAgentEvent(this, "processMessage", answer);
    notifyCIAgentEventListeners(e); // respond through Facilitator
    negotiations.remove(offer.id); // no further negotiations
    current.offer.id = null; // offer id has expired
    inventory.addElement(current); // place item back in inventory
}
boolean itemInInventory(String item) {
    Enumeration enum = inventory.elements();
    boolean haveItem = false;
    while (enum.hasMoreElements()) {
        BasicNegotiation stockItem = (BasicNegotiation) enum.nextElement();
        if (stockItem.getItem().equals(item)) {
            haveItem = true;
            break;
        } else {
            continue;
        }
    }
    return haveItem;
}

BasicNegotiation removeItemFromInventory(String item) {
    Enumeration enum = inventory.elements();
    BasicNegotiation stockItem = null;
    while (enum.hasMoreElements()) {
        stockItem = (BasicNegotiation) enum.nextElement();
        if (stockItem.getItem().equals(item)) {
            inventory.removeElement(stockItem);
            break;
        } else {
            stockItem = null; // this isn't it
            continue;
        }
    }
    return stockItem;
}

public class BestBuyerAgent extends BuyerAgent implements Effector {
    protected BooleanRuleBase rb = new BooleanRuleBase("BestBuyer");
    protected RuleVariable offerDelta;
    protected RuleVariable spread;
    protected RuleVariable firstOffer;
    protected Offer offer; // the current offer

    public BestBuyerAgent(String name) { super(name); }
    public BestBuyerAgent() { this("BestBuyer"); }

    public void initialize() {
        initBestBuyerRuleBase();
        super.initialize(); // call base class initialization
    }

    public long effector(Object obj, String eName, String args) {
        if (eName.equals("make-offer")) {
            // rule base decided to counter-offer
            long delta = (new Long(offerDelta.getValue())).longValue();
            current.lastOffer = offer.price - delta; //
            BuySellMessage answer = new BuySellMessage("make-offer", offer.item + " " + offer.id + " " + current.lastOffer, offer.item, msg.sender, offer.item, name);
            CIAgentEvent e = new CIAgentEvent(this, "processMessage", answer);
            notifyCIAgentEventListeners(e); // respond through Facilitator
            return 0;
        }
    }
}
if (eName.equals("accept-offer")) {
    // rule base decided to accept the offer; resend it to the seller
    BuySellMessage answer = new BuySellMessage("make-offer", msg.content,
        msg.replyWith, msg.sender, offer.item, name);
    CIAgentEvent e = new CIAgentEvent(this, "processMessage", answer);
    notifyCIAgentEventListeners(e);  // respond through Facilitator
    return 0;
} return 1;  // unknown effector method

void negotiate(Offer offer, BuySellMessage msg) {
    // figure out if 1st time or second time or other
    // compute spread
    // let rule base figure out our response
    // if deltaoffer is 0 then we should accept
    rb.reset();  // allow all rules to fire, clear vars
    this.offer = offer;  // make visible to rule base effectors
    if (offer.price >= current.strikePrice) {
        long delta = offer.price - current.strikePrice;
        if (delta < 25) { spread.setValue("<25");
            } else if (delta < 50) { spread.setValue("25-50");
            } else if (delta >= 50) { spread.setValue(">50");
            } else { spread.setValue("0");  // meets our price, accept
            }
if (current.prevOffer.id.equals("")) {
    firstOffer.setValue("yes");
} else {
    firstOffer.setValue("no");
}
    offerDelta.setValue(null);
    rb.forwardChain();  // inference
    if (offerDelta.getValue() == null) {
        trace(name + " rule base couldn't decide what to do.
    }
}

public void initBestBuyerRuleBase() {
    offerDelta = new RuleVariable(rb, "offerDelta");
    offerDelta.setLabels("0 25 50 100");
    firstOffer = new RuleVariable(rb, "firstOffer");
    firstOffer.setLabels("yes no");
    spread = new RuleVariable(rb, "spread");
    spread.setLabels("<25 25-50 >50");
    // Note: at this point all variables values are NULL
    Condition cEquals = new Condition("==");
    Condition cNotEquals = new Condition("!=");
    // define rules
    Rule first = new Rule(rb, "first",
        new Clause(firstOffer, cEquals, "yes"),
        new Clause(offerDelta, cEquals, "75"); // counter
    Rule second = new Rule(rb, "second", new Clause[]{
        new Clause(firstOffer, cEquals, "no"),
        new Clause(spread, cEquals, ">50")},
        new Clause(offerDelta, cEquals, "50")); // counter
Rule second2 = new Rule(rb, "second2", new Clause[]{
    new Clause(firstOffer, cEquals, "no"),
    new Clause(spread, cEquals, "25-50"),
    new Clause(offerDelta, cEquals, "25") });  // counter
Rule second3 = new Rule(rb, "second3", new Clause[]{
    new Clause(firstOffer, cEquals, "no"),
    new Clause(spread, cEquals, "<25"),
    new Clause(offerDelta, cEquals, "0") });  // accept

// action rule
Rule accept = new Rule(rb, "accept", new Clause(offerDelta, cEquals, "0"),
    new EffectorClause("accept-offer", "0");

// action rule
Rule counter = new Rule(rb, "counter",
    new Clause(offerDelta, cNotEquals, "0"),
    new EffectorClause("make-offer", null));  // use offerDelta

// define this object as effector implementor
rb.addEffector(this, "make-offer");
rb.addEffector(this, "accept-offer");
BestSellerAgent - 2

BuySellMessage answer = new BuySellMessage("make-offer", offer.item + " " + offer.id + " " + current.lastOffer, offer.item, msg.sender, offer.item, name);
CIAgentEvent e = new CIAgentEvent(this, "processMessage", answer);
notifyCIAgentEventListener(e); // respond through Facilitator
return 0;
}
if (eName.equals("reject-offer")) {
    // rule base decided to reject the offer
    rejectOffer(offer);
    return 0;
}
if (eName.equals("accept-offer")) {
    // rule base decided to accept the offer
    BuySellMessage answer = new BuySellMessage("accept-offer", msg.content, msg.replyWith, msg.sender, offer.item, name);
    CIAgentEvent e = new CIAgentEvent(this, "processMessage", answer);
    notifyCIAgentEventListener(e); // respond through Facilitator
    return 0;
}
return 1; // unknown effector method
}

void negotiate(Offer offer, BuySellMessage msg) {
    // figure out if 1st time or second time or other, then compute spread
    // let rule base figure out response; if deltaoffer is 0 then accept
    rb.reset(); // allow all rules to fire, clear vars
    this.offer = offer; // make visible to rule base effectors
    if (offer.price < current.strikePrice) {
        spread.setValue("<0"); // below asking price
    } else {
        // above asking price
        long delta = offer.price - current.strikePrice;
        if (delta < 25) { spread.setValue("0-25"); }
        else if (delta < 50) { spread.setValue("25-50"); }
        else if (delta >= 50) { spread.setValue(">50"); }
    }
    // if buyer echos the lastOffer, then he is agreeing
    if (offer.price == current.lastOffer) {
        spread.setValue(">50"); // accept via rule action
    }
    if (current.iteration == 0) { firstOffer.setValue("yes"); }
    else { firstOffer.setValue("no"); }
    current.iteration++; // increment iteration count
    if (current.iteration > 10) { // could be randomized
        rejectOffer(offer); return; // break off negotiations
    }
    rb.forwardChain(); // inference
    if (offerDelta.getValue() == null) {
        trace(name + " rule base couldn't decide what to do -- so reject it.");
        rejectOffer(offer);
    }
}
public void initBestSellerRuleBase() {
    offerDelta = new RuleVariable(rb, "offerDelta");
    offerDelta.setLabels("-100 0 30 60 100");
    firstOffer = new RuleVariable(rb, "firstOffer");
    firstOffer.setLabels("yes no");
    spread = new RuleVariable(rb, "spread");
    spread.setLabels("<0 0-25 25-50 >50");

    // Note: at this point all variables values are NULL
    Condition cEquals = new Condition("=");
    Condition cNotEquals = new Condition("!=");

    // define rules
    Rule first = new Rule(rb, "first", new Clause(firstOffer, cEquals, "yes"),
                           new Clause(offerDelta, cEquals, "50");  // counter
    Rule second1 = new Rule(rb, "second1", new Clause[]{
                              new Clause(firstOffer, cEquals, "no"),
                              new Clause(spread, cEquals, "25-50") },
                           new Clause(offerDelta, cEquals, "40");  // counter
    Rule second2 = new Rule(rb, "second2", new Clause[]{
                              new Clause(firstOffer, cEquals, "no"),
                              new Clause(spread, cEquals, "0-25") },
                           new Clause(offerDelta, cEquals, "0");  // accept
    Rule second3 = new Rule(rb, "second3", new Clause[]{
                              new Clause(firstOffer, cEquals, "no"),
                              new Clause(spread, cEquals, ">50") },
                           new Clause(offerDelta, cEquals, "0");  // accept
    Rule second4 = new Rule(rb, "second4", new Clause[]{
                              new Clause(firstOffer, cEquals, "no"),
                              new Clause(spread, cEquals, "<0") },
                           new Clause(offerDelta, cEquals, "-100");  // reject

    // action rule
    Rule accept = new Rule(rb, "accept", new Clause(offerDelta, cEquals, "0"),
                            new EffectorClause("accept-offer", "0"));

    // action rule
    Rule reject = new Rule(rb, "reject", new Clause[]{
                              new Clause(firstOffer, cEquals, "no"),
                              new Clause(offerDelta, cEquals, "-100") },
                           new EffectorClause("reject-offer", "0"));

    // action rule
    Rule counter = new Rule(rb, "counter",
                            new Clause(offerDelta, cNotEquals, "0"),
                            new EffectorClause("make-offer", null));  // use offerDelta

    // define this object as effector implementor
    rb.addEffector(this, "make-offer");
    rb.addEffector(this, "reject-offer");
    rb.addEffector(this, "accept-offer");
}