KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

DAUGIAAGENČIŲ SISTEMŲ PAGRINDAI T120B173

MAISTO PRODUKTŲ REGISTRAVIMO SISTEMA

KURSINIS DARBAS

Priėmė: dr. Laura Ostaševičiūtė

Atliko: IFC-1, Paulius Šukys

Kaunas, 2014

Turinys

Sistemos paskirtis

Maisto produktų registravimo sistema yra intelektualizuota daugiaagentė sistema, kurios tikslas yra apdoroti vartotojo produktų duomenis, juos įregistruoti, kai gaunami, išregistruoti, kai vartotojas juos suvartoja arba pastebi, kad sugedo. Sistemos pagrinde/serveryje saugomi ir tobulinami duomenys, kad vartotojas juos gautų patobulintus, labiau išsamesnius apie turimus maisto produktus. Svarbus maisto registravimo sistemos aspektas – įspėjimas apie galimai artėjantį maisto produkto galiojimo pabaigą. Pastebėtina, kad kuo ilgiau ši sistema bus naudojama, tuo paprasčiau bus vartotojui įvesti ir išvesti duomenis, o sistema sugebės tiksliau apskaičiuoti produktų galiojimo laiką.

Agentų aprašymas

Sistemos figūruoja kelių rūšių agentai. Pačiame branduolyje – serveryje, yra intelektualizuotas agentas, kuris apdoroja gautus duomenis, juos papildo naudodamasis kitų agentų resursais. Intelektualizuotam agentui padeda įvairių paslaugų, duomenų bazių agentai, kurie tiesiog grąžina apdorotus, paruoštus duomenis pagrindiniam agentui. Taip pat yra vienas komunikacijos agentas, kurio prasmė – bendrauti su išoriniais konteineriais, kurie yra klientiniai agentai.

Serverio agentai:

1. Intelektualizuotas agentas;
2. Komunikacijos agentas;
3. Yummly maisto produktų agentas;
4. Barkodų išorinės duomenų bazės agentas.

Klientiniai agentai yra tokie agentai, kurie bendrauja su serveriniu agentu ir jam pateikia duomenis, iš jo gauna duomenis. Iš esmės, jie nėra intelektualizuoti, jie tiesiog atlieka įvesties, išvesties ir perdavimo į serverio komunikatorių pareigas.

Kliento agentai:

1. Komunikacijos agentas.

Agentų sąveikos

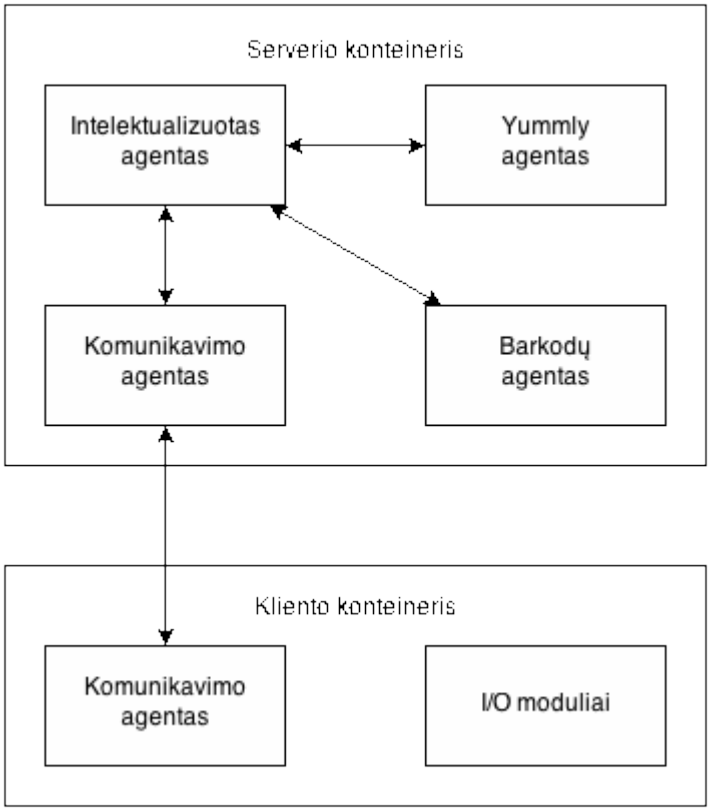
Agentai sąveikauja žvaigždės topologijos principu – visi jungiasi prie pagrindinio serverio komunikatoriaus ir iš jo gauna nurodymus, atsakymus. Pačiame serverio konteineryje – taip pat žvaigždės topologija – visi agentai jungiasi į intelektualizuotą agentą.

Agentinės sistemos architektūros aprašymas

Visa agentinė sistema yra paskirstyta į pirminį ir antrinius agentų konteinerius. Pirminis konteineris – serverio konteineris, su kuriuo visi antriniai agentų konteineriai bendrauja ir gauna/duoda duomenis.

Antrinis konteineris – kliento konteineris, kuris tiesiogiai bendrauja su pačiu klientu ir vykdo jo užklausas, kurias pateikia serverio konteineriui.

Žemiau pateiktas agentinės sistemos architektūros paveikslas (Pav.



Pav. 1: Agentinės sistemos architektūra

Serverio konteineris yra sudarytas iš 4 agentų, kurių centre intelektualizuoas agentas, kuris paskiria užduotis, tačiau tuo pačiu ir apdoroja užklausas gautas per serverio konteinerio komunikavimo agentą. Pagalbai naudojasi Yummly agentą ir Barkodų agentą.

Kliento konteineryja komunikavimo agentas yra pagrindas, nes jis perduoda tarp vartotojo (per I/O modulius) ir serverio (komunikuodamas su kitu komunikavimo agentu) informaciją.

Priedas

Serverio paketas

SmartFood.java

```
/*
 * The MIT License
 *
 * Copyright 2014 Paulius Šukys.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */
package smartfood;

import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.MongoClient;
import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.lang.acl.ACLMessage;
import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;
import jade.wrapper.StaleProxyException;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.net.UnknownHostException;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
```

```
/**
 * Main agent for Smart Food System server container
 * This class is the intelligent agent which configures it's container
 * And helping classes
 * @author Paulius Šukys
 */
public class SmartFood extends Agent
{
    private static final long serialVersionUID = 1L;
    final Logger logger = jade.util.Logger.getMyLogger(this.getClass().getName());
    //Main-Container -> Server container
    private AgentController yummly_agent;
    private AgentController comm_agent;
    private MongoClient mongo_client;
    private DB mongo_db;
    @Override
    /**
     * Includes agent initializations
     */
    protected void setup()
    {
        /**initial setup:
         * sets up mongo database
         * sets up yummly agent
         * sets up comm agent
         *
         * The setup is crucial for correct functioning, on exceptions - exit
         */
        addBehaviour(new OneShotBehaviour(this)
        {
            private static final long serialVersionUID = 1L;

            @Override
            public void action()
            {
                try
                {
                    mongo_client = new MongoClient("localhost");
                    mongo_db = mongo_client.getDB("SmartFood");
                    logger.log(Level.INFO, "Binding {0} database",
mongo_db.getName());
                    yummly_agent = createAgent("Yummly");
                    logger.log(Level.INFO, "Starting {0} agent",
yummly_agent.getName());
                    comm_agent = createAgent("Communicator");
                    logger.log(Level.INFO, "Starting {0} agent",
comm_agent.getName());
                }catch (StaleProxyException exc)
                {
                    logger.log(Level.SEVERE, exc.getMessage());
                    throw new RuntimeException("Error adding agent");
                } catch (UnknownHostException ex)
                {
                    logger.log(Level.SEVERE, ex.getMessage());
                    throw new RuntimeException("Error connecting to mongodb host");
                }
```

```
            }
        });

        /**messaging inbox:
         * retrieves messages from agents
         * return back
         */
        addBehaviour(new CyclicBehaviour(this)
        {
            private static final long serialVersionUID = 1L;

            @Override
            public void action()
            {
                String content;
                ACLMessage msg = myAgent.receive();
                if (msg != null)
                {
                    String sender_name = msg.getSender().getName();
                    switch (sender_name)
                    {
                        case "Yummly@SmartFoodSystem":
                            content = msg.getContent();
                            logger.log(Level.INFO, "Received message from {0}:
{1}", new Object[]{sender_name, content});
                            break;

                        case "Communicator@SmartFoodSystem":
                            BasicDBObject doc = new BasicDBObject();
                            JSONParser parser = new JSONParser();
                            switch (msg.getOntology())
                            {
                                case "products-request":
                                    //requests for the whole list of products
                                    switch(msg.getContent())
                                    {
                                        case "current-products":
                                            String b64str =
stringMatrixToBase64(getCurrentProducts());
                                            sendMessage(sender_name, b64str,
ACLMessage.INFORM, msg.getOntology());
                                            break;
                                        case "products":
                                            b64str =
stringMatrixToBase64(getUsedProducts());
                                            sendMessage(sender_name, b64str,
ACLMessage.INFORM, msg.getOntology());
                                            break;
                                    }
                                    break;
                                case "add-data":
                                    //add data to database

                                    try
                                    {
                                        JSONObject json =
(JSONObject)parser.parse(msg.getContent());
```

```
                                    if (json.get("product") != null)
                                    {
                                        doc.put("product",
json.get("product"));
                                    }

                                    if (json.get("barcode") != null)
                                    {
                                        doc.put("barcode",
json.get("barcode"));
                                    }

                                    if (json.get("expiry") != null)
                                    {
                                        doc.put("expiry", json.get("expiry"));
                                    }
                                } catch (ParseException ex)
                                {

Logger.getLogger(SmartFood.class.getName()).log(Level.SEVERE, null, ex);
                                }


                                DBCursor cursor =
mongo_db.getCollection("products").find(doc);
                                if (cursor.count() == 0)
                                {
                                    logger.log(Level.INFO, "Adding product: " +
msg.getContent());

mongo_db.getCollection("products").insert(doc);
                                }else
                                {
                                    logger.log(Level.INFO, "Product already
exists: " + msg.getContent());
                                }
                                //add to current products collection

mongo_db.getCollection("current_products").insert(doc);
                                break;
                            case "remove-data":
                                try
                                {
                                    JSONObject json = (JSONObject)
parser.parse(msg.getContent());
                                    if (json.get("product") != null)
                                    {
                                        doc.put("product",
json.get("product"));
                                    }

                                    if (json.get("barcode") != null)
                                    {
                                        doc.put("barcode",
json.get("barcode"));
                                    }
```

```
                                        if (json.get("expiry") != null)
                                        {
                                            doc.put("expiry", json.get("expiry"));
                                        }
                                    } catch (ParseException ex)
                                    {
Logger.getLogger(SmartFood.class.getName()).log(Level.SEVERE, null, ex);
                                    }


mongo_db.getCollection("current_products").remove(doc);
                                    break;
                                }
                                break;
                            }
                        }else
                        {

                        }
                    }

            });
        }

        @Override
        protected void takeDown()
        {
            logger.log(Level.INFO, "Agent {0} terminating.", getAID().getName());
        }

        /**
         * Creates a new agent in local container
         *
         * @param agent_name the name of the agent
         * @return AController AgentController object created in local container
         * @throws StaleProxyException when an attempt to use stale (i.e. outdated)
wrapper object is made
         */
        protected AgentController createAgent(String agent_name)
                throws StaleProxyException
        {
            AgentContainer container = getContainerController();
            AgentController AController = container.createNewAgent(agent_name,
                    this.getClass().getPackage().getName() + "." + agent_name,
                    null);
            AController.start();
            return AController;
        }

        /**
         * Gets all used products
         *
         * @return String[] array of used products
         */
        public String[][] getUsedProducts()
        {
```

```
        DBCollection product_collection  = mongo_db.getCollection("products");
        DBCursor cursor = product_collection.find();
        int product_count = (int)product_collection.getCount();
        String[][] products = new String[product_count][3];
        for(int i = 0; i < product_count; i++)
        {
            BasicDBObject obj = (BasicDBObject) cursor.next();
            products[i][0] = obj.getString("product");
            products[i][1] = obj.getString("barcode");
            products[i][2] = obj.getString("expiry");
        }
        return products;
    }

    public String[][] getCurrentProducts()
    {
        DBCollection product_collection  =
mongo_db.getCollection("current_products");
        DBCursor cursor = product_collection.find();
        int product_count = (int)product_collection.getCount();
        String[][] products = new String[product_count][3];
        for(int i = 0; i < product_count; i++)
        {
            BasicDBObject obj = (BasicDBObject) cursor.next();
            products[i][0] = obj.getString("product");
            products[i][1] = obj.getString("barcode");
            products[i][2] = obj.getString("expiry");
        }
        return products;
    }

    /**
     * Sends a message to agent
     *
     * @param to agent GUID name
     * @param content content to be sent
     * @param performative performative level
     */
    private void sendMessage(String to, String content, int performative, String
ontology)
    {
        ACLMessage msg;
        msg = new ACLMessage(performative);
        msg.addReceiver(new AID(to, true));
        msg.setOntology(ontology);
        msg.setContent(content);
        send(msg);
    }

    /**
     * Serializes a string array into base64 single string
     *
     * @param array the string array to be serialized
     * @return serialized base64 string
     */
    private String stringMatrixToBase64(String[][] array)
    {
```

```
        ByteArrayOutputStream bout = new ByteArrayOutputStream();
        try
        {
            ObjectOutputStream oout;
            oout = new ObjectOutputStream(bout);
            oout.writeObject(array);
            oout.close();
        } catch (IOException ex)
        {
            Logger.getLogger(SmartFood.class.getName()).log(Level.SEVERE, "Error in
serialization");
            Logger.getLogger(SmartFood.class.getName()).log(Level.SEVERE, null,
ex);
        }
        String b64str = Base64.encode(bout.toByteArray());
        return b64str;
    }
}
```

Communicator.java

```
/*
 * The MIT License
 *
 * Copyright 2014 Paulius Šukys.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

package smartfood;

import jade.core.AID;
import jade.core.Agent;
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.lang.acl.ACLMessage;
import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;
import jade.wrapper.StaleProxyException;
```

```
import java.util.Calendar;
import java.util.Date;
import java.util.logging.Level;
//apparently Logger is not such a good package, is it?
import java.util.logging.Logger;

/**
 * Communication agent between mobile and server containers.
 * In a simple context, it currently talks to another packet
 * @author Paulius Šukys
 */
public class Communicator extends Agent
{
    final Logger logger = jade.util.Logger.getMyLogger(this.getClass().getName());
    private static final long serialVersionUID = 1L;
    private final AID sf_aid = new AID("SmartFood@SmartFoodSystem", true);
    @Override
    protected void setup()
    {
        //apparently this is bad practice
        /**basic initiation*/
        addBehaviour(new OneShotBehaviour(this)
        {
            private static final long serialVersionUID = 1L;
           @Override
           public void action()
           {
               initMobile();
               logger.log(Level.INFO, "{0} initiated", getAID().getName());
           }
        });

        addBehaviour(new CyclicBehaviour(this)
        {

            @Override
            public void action()
            {
                //reading ALL message received
                String sender_name;
                ACLMessage msg = myAgent.receive();
                if (msg != null)
                {
                    sender_name = msg.getSender().getName();
                    switch(msg.getPerformative())
                    {
                        case ACLMessage.REQUEST:
                            //incoming agents requesting some kind of service/data
                            switch(msg.getOntology())
                            {
                                case "products-request":
                                    //asking for the list of all products
                                    sendMessage(sf_aid.getName(), msg.getContent(),
ACLMessage.REQUEST, msg.getOntology());

                                    //TODO: add asynchronized version
                                    String return_content =
```

```
waitForMessage(ACLMessage.INFORM,
                                        msg.getOntology());
                                    //return back the message
                                    sendMessage(sender_name, return_content,
                                        ACLMessage.INFORM, msg.getOntology());
                                    break;
                                default:
                                    logger.log(Level.WARNING, "Unknown message
ontology: " + msg.getOntology());
                                    break;
                            }
                            break;
                        case ACLMessage.INFORM:
                            //retrieving data from mobile platform
                            switch(msg.getOntology())
                            {
                                case "add-data":
                                    sendMessage(sf_aid.getName(), msg.getContent(),
msg.getPerformative(), msg.getOntology());
                                    break;
                                case "remove-data":
                                    sendMessage(sf_aid.getName(), msg.getContent(),
msg.getPerformative(), msg.getOntology());
                                    break;
                                default:
                                    logger.log(Level.WARNING, "Unknown message
ontology: " + msg.getOntology());
                            }
                            break;
                        default:
                            logger.log(Level.WARNING, "Unknown performative: " +
msg.getPerformative());
                            break;
                    }
                }
            }

        });
    }

    @Override
    protected void takeDown()
    {
        logger.log(Level.INFO, "Agent {0} terminating.", getAID().getName());
    }

    /**
     * A workaround for running two containers in one process
     * It is crucial, that the mobile containers is set up correctly
     */
    private void initMobile()
    {
        try
        {
            jade.core.Runtime rt = jade.core.Runtime.instance();
            jade.core.Profile p = new ProfileImpl();
            p.setParameter(Profile.MAIN_HOST, "");
```

```
            p.setParameter(Profile.MAIN_PORT, "");
            p.setParameter(Profile.CONTAINER_NAME, "Mobile");
            AgentContainer cc = rt.createAgentContainer(p);
            AgentController mobile_comm = cc.createNewAgent("Comm",
                    "smartfood.mobile.Comm", null);
            mobile_comm.start();
        }catch(StaleProxyException exc)
        {
            logger.log(Level.SEVERE, exc.getMessage());
            throw new RuntimeException("Cannot init mobile agent");
        }
    }

    /**
     * Sends a message to agent
     *
     * @param to agent GUID name
     * @param content content to be sent
     * @param performative performative level
     */
    private void sendMessage(String to, String content, int performative, String
ontology)
    {
        ACLMessage msg;
        msg = new ACLMessage(performative);
        msg.addReceiver(new AID(to, true));
        msg.setOntology(ontology);
        msg.setContent(content);
        send(msg);
    }
    /**
     * Waits for the message to return of specific performative and ontology
     *
     * @param performative message's performative
     * @param ontology message's ontology
     * @return String content
     */
    private String waitForMessage(int performative, String ontology)
    {
        ACLMessage msg = receive();
        int waitTime = 10;//seconds

        //using the time comparisson method rather than Thread.sleep()
        Calendar current_time = Calendar.getInstance();
        current_time.setTime(new Date());
        Calendar wait_time = Calendar.getInstance();
        wait_time.setTime(new Date());
        wait_time.add(Calendar.SECOND, waitTime);

        while(msg == null &&
                current_time.get(Calendar.SECOND) !=
                wait_time.get(Calendar.SECOND))
        {
            msg = receive();
            if (msg != null)
            {
                if (msg.getPerformative() != performative ||
```

```
                    !msg.getOntology().equals(ontology)) {
                msg = null;//not the message we're waiting for
            }
        }
        if (msg != null)
        {
            return msg.getContent();
        }else
        {
            logger.log(Level.SEVERE, "Waited for 10 seconds and no response!");
            return "-";
        }
    }
}
```

Yummly.java

```
/*
 * The MIT License
 *
 * Copyright 2014 Paulius Šukys.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

package smartfood;

import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;
import java.util.logging.Level;
import java.util.logging.Logger;
/**
 * Yummly service communicator.
 * It serves for the SmartFood agent and communicates with Yummly to get data.
 * @author Paulius Šukys
 */
public class Yummly extends Agent
{
    private static final long serialVersionUID = 1L;
    final Logger logger = jade.util.Logger.getMyLogger(this.getClass().getName());
```

```
    private final YummlyWrapper wrapper = new YummlyWrapper();
    @Override
    protected void setup()
    {
        addBehaviour(new OneShotBehaviour(this)
        {
            private static final long serialVersionUID = 1L;
           @Override
           public void action()
           {
               logger.log(Level.INFO, "I am {0}", getAID().getName());
               String ingredients[] = {"apple", "chocolate", "milk"};
               //List matched_recipes = wrapper.searchRecipe("pie", ingredients);
           }
        });
    }

    @Override
    protected void takeDown()
    {
        logger.log(Level.INFO, "Agent {0} terminating.", getAID().getName());
    }
}
```

YummlyWrapper.java

```
/*
 * The MIT License
 *
 * Copyright 2014 Paulius Šukys.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

package smartfood;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.net.MalformedURLException;
```

```
import java.net.URL;
import java.net.URLEncoder;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.net.ssl.HttpsURLConnection;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

/**
 * A wrapper for Yummly
 * @author Paulius Šukys
 */
public class YummlyWrapper
{
    private final Logger logger =
jade.util.Logger.getMyLogger(this.getClass().getName());
    private final static String API_URL = "https://api.yummly.com/v1/api/";
    private final static String APP_ID = "1cf18976";
    private final static String APP_KEY = "59bc08e9d8e8d840454478fbca8ae959";

    YummlyWrapper()
    {
        //exists to defeat instantiation
    }

    /**
     * Searches for a recipe and returns a list of recipes
     * @param recipe recipe name
     * @param allowedIngredients list of allowed ingredients
     * @return list of found recipes
     */
    public List searchRecipe(String recipe, String[] allowedIngredients)
    {
        String query = "recipes?q=" + recipe;
        List<Recipe> recipes;
        recipes = new ArrayList();
        try
        {
            query += getAllowedIngredients(allowedIngredients);

            String response = getResponse(API_URL + query);
            Iterator<JSONObject> iterator = getJSONmatchArray(response, "matches");

            while (iterator.hasNext())
            {
                JSONObject match = iterator.next();

                //setting up a Recipe object and appending to arraylist
                Recipe r = new Recipe();
                JSONObject flavors = (JSONObject)match.get("flavors");
                if (flavors != null)
                {
```

```
                r.addFlavor("salty", (double)flavors.get("salty"));
                r.addFlavor("meaty", (double)flavors.get("meaty"));
                r.addFlavor("sour", (double)flavors.get("sour"));
                r.addFlavor("sweet", (double)flavors.get("sweet"));
                r.addFlavor("bitter", (double)flavors.get("bitter"));
            }
            r.setRating((long)match.get("rating"));
            r.setName((String)match.get("recipeName"));
            r.setSource((String)match.get("sourceDisplayName"));
            JSONArray ing = (JSONArray)match.get("ingredients");
            Iterator<String> ii = ing.iterator();
            while(ii.hasNext())
            {
                r.addIngredient(ii.next());
            }
            r.setId((String)match.get("id"));
            recipes.add(r);
        }

    }catch(UnsupportedEncodingException exc)
    {
        logger.log(Level.SEVERE, "Encoding error");
        logger.log(Level.SEVERE, exc.getMessage());
    }
    return recipes;
}

private String getResponse(String query)
{
    StringBuilder response = new StringBuilder();
    try
    {
        URL url = new URL(query);
        HttpsURLConnection conn = (HttpsURLConnection)url.openConnection();
        conn.setRequestMethod("GET");
        conn.setRequestProperty("X-Yummly-App-ID", APP_ID);
        conn.setRequestProperty("X-Yummly-App-Key", APP_KEY);

        String res = checkResponse(conn.getResponseCode());
        if(!res.isEmpty())
        {
            System.out.println(res);
        }else
        {
            try (BufferedReader in = new BufferedReader(new InputStreamReader(
                    conn.getInputStream())))
            {
                String inputLine = in.readLine();
                while (inputLine != null)
                {
                    response.append(inputLine);
                    inputLine =in.readLine();
                }
            }
        }
        conn.disconnect();
    }catch(MalformedURLException exc)
```

```
        {
            logger.log(Level.SEVERE, "Malformed URL:");
            logger.log(Level.SEVERE, exc.getMessage());
        }catch(IOException exc)
        {
            logger.log(Level.SEVERE, "IO error");
            logger.log(Level.SEVERE, exc.getMessage());
        }
        return response.toString();
    }

    private String checkResponse(int response)
    {
        String msg = "";
        switch(response)
        {
            case 400:
                msg = "Bad request";
                break;
            case 409:
                msg = "API Rate Limit Exceeded";
                break;
            case 500:
                msg = "Internal Server Error";
                break;
            default:
                msg = "Unknown error";
                break;
        }
        return msg;
    }

    /**
     * Simply a method to make a query string for GET request
     * @param ingredients String array of ingredients
     * @return query string for GET request part
     */
    private String getAllowedIngredients(String[] ingredients) throws
UnsupportedEncodingException
    {
        String query = "";
        for (String ingredient: ingredients)
            {
                query += "&allowedIngredient[]=";
                query += URLEncoder.encode(ingredient, "UTF-8");
            }
        return query;
    }

    /**
     * Parses JSON into found queried objects iterator
     * @param content string to parse as json
     * @param query text to match in json
     * @return iterator for jsonarray
     */
    private Iterator<JSONObject> getJSONmatchArray(String content, String query)
    {
```

```
        try
        {
            //since response is in json - parse it!
            JSONParser jsonparser = new JSONParser();
            JSONObject obj = (JSONObject)jsonparser.parse(content);
            //adding all matches to array
            JSONArray matches = (JSONArray)obj.get(query);
            return matches.iterator();
        } catch (ParseException ex)
        {
            logger.log(Level.SEVERE, "Error while parsing JSON");
            logger.log(Level.SEVERE, null, ex);
        }
        return null;
    }
}
```

Recipe.java

```
/*
 * The MIT License
 *
 * Copyright 2014 Paulius Šukys.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

package smartfood;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * A recipe object for retrieved Yummly request
 * @author Paulius Šukys
 */
public class Recipe
{
    private final Map<String, Double> flavors;//flavor - 0..1 percentage
```

```
private String name;
private String source;
private int rating;
private final List ingredients;
private String id;
public Recipe()
{
    ingredients = new ArrayList();
    flavors = new HashMap<>();
    flavors.put("salty", 0.0);
    flavors.put("meaty", 0.0);
    flavors.put("sour", 0.0);
    flavors.put("sweet", 0.0);
    flavors.put("bitter", 0.0);
}
public void addFlavor(String flavor, double ratio)
{
    flavors.put(flavor, ratio);
}

public void setName(String name)
{
    this.name = name;
}

public void setSource(String source)
{
    this.source = source;
}

public void setRating(int rating)
{
    this.rating = rating;
}

public void setRating(long rating)
{
    this.rating = (int) rating;
}

public void addIngredient(String ingredient)
{
    ingredients.add(ingredient);
}

public void setId(String id)
{
    this.id = id;
}

public double getFlavor(String flavor)
{
    return flavors.get(flavor);
}

public String getName()
{
```

```
        return name;
    }

    public String getSource()
    {
        return source;
    }

    public int getRating()
    {
        return rating;
    }

    public String getIngredients(int i)
    {
        return (String)ingredients.get(i);
    }

    public String getId()
    {
        return id;
    }

    //mainly for debugging
    @Override
    public String toString()
    {
        String r = "Name: " + name + "\n";
        r += "ID: " + id + "\n";
        r += "Source: " + source + "\n";
        r += "Rating: " + rating + "\n";
        r += "Ingredients: ";
        for (Object ingredient: ingredients)
        {
            r += ingredient + "; ";
        }
        r += "\n";
        r += "Flavors:\n";
        for (Map.Entry mapEntry : flavors.entrySet())
        {
            r += "\t" + mapEntry.getKey() + ": " + mapEntry.getValue() + "\n";
        }
        return r;
    }
}
```

Mobilusis paketas

```
/*
 * The MIT License
 *
 * Copyright 2014 Paulius Šukys.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
```

```
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

package smartfood.mobile;

import com.github.sarxos.webcam.WebcamPanel;
import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;
import jade.wrapper.ControllerException;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.regex.PatternSyntaxException;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.KeyStroke;
import javax.swing.ListSelectionModel;
import javax.swing.RowFilter;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.TableRowSorter;
import net.sourceforge.jdatepicker.impl.JDatePanelImpl;
import net.sourceforge.jdatepicker.impl.JDatePickerImpl;
import net.sourceforge.jdatepicker.impl.UtilDateModel;
import org.json.simple.JSONObject;
```

```
/**
 *
 * @author
 */
public class GUI extends JFrame
{
    private static final long serialVersionUID = 1L;
    private final Logger logger = Logger.getLogger(GUI.class.getName());

    //main, consistent components
    private JMenuBar menuBar;
    private JComponent ctrlPanel;
    private JComponent mainPanel;

    private TableRowSorter<SFTableModel> sorter;
    private JTable table;
    private JScrollPane scrollPane;
    private JTextField product_text;
    private JDatePickerImpl date_picker;
    private JTextField barcode_text;
    //cache of all products
    private String table_data = "";
    //cache of currently added products
    private String table_current_data = "";


    private final Cam c;
    private final Reader r;
    private final Comm comm;

    public GUI(Comm comm) throws ControllerException
    {
        //helpers
        c = new Cam();
        r = new Reader();
        //communicator
        this.comm = comm;
        //simple data
        setName("SmartFood");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridLayout(1, 1));
        setMinimumSize(new Dimension(480, 320));

        initComponents();
        pack();
        setVisible(true);
    }

    /**
     * Initiates the graphical components
     */
    private void initComponents()
    {
        //add menu
        menuBar = initMenuBar();
        setJMenuBar(menuBar);
```

```
        //add panels
        initPanelState("view");
    }

    /**
     * Initiates the menu bar with menus and menu items
     * @return main menu bar
     */
    private JMenuBar initMenuBar()
    {
        JMenuBar mb = new JMenuBar();
        JMenu manage_menu = new JMenu("Manage");
        manage_menu.setMnemonic(KeyEvent.VK_M);
        manage_menu.getAccessibleContext().setAccessibleDescription("Manage
products");
        mb.add(manage_menu);

        //menu items
        JMenuItem mitem = new JMenuItem("Add product");
        mitem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_1,
ActionEvent.ALT_MASK));
        mitem.getAccessibleContext().setAccessibleDescription("Register a new
product to SmartFood system");
        mitem.addActionListener(new ActionListener()
        {

            @Override
            public void actionPerformed(ActionEvent e)
            {
                //opens add product
                initPanelState("add");
            }

        });
        manage_menu.add(mitem);

        mitem = new JMenuItem("Remove product");
        mitem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_2,
ActionEvent.ALT_MASK));
        mitem.getAccessibleContext().setAccessibleDescription("Remove an old/used
product from SmartFood system");
        mitem.addActionListener(new ActionListener()
        {

            @Override
            public void actionPerformed(ActionEvent e)
            {
                //opens remove product
                initPanelState("remove");
            }

        });
        manage_menu.add(mitem);

        mitem = new JMenuItem("View current products");
        mitem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_3,
```

```
ActionEvent.ALT_MASK));
        mitem.getAccessibleContext().setAccessibleDescription("View currently
registered products");
        mitem.addActionListener(new ActionListener()
        {

            @Override
            public void actionPerformed(ActionEvent e)
            {
                //opens view products list
                initPanelState("view");
            }

        });
        manage_menu.add(mitem);

        return mb;

    }

    /**
     * Simply reinitiates the state of main panel and control panel
     * @param state name of state (add, remove, view)
     */
    private void initPanelState(String state)
    {
        if (mainPanel != null)
        {
            remove(mainPanel);
        }
        mainPanel = makePanel();
        ctrlPanel = initControlPanel(state);
        mainPanel.add(ctrlPanel);
        add(mainPanel);
        pack();
    }

    /**
     * Initiates the control panel for definable action
     * @param action
     * @return
     */
    private JComponent initControlPanel(String action)
    {
        JComponent panel = makePanel();

        switch(action)
        {
            case "add":
            case "remove":
                panel.setLayout(new BoxLayout(panel, BoxLayout.PAGE_AXIS));

                JButton barcode_button = getBarcodeButton(action);
                JButton input_button  = getInputProdButton(action);
                panel.add(barcode_button);
                panel.add(input_button);
                break;
```

```
            case "view":
                //TODO: create viewable table
                break;
            default:
                logger.log(Level.WARNING,
                        "Unknown action used for initiating ctrl panel");
        }
        return panel;
    }

    /**
     * General method for getting active barcode reader button which inhibits
     * webcam panel, reads the barcode, removes the panel and sends the data to
server
     * @param action - 'add' or 'remove' action
     * @return "Read barcode" button
     */
    private JButton getBarcodeButton(final String action)
    {
        JButton button = new JButton("Read barcode");
        button.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent e)
            {
                Thread t = new Thread()
                {
                    @Override
                    public void run()
                    {
                        try
                        {
                            enableComponents(ctrlPanel, false);
                            WebcamPanel webcam_panel = c.getPanel(true, true);
                            mainPanel.add(webcam_panel);
                            pack();

                            //try to get the barcode
                            String barcode = retrieveBarcode();
                            JSONObject data = new JSONObject();
                            data.put("barcode", barcode);
                            data.put("expiry", "");
                            data.put("product", "");
                            switch(action)
                            {
                                case "add":
                                    comm.addData(data.toString());
                                    break;
                                case "remove":
                                    comm.removeData(data.toString());
                                    break;
                                default:
                                    logger.log(Level.WARNING, "Unknown tab name!");
                            }
                            //get pack to previous state
                            mainPanel.remove(webcam_panel);
                            c.stopWebcam();
```

```
                            enableComponents(ctrlPanel, true);
                            pack();
                        }catch (InterruptedException exc)
                        {
                            logger.log(Level.WARNING, "Thread has been
interrupted!");
                        }
                    }
                };
                t.start();
            }
        });
        return button;
    }

    /**
     * General method for getting active input reader button which inhibits live
     * product name and barcode search, reads the inputand sends the data to server
     * @param action 'add' or 'remove' action
     * @return "Input product" button
     */
        private JButton getInputProdButton(final String action)
    {
        JButton button = new JButton("Input product");
        button.addActionListener(new ActionListener()
        {

            @Override
            public void actionPerformed(ActionEvent e)
            {
                Thread t = new Thread()
                {
                    @Override
                    public void run()
                    {
                        try
                        {
                            enableComponents(ctrlPanel, false);
                            /*ask for asynchronous data retrieval. When retrieved
                             *the class global "table_data" will be changed*/
                            switch(action)
                            {
                                case "add":
                                    comm.getData("products");
                                    break;
                                case "remove":
                                    comm.getData("current-products");
                                    break;
                            }
                            waitForDataRetrieval(action, 10);
                            product_text = getTextField("Product name");

product_text.getDocument().addDocumentListener(getDataTableDocumentListener(action,
"input"));
                            date_picker = getDatePicker();
                            barcode_text = getTextField("Barcode");
```

```
barcode_text.getDocument().addDocumentListener(getDataTableDocumentListener(action,
"barcode"));
                            JButton submit_button = getSubmitButton(action);
                            ctrlPanel.add(product_text);
                            ctrlPanel.add(barcode_text);
                            ctrlPanel.add(date_picker);
                            ctrlPanel.add(submit_button);

                            //I hate switches
                            String data = "";
                            switch(action)
                            {
                                case "add":
                                    data = table_data;
                                    break;
                                case "remove":
                                    data = table_current_data;
                                    break;
                            }

                            if (!data.equals("") && getProducts(data).length != 0)
                            {
                                String[][] products = getProducts(data);
                                scrollPane = getDataTableSP(action, products);
                                mainPanel.add(scrollPane);
                            }
                            pack();
                        }catch(InterruptedException exc)
                        {
                            logger.log(Level.WARNING, "Thread has been
interrupted");
                        }
                    }
                };
                t.start();
            }

        });
        return button;
    }

    /**
     * Gets a text field with a placholder
     * @param placeholder text to be put in placholder
     * @return text field object
     */
    private JTextField getTextField(String placeholder)
    {
        JTextField tf = new JTextField();
        tf.setPreferredSize(new Dimension(80, 20));
        TextPrompt tp = new TextPrompt(placeholder, tf);
        return tf;
    }

    /**
     * Gets the date picker implementation
     * @return date picker
```

```
 */
private JDatePickerImpl getDatePicker()
{
    return new JDatePickerImpl(new JDatePanelImpl(new UtilDateModel()));
}

/**
 * Waits for table_data variable to be populated
 * @param waitSeconds how much seconds to wait
 */
private void waitForDataRetrieval(String tab, int waitSeconds)
{
    Calendar current_time = Calendar.getInstance();
    current_time.setTime(new Date());
    Calendar wait_time = Calendar.getInstance();
    wait_time.setTime(new Date());
    wait_time.add(Calendar.SECOND, waitSeconds);
    switch(tab)
    {
        case "add":
            while(table_data.equals("") &&
            current_time.get(Calendar.SECOND) !=
            wait_time.get(Calendar.SECOND))
            break;
        case "remove":
            while(table_current_data.equals("") &&
            current_time.get(Calendar.SECOND) !=
            wait_time.get(Calendar.SECOND))
            break;
    }


}

/**
 * Creates a sortable table model with populated data
 * @param tableData data to populate
 * @return scroll pane  for the table model
 */
private JScrollPane getDataTableSP(String tab, String[][] tableData)
{
    SFTableModel model = new SFTableModel();
    model.setData(tableData);
    JScrollPane pane = new JScrollPane();
    sorter = new TableRowSorter<>(model);
    table = new JTable(model);
    table.setRowSorter(sorter);
    table.setFillsViewportHeight(true);
    table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    pane = new JScrollPane(table);
    return pane;
}

/**
 * Creates a simple data table document listener which updates on any action
 * @return the document listener
 */
```

```
    private DocumentListener getDataTableDocumentListener(final String tab, final
String target)
    {
        DocumentListener dl = new DocumentListener()
        {

            @Override
            public void insertUpdate(DocumentEvent e)
            {
                filterData(target);
            }

            @Override
            public void removeUpdate(DocumentEvent e)
            {
                filterData(target);
            }

            @Override
            public void changedUpdate(DocumentEvent e)
            {
                filterData(target);
            }
        };
        return dl;
    }

    /**
     * Returns a tab specific submit button
     * @param tab tab name
     * @return button for submission
     */
    private JButton getSubmitButton(final String action)
    {
        JButton button = new JButton("Submit");
        button.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent e)
            {
                try
                {
                    if (scrollPane != null && table.getSelectedRow() != -1)
                    {
                        JSONObject data = new JSONObject();
                        data.put("product",
table.getValueAt(table.getSelectedRow(), 0).toString().trim());
                        data.put("barcode",
table.getValueAt(table.getSelectedRow(), 1).toString().trim());
                        data.put("expiry", table.getValueAt(table.getSelectedRow(),
2).toString().trim());

                        comm.addData(data.toString());
                        logger.log(Level.INFO, "Adding {0}",
                                table.getValueAt(table.getSelectedColumn(),
0).toString());
                    }else
```

```
                    {
                        JSONObject data = new JSONObject();
                        data.put("product", product_text.getText().trim());
                        data.put("barcode", barcode_text.getText().trim());
                        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
                        if (date_picker.getModel().getValue() == null)
                        {
                            data.put("expiry", "");
                        }else
                        {
                            data.put("expiry",
sdf.format(date_picker.getModel().getValue()));
                        }
                        comm.addData(data.toString().trim());
                        logger.log(Level.INFO,
                                "Adding {0}", product_text.getText());
                    }
                    enableComponents(ctrlPanel, true);
                    initPanelState(action);
                    //refresh data
                    switch(action)
                    {
                        case "add":
                            comm.getData("products");
                            break;
                        case "remove":
                            comm.getData("current-products");
                    }
                }catch(InterruptedException exc)
                {
                    logger.log(Level.WARNING, "Thread has been interrupted");
                }
            }
        });
        return button;
    }


    private void filterData(String whatToFilter)
    {
        try
        {
            RowFilter<SFTableModel, Object> rf = null;
            switch(whatToFilter)
            {
                case "barcode":
                    rf = RowFilter.regexFilter(product_text.getText(), 0);
                    break;
                case "input":
                    rf = RowFilter.regexFilter(barcode_text.getText(), 0);
                    break;
            }
            sorter.setRowFilter(rf);
        }catch(PatternSyntaxException ex)
        {
            logger.log(Level.SEVERE, null, ex);
            return;
```

```
    }
}

/**
 * Creates a panel with label
 * @return panel
 */
private JComponent makePanel()
{
    JPanel panel = new JPanel(false);
    return panel;
}

/**
 * Notifies the user with a notification about something
 * @param content notification text
 */
public void notify(String content)
{

}

/**
 * Retrieves serialized base64 string and decodes and
 * de-serializes it into String array
 * @param serString
 * @return array of products as in strings
 */
private String[][] getProducts(String serString)
{
    try
    {
        if (serString.equals(""))
        {
            return new String[0][0];
        }
        ByteArrayInputStream in;
        in = new ByteArrayInputStream(Base64.decode(serString));
        String[][] ret = (String[][]) new ObjectInputStream(in).readObject();
        return ret;
    } catch (IOException | ClassNotFoundException ex)
    {
        logger.log(Level.SEVERE, null, ex);
    }
    return null;
}

private void enableComponents(JComponent panel, boolean enable)
{
    Component[] components = panel.getComponents();
    for (Component c : components)
    {
        c.setEnabled(enable);
    }
}

private String retrieveBarcode()
```

```
    {
        try
        {
            String barcode = r.readImage(c.takePicture());
            while (barcode.equals(""))
            {
                barcode = r.readImage(c.takePicture());
            }
            return barcode;
        } catch (IOException ex)
        {
            Logger.getLogger(GUI.class.getName()).log(Level.SEVERE, null, ex);
            throw new RuntimeException("Error while trying to take a picture");
        }
    }

    public void setCurrentData(String data)
    {
        table_current_data = data;
    }

    public void setTableData(String data)
    {
        table_data = data;
    }
}

class SFTableModel extends AbstractTableModel
{
    private static final long serialVersionUID = 1L;
    private final String[] columNames = {"Product name", "Barcode", "Expiry date"};

    private String[][] data;
    @Override
    public int getRowCount()
    {
        return data.length;
    }

    @Override
    public int getColumnCount()
    {
        return columNames.length;
    }

    @Override
    public String getColumnName(int col)
    {
        return columNames[col];
    }
    @Override
    public Object getValueAt(int i, int i1)
    {
        return data[i][i1];
    }

    @Override
```

```
    public Class getColumnClass(int c)
    {
        return getValueAt(0, c).getClass();
    }

    public void setData(String[][] data)
    {
        this.data = data;
    }

    public String[][] getData()
    {
        return data;
    }
}
```

Comm.java

```
/*
 * The MIT License
 *
 * Copyright 2014 Paulius Šukys.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

package smartfood.mobile;

import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.lang.acl.ACLMessage;
import jade.wrapper.AgentController;
import jade.wrapper.ContainerController;
import jade.wrapper.ControllerException;
import jade.wrapper.StaleProxyException;
import java.util.Calendar;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;
```

```
/**
 *
 * @author
 */
public class Comm extends Agent
{
    final AID server_comm = new AID("Communicator@SmartFoodSystem", true);
    final Logger logger = jade.util.Logger.getMyLogger(this.getClass().getName());
    private ContainerController cc;
    private static final long serialVersionUID = 1L;
    private GUI gui;

    @Override
    protected void setup()
    {
        addBehaviour(new OneShotBehaviour(this)
        {
            private static final long serialVersionUID = 1L;
           @Override
           public void action()
           {
               //creates his container
               cc = getContainerController();
                try
                {
                    gui = new GUI((Comm)myAgent);
                } catch (ControllerException ex)
                {
                    Logger.getLogger(Comm.class.getName()).log(Level.SEVERE, null,
ex);
                }
           }
        });
        //reading messages
        addBehaviour(new CyclicBehaviour()
        {
            private static final long serialVersionUID = 1L;
            @Override
            public void action()
            {
                String name, content;
                ACLMessage msg = myAgent.receive();
                if (msg != null)
                {
                    name = msg.getSender().getName();
                    if (name.equals("Communicator@SmartFoodSystem"))
                    {
                        switch(msg.getPerformative())
                        {
                            case ACLMessage.REQUEST:
                                logger.log(Level.INFO, "Server request received,
ID: " + msg.getConversationId());
                                break;
                            case ACLMessage.INFORM:
                                //give info
                                switch(msg.getOntology())
```

```
                                {
                                    case "products-request":
                                        gui.setTableData(msg.getContent());
                                        break;
                                    case "current-products-request":
                                        gui.setCurrentData(msg.getContent());
                                    case "notification":
                                        break;
                                }
                                break;
                        }
                    }else
                    {
                        logger.log(Level.WARNING, "A non serverish communicator
message received.");
                    }
                }
            }
        });
    }

    @Override
    protected void takeDown()
    {
        logger.log(Level.INFO, "Agent {0} terminating.", getAID().getName());
    }

    protected AgentController createAgent(String agent_name)
    {

        AgentController AController = null;
        try
        {
            AController = cc.createNewAgent(agent_name,
                    this.getClass().getPackage().getName() + "." + agent_name,
                    null);
            AController.start();
        }catch(StaleProxyException exc)
        {
            logger.log(Level.SEVERE, "Problem creating new agent.");
            logger.log(Level.SEVERE, exc.getMessage());
        }
        return AController;
    }

    /**
     * Gets data from the main container
     * @param dataName
     * @return
     * @throws java.lang.InterruptedException
     */
    public void getData(String data) throws InterruptedException
    {

        sendMessage(server_comm.getName(), data, ACLMessage.REQUEST, "products-
request");
    }
```

```
    public void addData(String content) throws InterruptedException
    {
        sendMessage(server_comm.getName(), content, ACLMessage.INFORM, "add-data");
    }

    public void removeData(String content) throws InterruptedException
    {
        sendMessage(server_comm.getName(), content, ACLMessage.INFORM, "remove-
data");
    }

    /**
     * Sends a message to agent
     *
     * @param to agent GUID name
     * @param content content to be sent
     * @param performative performative level
     */
    private void sendMessage(String to, String content, int performative, String
ontology)
    {
        ACLMessage msg;
        msg = new ACLMessage(performative);
        msg.addReceiver(new AID(to, true));
        msg.setOntology(ontology);
        msg.setContent(content);
        send(msg);
    }
    /**
     * Waits for the message to return of specific performative and ontology
     * @param performative message's performative
     * @param ontology message's ontology
     * @return String content
     */
    private String waitForMessage(int performative, String ontology)
    {
        ACLMessage msg = receive();
        int waitTime = 10;//seconds

        //using the time comparisson method rather than Thread.sleep()
        Calendar current_time = Calendar.getInstance();
        current_time.setTime(new Date());
        Calendar wait_time = Calendar.getInstance();
        wait_time.setTime(new Date());
        wait_time.add(Calendar.SECOND, waitTime);

        while(msg == null &&
                current_time.get(Calendar.SECOND) !=
                wait_time.get(Calendar.SECOND))
        {
            msg = receive();
            if (msg != null)
            {
                if (msg.getPerformative() != performative ||
                        !msg.getOntology().equals(ontology)) {
                    msg = null;//not the message we're waiting for
```

```
                }
            }
        }
        if (msg != null)
        {
            return msg.getContent();
        }else
        {
            logger.log(Level.SEVERE, "Waited for 10 seconds and no response!");
            return "-";
        }
    }
}
```

Cam.java

```
/*
 * The MIT License
 *
 * Copyright 2014 Paulius Šukys.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

package smartfood.mobile;

import com.github.sarxos.webcam.Webcam;
import com.github.sarxos.webcam.WebcamPanel;
import java.awt.image.BufferedImage;
/**
 * A general use class for interacting with a webcam.
 * Using sarxos.webcam lib
 * @author Paulius Šukys
 */
public class Cam
{
    /**
     * Takes a single picture
     * @return picture
     */
    public BufferedImage takePicture()
```

```
    {
        Webcam.setAutoOpenMode(true);
        BufferedImage image = getWebcam().getImage();
        return image;
    }

    /**
     * Gets a view panel for webcam
     * @param fps_display display frames per second
     * @param auto_start automatically start the webcam
     * @return view panel for webcam
     */
    public WebcamPanel getPanel(boolean fps_display, boolean auto_start)
    {
        WebcamPanel panel = new WebcamPanel(getWebcam(), auto_start);
        panel.setFPSDisplayed(fps_display);
        return panel;
    }

    /**
     * somewhat oop to return webcam
     * @return webcam
     */
    private Webcam getWebcam()
    {
        return Webcam.getDefault();
    }

    public void stopWebcam()
    {
        getWebcam().close();
    }
}
```

Reader.java

```
/*
 * The MIT License
 *
 * Copyright 2014 Paulius Šukys.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
```

```
 * THE SOFTWARE.
 */

package smartfood.mobile;

import com.google.zxing.BinaryBitmap;
import com.google.zxing.LuminanceSource;
import com.google.zxing.MultiFormatReader;
import com.google.zxing.NotFoundException;
import com.google.zxing.Result;
import com.google.zxing.client.j2se.BufferedImageLuminanceSource;
import com.google.zxing.common.HybridBinarizer;
import java.awt.image.BufferedImage;
import java.io.FileNotFoundException;
import java.io.IOException;

/**
 * Barcode reader
 * @author Paulius Šukys
 */
public class Reader
{
    /**
     * Reads an image and tries to find barcode
     * @param image bufferedimage object
     * @return barcode
     * @throws IOException if there's problem with reading bufferedimage object
     */
    public String readImage(BufferedImage image) throws IOException
    {
        LuminanceSource source = new BufferedImageLuminanceSource(image);
        MultiFormatReader barcodeReader = new MultiFormatReader();
        BinaryBitmap bitmap = new BinaryBitmap(new HybridBinarizer(source));
        Result result;
        String finalResult = "";
        try
        {
            result = barcodeReader.decode(bitmap);
            finalResult = String.valueOf(result.getText());
        }catch (NotFoundException e)
        {
            //not needed
        }
        return finalResult;
    }
}
```

TextPrompt.java

```
package smartfood.mobile;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import javax.swing.text.*;
```

```
/**
 *  The TextPrompt class will display a prompt over top of a text component when
 *  the Document of the text field is empty. The Show property is used to
 *  determine the visibility of the prompt.
 *
 *  The Font and foreground Color of the prompt will default to those properties
 *  of the parent text component. You are free to change the properties after
 *  class construction.
 */
public class TextPrompt extends JLabel
      implements FocusListener, DocumentListener
{
      public enum Show
      {
            ALWAYS,
            FOCUS_GAINED,
            FOCUS_LOST;
      }

      private JTextComponent component;
      private Document document;

      private Show show;
      private boolean showPromptOnce;
      private int focusLost;

      public TextPrompt(String text, JTextComponent component)
      {
            this(text, component, Show.ALWAYS);
      }

      public TextPrompt(String text, JTextComponent component, Show show)
      {
            this.component = component;
            setShow( show );
            document = component.getDocument();

            setText( text );
            setFont( component.getFont() );
            setForeground( component.getForeground() );
            setBorder( new EmptyBorder(component.getInsets()) );
            setHorizontalAlignment(JLabel.LEADING);

            component.addFocusListener( this );
            document.addDocumentListener( this );

            component.setLayout( new BorderLayout() );
            component.add( this );
            checkForPrompt();
      }

      /**
       *  Convenience method to change the alpha value of the current foreground
       *  Color to the specifice value.
       *
       *  @param alpha value in the range of 0 - 1.0.
```

```
 */
public void changeAlpha(float alpha)
{
      changeAlpha( (int)(alpha * 255) );
}

/**
 *  Convenience method to change the alpha value of the current foreground
 *  Color to the specifice value.
 *
 *  @param alpha value in the range of 0 - 255.
 */
public void changeAlpha(int alpha)
{
      alpha = alpha > 255 ? 255 : alpha < 0 ? 0 : alpha;

      Color foreground = getForeground();
      int red = foreground.getRed();
      int green = foreground.getGreen();
      int blue = foreground.getBlue();

      Color withAlpha = new Color(red, green, blue, alpha);
      super.setForeground( withAlpha );
}

/**
 *  Convenience method to change the style of the current Font. The style
 *  values are found in the Font class. Common values might be:
 *  Font.BOLD, Font.ITALIC and Font.BOLD + Font.ITALIC.
 *
 *  @param style value representing the the new style of the Font.
 */
public void changeStyle(int style)
{
      setFont( getFont().deriveFont( style ) );
}

/**
 *  Get the Show property
 *
 *  @return the Show property.
 */
public Show getShow()
{
      return show;
}

/**
 *  Set the prompt Show property to control when the promt is shown.
 *  Valid values are:
 *
 *  Show.AWLAYS (default) - always show the prompt
 *  Show.Focus_GAINED - show the prompt when the component gains focus
 *      (and hide the prompt when focus is lost)
 *  Show.Focus_LOST - show the prompt when the component loses focus
 *      (and hide the prompt when focus is gained)
 *
```

```
 *  @param show a valid Show enum
 */
public void setShow(Show show)
{
      this.show = show;
}

/**
 *  Get the showPromptOnce property
 *
 *  @return the showPromptOnce property.
 */
public boolean getShowPromptOnce()
{
      return showPromptOnce;
}

/**
 *  Show the prompt once. Once the component has gained/lost focus
 *  once, the prompt will not be shown again.
 *
 *  @param showPromptOnce  when true the prompt will only be shown once,
 *                         otherwise it will be shown repeatedly.
 */
public void setShowPromptOnce(boolean showPromptOnce)
{
      this.showPromptOnce = showPromptOnce;
}

/**
 *    Check whether the prompt should be visible or not. The visibility
 *  will change on updates to the Document and on focus changes.
 */
private void checkForPrompt()
{
      //  Text has been entered, remove the prompt

      if (document.getLength() > 0)
      {
            setVisible( false );
            return;
      }

      //  Prompt has already been shown once, remove it

      if (showPromptOnce && focusLost > 0)
      {
            setVisible(false);
            return;
      }

      //  Check the Show property and component focus to determine if the
      //  prompt should be displayed.

  if (component.hasFocus())
  {
      if (show == Show.ALWAYS
```

```
            ||  show == Show.FOCUS_GAINED)
                  setVisible( true );
            else
                  setVisible( false );
        }
        else
        {
            if (show == Show.ALWAYS
            ||  show == Show.FOCUS_LOST)
                  setVisible( true );
            else
                  setVisible( false );
        }
      }

//  Implement FocusListener

      public void focusGained(FocusEvent e)
      {
            checkForPrompt();
      }

      public void focusLost(FocusEvent e)
      {
            focusLost++;
            checkForPrompt();
      }

//  Implement DocumentListener

      public void insertUpdate(DocumentEvent e)
      {
            checkForPrompt();
      }

      public void removeUpdate(DocumentEvent e)
      {
            checkForPrompt();
      }

      public void changedUpdate(DocumentEvent e) {}
}
```