

HETS - Home Energy Trading System

Team name: Swin-HETS

Julian David Schaefer (101985593)

Malaka Perera (101065042)

Robert Bagnato (100585471)

COS30018 - Intelligent Systems

Due: 2nd November 2018

1. Introduction	3
2. Overall System Architecture	3
2.1 Logic Package	3
Appliance Agent	4
Home Agent	4
Retailer Agent	5
Agent communication	5
2.2 Prediction Package	6
Neural Network Manager	6
Neural Network	7
2.3 Integration of Forecasts/Predictions into Appliances	7
2.4 UI Package	9
GUI communication with JADE	10
3. Implemented Interaction Protocols	11
3.1 FIPA Request Interaction Protocol	11
3.2 FIPA Iterated Contract Net Interaction Protocol	12
4. Implemented Negotiation Strategies	12
4.1 Retailer Tariffs	13
Random Tariff	13
Block Tariff	13
Time of Use Tariff	14
4.2 Negotiation Strategies	14
Fixed Price	14
Time-Dependent	15
Random Tit-for-Tat	16
4.3 The Actual Negotiation	16
5. Implemented Prediction Algorithms	16
5.1 Overall Design	16
5.2 Activation Function	17
5.3 Training	18
6. Scenarios and Examples of the System Working	18
7. Critical Analysis of Performance With and Without Negotiation	19
7.1 No Home Agent Negotiation	19
7.2 No Retailer Agent Negotiation	19
7.3 Ideal Negotiation Scenario	19
8. Conclusion	21
Appendix	22
1.1 Team Contribution	22

1.2 UI Mockups

25

1. Introduction

With the advancement of AI in recent years, there have been many products introduced to the market to help manage various aspects of our daily lives. In this project we aimed to implement a Home Energy Trading System that is able to effectively measure and manage the usage of energy in your home.

The system uses past readings to predict the energy demand for the next time period and automatically purchases energy from different retailers based on the usage forecasts in order to minimise the costs of energy in the home. During the process of purchasing energy, multiple negotiation strategies are used to bargain with different energy selling retailers and get the best price possible for the allocated time period. Given a scenario in which the user has a solar panel or other power generating appliance present in the home, the home agent will also be able to either sell energy or use it as part of its own surplus.

In order to run prediction algorithms and run simulations, electricity usage and weather data sets were outsourced from the Harvard Dataverse and act as the basis for all results presented here.

2. Overall System Architecture

The system is divided into four loosely coupled packages:

1. Logic package
2. Prediction package
3. Negotiation package
4. GUI package

These packages are split up according to their responsibility and communicate with each other using simple interfaces.

2.1 Logic Package

This package contains all the agent related infrastructure. The Home Energy Trading System has been built using software agents. To implement these agents, we used JADE, an open source Java Framework to develop peer-to-peer agent based applications.

The Home Energy Trading system essentially consists of appliances, homes and retailers. Therefore, we implemented three different agents in order to represent them:

1. Appliance Agent
2. Home Agent
3. Retailer Agent

Appliance Agent

The appliance agent is a very basic agent. It is only responsible for providing the Home agent with its actual energy usages as well as usage forecasts for future periods. For that reason, it only has one behaviour, the **ApplianceResponderBehaviour**, which simply responds to the usage forecast requests or the actual usage requests, respectively, by the home agent.

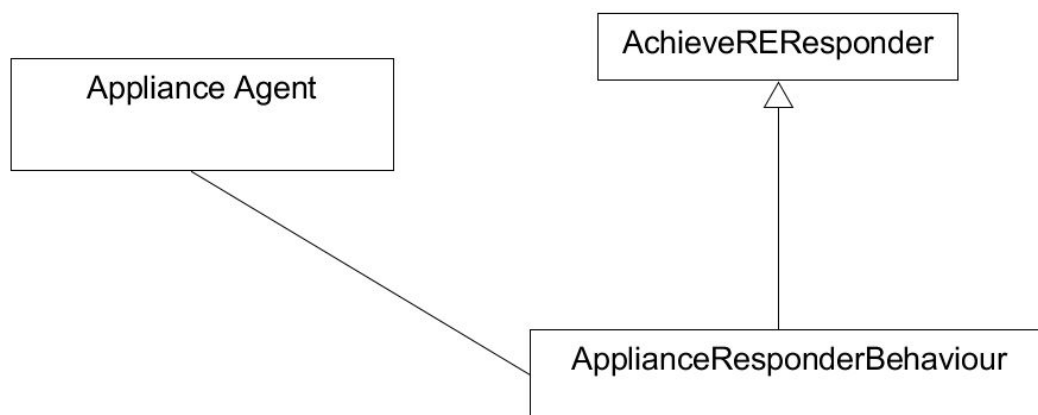


Figure 1: Appliance Agent Structure Diagram

Home Agent

The Home Agent is the mediator between the appliances and the retailers and it has the biggest impact on the overall performance of the system. Hence, it is much more complex than the appliance agent and has more responsibilities. It is responsible for collecting and aggregating the actual usages as well as the forecasts from all of its appliances periodically. After that, it then negotiates with different retailer agents in order to get the best price according to the usage forecast or to sell the energy that the home has produced.

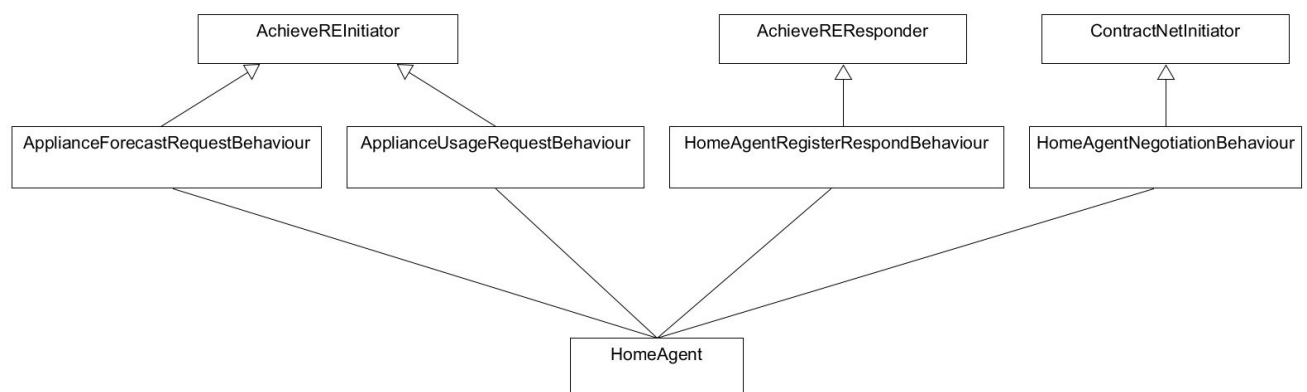


Figure 2: Home Agent Structure Diagram

Retailer Agent

The retailer agent is responsible for selling and buying energy to home agents. It accepts negotiation requests from home agents and then negotiates with them to either sell or buy energy. For that, it uses two behaviours: The **RetailerResponderBehaviour** responds to all negotiation requests that it gets from a home agent. For every negotiation request it gets, it creates a new **RetailerNegotiationBehaviour**. The **RetailerNegotiationBehaviour** then handles the request and is responsible for the actual negotiation with the specific home agent. Hence, it represents a single negotiation session between a home agent and the retailer agent.

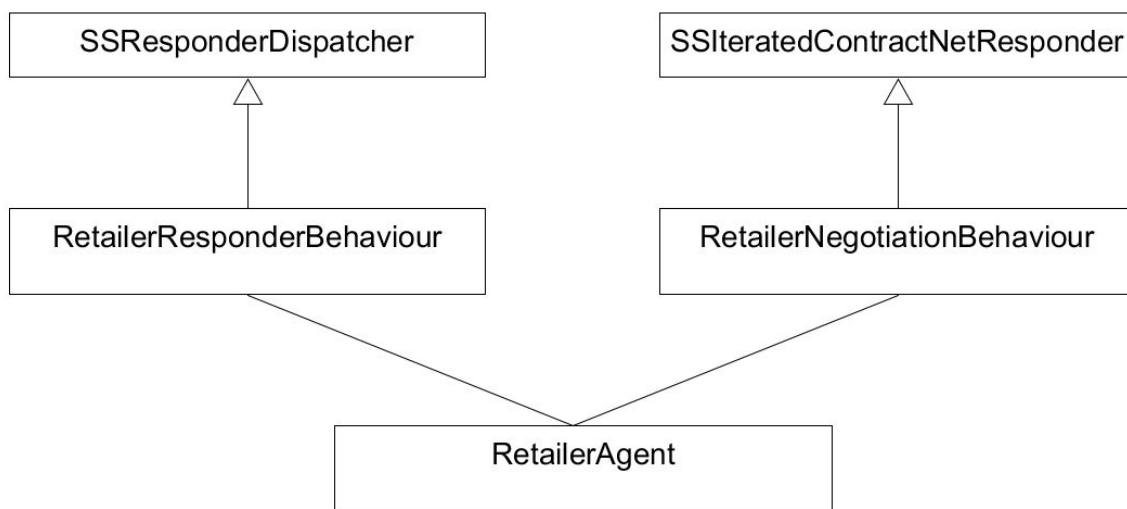


Figure 3: Retailer Agent Structure Diagram

Agent communication

Agents, by nature, do not depend on each other. However, usually they can only reach their goals by communicating with one another. The Home Energy Trading System is no exception because the system only works when the different agents communicate and work together. Therefore, to enable communication between the different agents, we leveraged the Directory Facilitator which comes as part of the JADE framework.

Using the Directory Facilitator, the Home agent registers itself as a service on startup so that other agents can look it up. After that, when appliance agents or retailer agents get created, they search for the Home agent using the Directory facilitator. Then they register themselves with the Home agent by sending a register message. They also unregister themselves from the Home agent in case they get shut down again. Because both the appliance agents and the retailer agents share the same registering behaviour, we implemented a common super class for them called **RegisteringAgent**. This class takes care of registering the appliance or the retailer, respectively, with the home agent.

We chose this approach of letting the home agent register itself as a service because it is the mediator between appliances and retailers and it also initiates the interactions with them. For that reason, it makes sense that the other agents register to it. Also, as the simulation system will only ever have one home agent, it appeared to be most intuitive. Last but not least this also made the implementation much easier, because the Home agent is the first agent that gets started in the system.

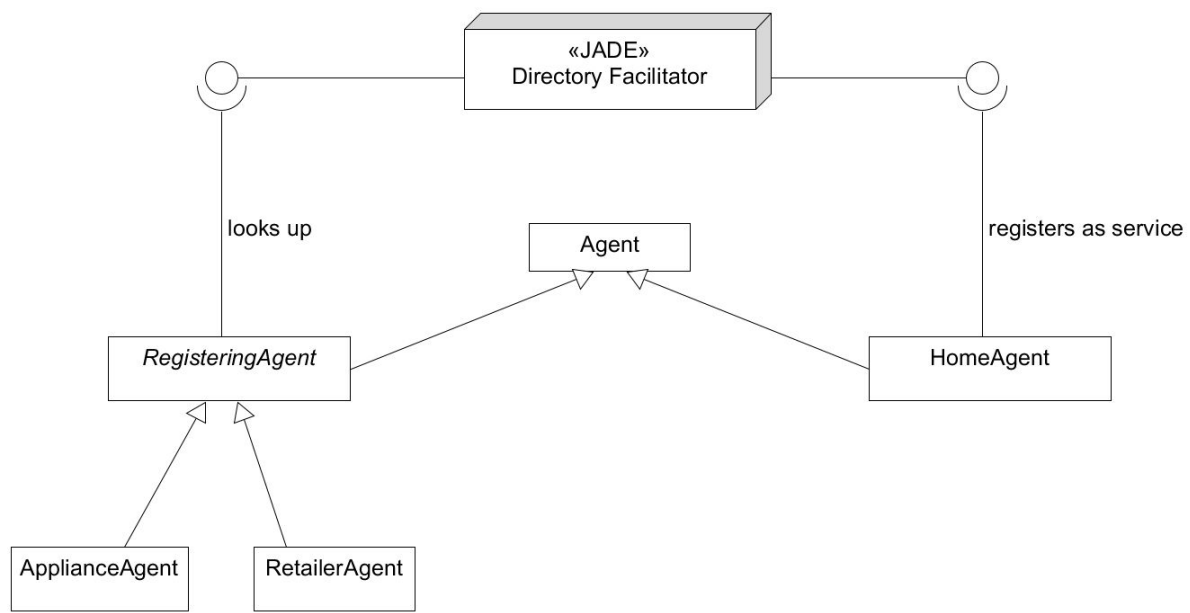


Figure 4: Agent registration and lookup using the Directory Facilitator

2.2 Prediction Package

The prediction package contains all the classes for the functionality of all neural network prediction algorithms as well as code to read the CSV files with all the relevant reading and weather data that will be needed during both training and simulations.

Neural Network Manager

The Neural Network Manager functions as the bridge between the JADE interface and the neural network - handling any input from the main system that should be accounted for as well as data reading. All data files are read and split into training data and “actual” data, with the majority of it being used for training. The training data is used only when the neural network is run to train while the actual data is what is used during simulations. Each Appliance Agent has its own manager, and stores power readings relevant only to itself.

Currently, each time a new agent is created its Neural Network Manager will tell the neural network to train given a certain set of parameters. These are the number of hidden layers, the number of neurons in each layer and the number of training sessions.

Upon requiring a prediction, this manager will be called to run the neural network given the current time in the system.

Neural Network

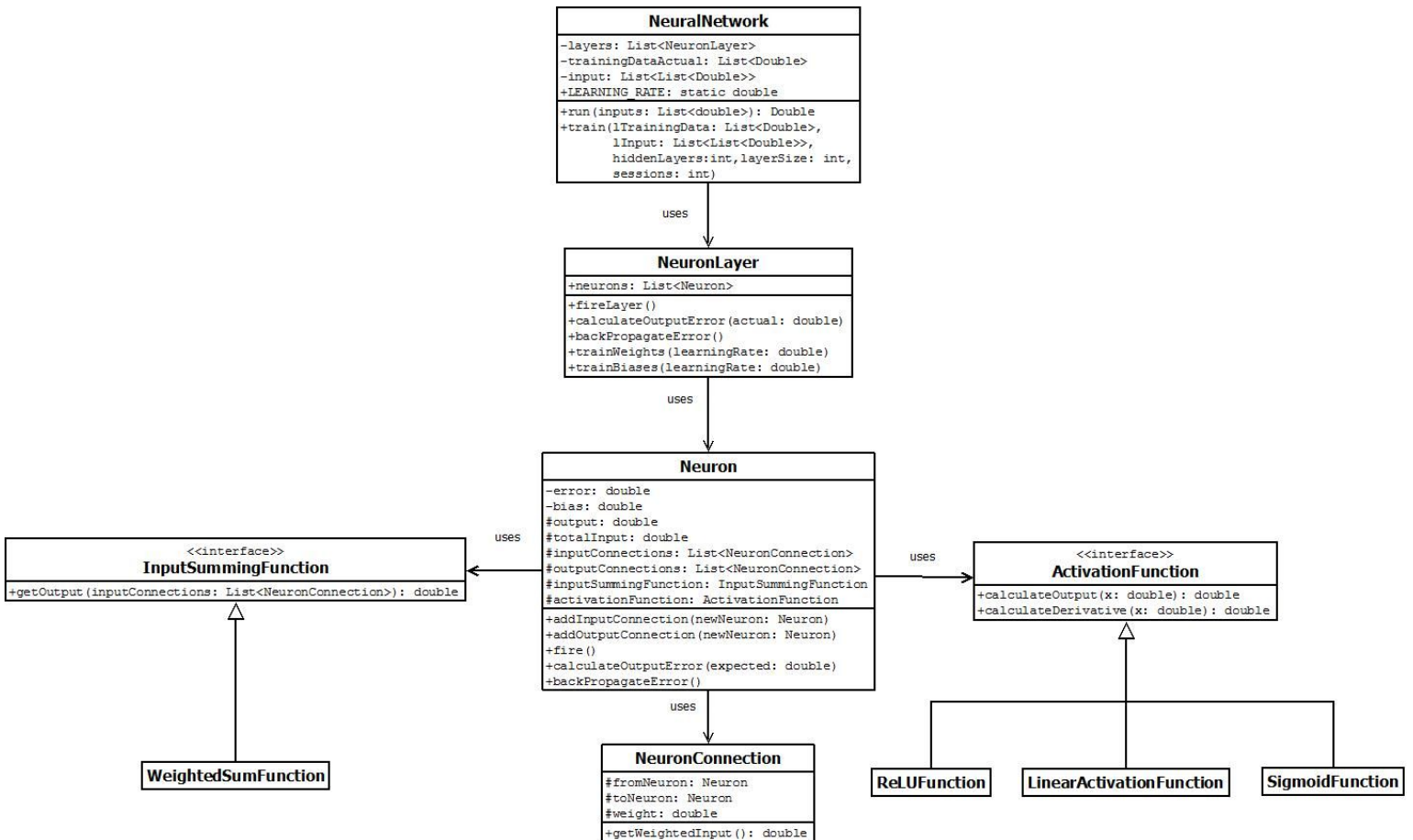


Figure 5: Neural Network UML Class Diagram

The neural network itself is structured as shown in Figure 5. The neural network contains multiple neuron layers, which then contain neurons that are connected together. Each neuron also has an activation and summing function in order to operate correctly. More detail on the functionality of each structure will be given in section 5.

2.3 Integration of Forecasts/Predictions into Appliances

In order to make the integration of forecasting methods as flexible as possible we created a simple interface **UsageForecast** which simply defines one method **calculateForecast** that the appliance can call to get forecasts for a specific period. That way, the appliance does not depend on the actual implementation of the **UsageForecast** and the forecast is easily

interchangeable. We implemented three different Usage Forecasting methods into the system: Simple, Moderate and Complex.

Simple

The Simple Forecasting method is the most basic forecasting method we implemented. It just uses the last actual usage of the appliance as the forecast for the next period. This has been implemented using the **SimpleUsageForecast** class.

Moderate

The Moderate Forecasting method uses the average of all the past actual usages as the forecast for the next period. The problem with this method is that it does not take the usage variations during a day into account. Instead, it stays relatively stable throughout the whole a day and becomes even more stable the more the time progresses. This method has been implemented using the **ModerateUsageForecast** class.

Complex

The Complex Forecasting method is much more advanced than the other two methods. It takes into account various factors like the time, temperature and more in order to accurately predict the usage for the next time period. It does so by using a Neural Network that has been trained using real world data of different appliances. This method has been implemented by the **NeuralNetworkForecast** class. This class uses the class **NeuralNetworkManager** from the prediction package to get the forecast for the next period. So, other than the Simple and the Moderate Forecasting methods, this method does not rely on the past actual usages, but on a Neural Network that has been trained using real world data.

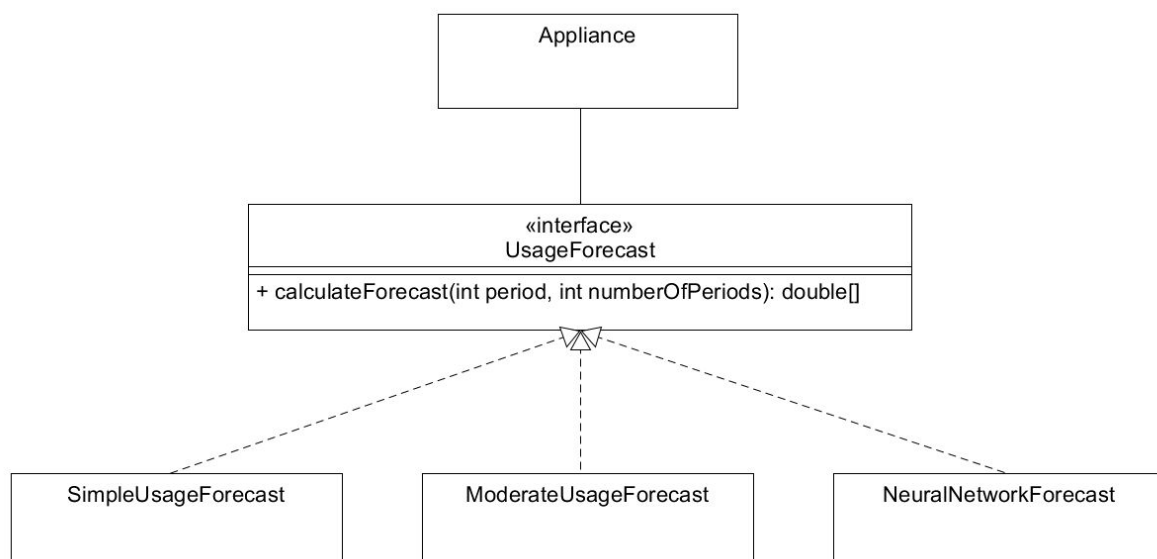


Figure 6: Appliance forecasting class structure

2.4 UI Package

GUI of the application is developed by the Java Swing Framework which functions as an interface that allow user to interact and control the system. It outputs all the information necessary to monitor the system and how it performs. This uses a MVC (Model, View Controller) software design pattern which separate the data and logic performed from UI components and controllers.

UI layout consists of nested panels act as containers and buttons for user interactions and to navigate around the system. It contains three main Sections which contains the view of the system as Dashboard Panel and two more displaying added Appliances and Retailers. Each contains the ui components necessary for interactions and functionality of the system, also all of the button clicks are handled by listeners which maintained by controllers of each class. Dialog windows are used to get user input in adding agents and setting configuration parameters for forecast and running configurations.

UI is separated with its controllers and logic, by having all interactions accessed via interfaces handled by JadeController class. Each interactions are saving data and information which used by to above described logic and prediction packages and then updates the view with relevant information that changes dynamically. Main Dashboard panel gives user critical information on system status and how predictions happening using forecasted data using graphs and an information panel.

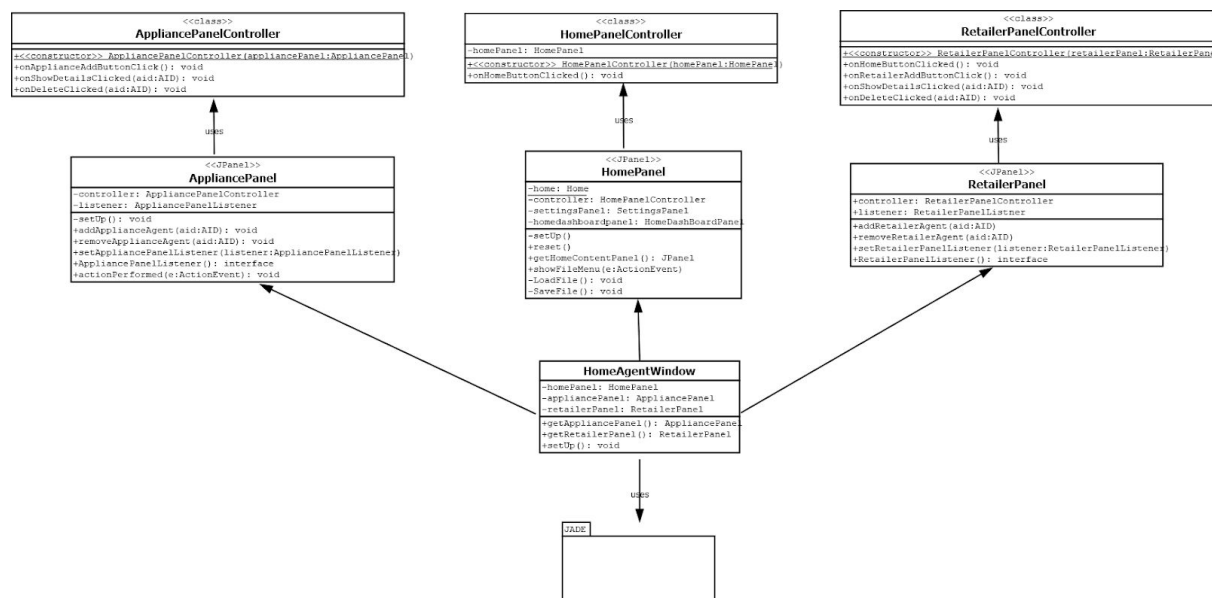


Figure 7: UI UML Class Diagram

GUI communication with JADE

The GUI communicates with the JADE agents using the `AgentContainer` and `AgentController` wrapper classes of the JADE Framework. To simplify the communication, a centralized class called `JADEController` has been developed that acts as a facade for all communications and interactions between the GUI and the agents. The `JADEController` also implements the Singleton design pattern so that it is easily accessible from anywhere in the program and ensures that there can only be one instance of the class.

In order to communicate with the agents from the GUI, we are using the O2A feature of JADE. This feature allows to map an Interface to an Agent. After that, you can retrieve the Interface using the **`AgentController`** class. Appliances, Home and Retailers have been defined using interfaces that contain all the necessary methods and functions to control and interact with them. The actual JADE agents then implement their respective interface. The GUI then only uses the interfaces to control the behaviour of the agents. That way, the JADE framework specific code is decoupled from the GUI code.

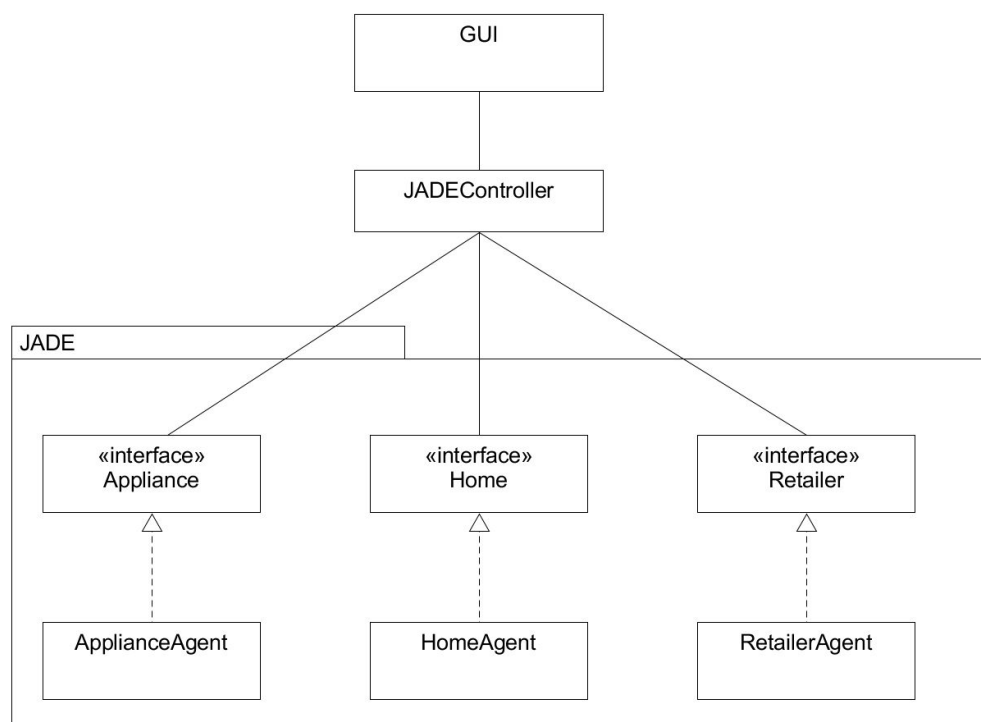


Figure 8: JADE/GUI interaction architecture class diagram

3. Implemented Interaction Protocols

3.1 FIPA Request Interaction Protocol

The FIPA Request Interaction Protocol has been used for two different use cases: the Agent Registration as well as the communication between the Appliance Agents and the Home Agent.

Agent Registration

Both the Appliance Agents and the Retailer Agents register themselves with the Home Agent when they get started. Because they both share the same registration process, a common superclass **RegisteringAgent** has been created from which they inherit. That way, the RegisteringAgent superclass then takes care of the registration process for them automatically. The RegisteringAgent adds the **HomeAgentRegisterBehaviour** to the agent on startup. The HomeAgentRegisterBehaviour is an implementation of the **AchieveREInitiator** class and therefore acts as the initiator of the Request Interaction Protocol.

The counterpart to this is the **HomeAgentRegisterRespondBehaviour** which inherits from the **AchieveREResponder** class. The Home Agent uses this behaviour to respond to the registration requests.

Using these two behaviours, the registration works as following: The registering agent sends a registration message to the home agent with the “Request” performative.

If it is registering an appliance agent it sends “REGISTER APPLIANCE” as the message content and if it is registering a retailer agent it sends “REGISTER RETAILER” respectively.

The HomeAgentRegisterRespondBehaviour then sends an “Agree” message back if the message content is equal to one of the two messages above. After that, it registers the appliance or retailer agent internally with the Home Agent. Once this is done, it sends an “Inform” message back to the registering agent. That way the registering agent knows that it is registered by the home agent know and this finalizes the registration process.

Communication between Appliance Agents and the Home Agent

The energy usage and forecast communication between the Appliance agents and the Home agent also uses the FIPA Request interaction protocol. The Home agent periodically requests usage information from its appliances, such as actual usage and a usage forecast.

Therefore, the Appliance Agent has an **ApplianceResponderBehaviour** to respond to requests. As counterpart, the Home Agent has two request behaviours: The

ApplianceUsageRequestBehaviour and the **ApplianceForecastRequestBehaviour**.

The home agent uses these behaviours periodically to get the actual and forecast usages from the appliances for the corresponding period. To do that, the two behaviours send

“Request” messages to all the appliances of the home agent. The content of the message is “ACTUAL_USAGE x” or “FORECAST x”, respectively, where x corresponds to the period. The ApplianceResponderBehaviour then agrees to deliver that information by responding with an “Agree” message. After that, it sends an “Inform” message containing the actual usage or the usage forecast, respectively.

3.2 FIPA Iterated Contract Net Interaction Protocol

The negotiation between the home agent and the different retailer agents uses the FIPA Iterated Contract Net Interaction Protocol. This protocol allows for multiple iterations of CFPs (Call for proposals) followed by propositions by the different retailer agents. This enables a flexible multi-round negotiation between the home agent and the retailer agents.

In order to implement the Iterated Contract Net Interaction Protocol, JADE provides three classes.

1. ContractNetInitiator
2. SSResponderDispatcher
3. SSIteratedContractNetResponder

The Home Agent uses the **HomeAgentNegotiationBehaviour** which inherits from the **ContractNetInitiator** in order to initiate the negotiation with the retailer every period.

To respond to these requests, the Retailer Agents then have two behaviours: The **RetailerResponderBehaviour** inherits from the **SSResponderDispatcher** and accepts the negotiation requests from home agent. This behaviour creates a new **RetailerNegotiationBehaviour** for every incoming negotiation request. The **RetailerNegotiationBehaviour** inherits from the **SSIteratedContractNetResponder** and basically represents a single negotiation session between the home agent and the retailer agent using the FIPA Iterated Contract Net Interaction Protocol.


4. Implemented Negotiation Strategies

Using the forecasts from the neural network, the home agent then aggregates its total predicted usage and assuming extra energy is required, attempts to buy this energy from retailers. Each retailer uses a different pricing mechanism to determine the volume charges as well as the feed-in rates. Each time step (or as required) the home agent will attempt to negotiate with each retailer to get their best price before buying the energy. It is also able to sell energy to retailers in case the Home produced more energy than it consumed. For that reason, the Home agent has two separate negotiation strategies: a buying strategy and a selling strategy. Consequently, the retailer agents also have a separate selling and buying strategy.

4.1 Retailer Tariffs

There are three different potential tariffs that retailer agents may use. These must be set when creating a new retailer agent. The retailer uses the tariffs to calculate the initial value for its negotiation strategy based on the current period and requested amount of energy. It uses the volume charge for its selling strategy and the feed-in rate for its buying strategy respectively.

Random Tariff



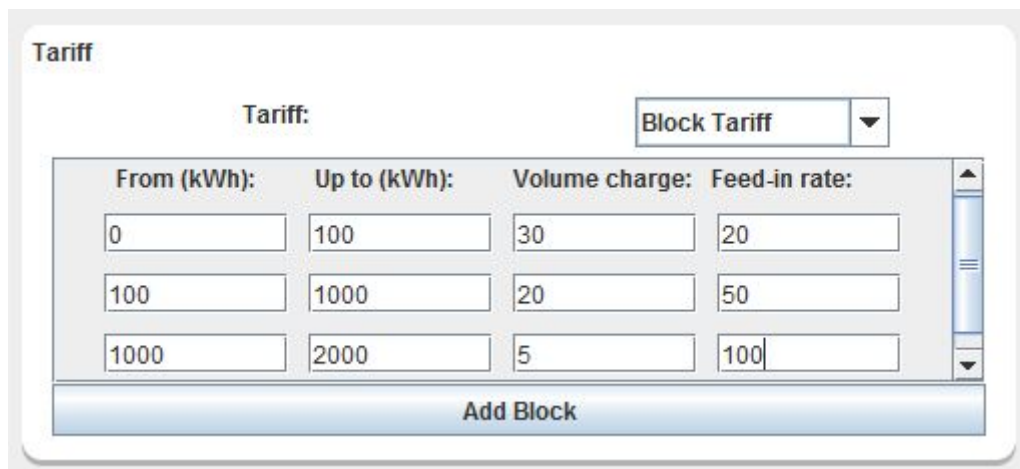
The interface for the Random Tariff shows a dropdown menu labeled 'Tariff:' with 'Random Tariff' selected. Below this are four input fields:

- Minimum Volume Charge Price: []
- Maximum Volume Charge Price: []
- Minimum Feed-in Price: []
- Maximum Feed-in Price: []

Figure 9: Random tariff interface where min/max price values can be set

The Random Tariff setting generates a random value between the maximum and minimum prices for energy bought as well as feed-in prices for extra energy generated by the solar panel.

Block Tariff



The interface for the Block Tariff shows a dropdown menu labeled 'Tariff:' with 'Block Tariff' selected. Below this is a table with four columns: 'From (kWh):', 'Up to (kWh):', 'Volume charge:', and 'Feed-in rate:'. The table contains three rows of data:

From (kWh):	Up to (kWh):	Volume charge:	Feed-in rate:
0	100	30	20
100	1000	20	50
1000	2000	5	100

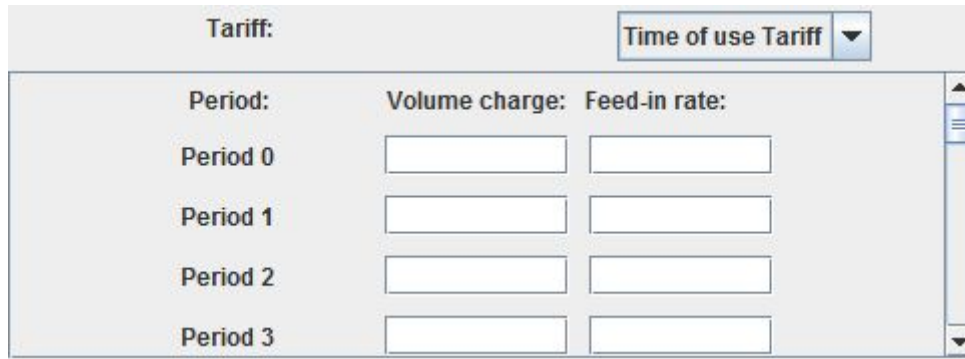
At the bottom of the table is a button labeled 'Add Block'.

Figure 10: Block tariff interface where new blocks can be added

The Block tariff setting allows the retailer agent to sell energy based on the amount required in set blocks. Typically, buying large amounts of energy will result in some sort of discount while selling energy will result in a reduced amount per watt. Any energy requirement that

is outside the range of one of the blocks results in the retailer agent stepping away from negotiations. For example, a retailer agent who sells very small amounts of energy up to 100W and then large amounts of energy from 1000W upwards will not attempt to sell the home agent 500W at all.

Time of Use Tariff



Period:	Volume charge:	Feed-in rate:
Period 0	<input type="text"/>	<input type="text"/>
Period 1	<input type="text"/>	<input type="text"/>
Period 2	<input type="text"/>	<input type="text"/>
Period 3	<input type="text"/>	<input type="text"/>

Figure 11: TOU tariff interface showing where volume and feed-in rates can be set per hour

The Time of Use pricing mechanism adjusts both the volume charge and feed-in rate for each hour. These must be set individually as shown in Figure 11. Ideally, there should be a certain price range for peak (when the most energy is used on average), off-peak (the least amount of energy is used) and shoulder (in between) time ranges; behaving differently on weekdays versus weekends. While there is no settings available to automate this, it is sufficient.

4.2 Negotiation Strategies

Negotiation strategies are implemented by both the retailer and home agents to maximise their own cost. Similar to the tariffs, the negotiation strategy may be also be set through the UI when creating a new retailer agent or through the main settings menu for the home agent. There are x different options available explained in detail as follows.

Fixed Price

The fixed price strategy is available for both the retailer and home agent. Each round it will return the same price based on its tariff setting and does not negotiate up or down. This agent also has no deadline and therefore in the case of the retail agent, also acts as a backup in the case that a deal is unable to be made with any of the other agents. That is, the home agent will always have an energy provider, even if it is suboptimal. In the case of the home agent, it will not budge from its desired price and may cause negotiations to break down. However, there will always be a fixed price retailer agent present to ensure that there is no situation in which energy cannot be provided to the home.

Time-Dependent

This strategy is available to both the retailer and the home agents. As the negotiation progresses, the behaviour of the agents and the offers they give or are willing to accept will change.

Negotiation Strategy:	Time-Dependent ▼
Deadline (Rounds):	<input type="text"/>
Reservation Value:	<input type="text"/>
Beta Value:	<input type="text"/>

Figure 12: Time-Dependant negotiation strategy UI

During setup, the number of maximum negotiation rounds (deadline) can be set for both the home and retailer agents. The behaviour of the agent over time depends on two factors - the reservation value and the beta value. The reservation value is the least favourable price for both agents respectively and they will not concede past this point.

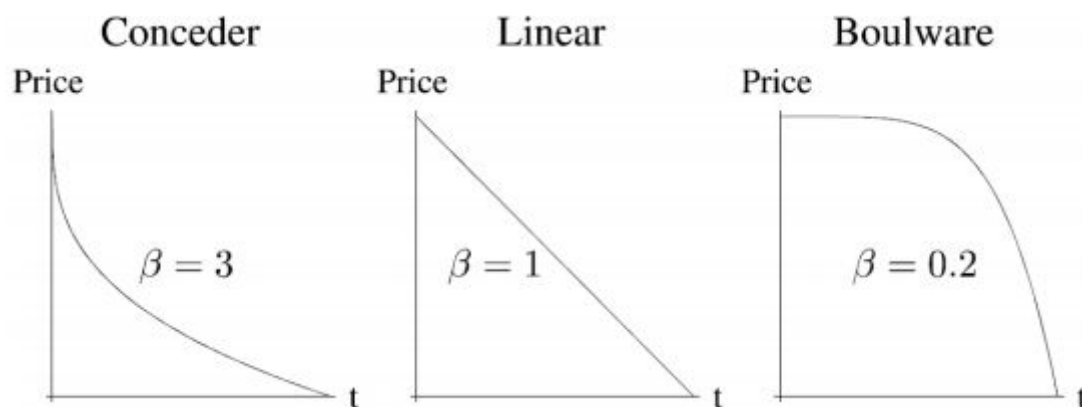


Figure 13: Conceder, Linear and Boulware price-time behaviours. (Source: Week 8 lecture slides)

The beta value determines how the system will behave over time. $\beta > 1$ classifies the agent as a “conceder” - the offered price will quickly approach the reservation value over the first few rounds. In real life, this type of agent would be used to prioritise selling the resource over maximising profit per unit. $\beta < 1$ gives the agent a Boulware behaviour, where the agent will stick very close to its original price for the majority of the negotiation, only dropping towards the end when there may be a risk of not selling at all. Finally, $\beta = 1$ makes the system linear, which as it implies means the negotiation price decreases at the same rate each round.

Random Tit-for-Tat

Last but not least we also implemented the Random Tit-for-Tat negotiation strategy. This strategy exactly models the behaviour of the opponent. In addition to that, it adds some variation by adding or subtracting random values.

4. 3 The Actual Negotiation

An actual negotiation between the Home Agent and a Retailer Agent is done using the **Negotiation** class. For that, this class uses the agent's strategy as well as an instance of the **CardinalUtility** class to compare the utility of the offers.

Both the Home Agent and the Retailer Agent have their own private instance of the **Negotiation** class. The **Negotiation** class essentially represents a single negotiation between the Home Agent and a Retailer and it is able to generate counter offers based on the incoming offers.

In case the deadline or the reservation value of the used strategy have been exceeded, it returns a "Refuse" offer which in turn terminates the negotiation. If the incoming offer is better than the last outgoing offer, the negotiation returns an "Accept" Offer and the negotiation was successful. In order to determine if the incoming offer is better or worse than the last outgoing offer, it uses the **CardinalUtility** class. This class allows the mapping of the initial value as well as the reservation value of the strategy into a utility value in the range between 0 and 1. That way it is possible simple to compare the offers, even though the two negotiating agents might have different initial values or reservation values.

5. Implemented Prediction Algorithms

5.1 Overall Design

The method of prediction used in this system is a neural network as previously mentioned during the overall system architecture. Here, the structure and approach will be expanded on.

This neural network was implemented from scratch and can be considered to be potentially the most basic type of neural network. Each Neural Network object contains a number of Neuron Layers. The input and output layers are created by default, while additional hidden layers can be added in the constructor. The input layer contains neurons equal to the number of inputs. For example, if the network is set to be trained using time, temperature and day of the week, there will be three neurons in the input layer. The output layer always contains only one neuron but could easily be changed to contain as many as one wants,

however, the home energy trading system only requires a single forecast value. The number of neurons in hidden layer may also be set when creating the network.

Each neuron layer contains a list of neurons that have input and output connections. These connections are automatically created and linked at the initialisation, so that each neuron in the preceding layer is connected to each neuron in the next via a Neuron Connection object. Each connection contains a weight value and each neuron contain a bias.

When forward propagating, or running the network as usual, all inputs from layer n will be multiplied by the weights in their output connection and then summed inside the corresponding neuron in layer $n + 1$. The total input to a single neuron is calculated via the WeightedSumFunction, an implementation of the InputSummingFunction interface. An interface has been used for the weight function as well as the activation function in order to make it easy to swap them.

5.2 Activation Function

The output of each neuron is then calculated by passing the input into the activation function and then adding on the bias. There are three different implementation of the activation function: Sigmoid (logistic), linear and ReLU (rectified linear unit).

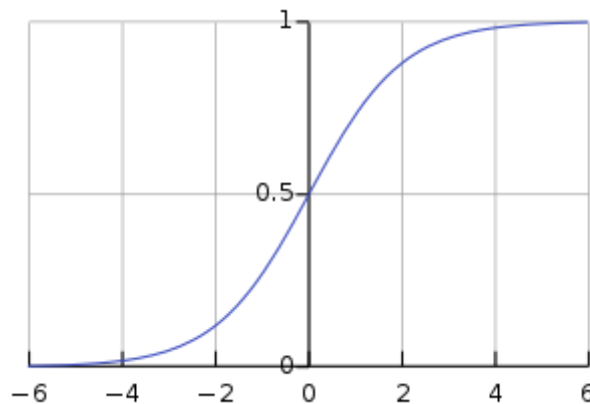


Figure 14: Sigmoid activation function, defined as $f(x) = \frac{1}{1+e^{-x}}$

The sigmoid function “squeezes” any input into a value between 0 and 1. While this is appropriate for a classification problem where each output neurons activation may respond to a different result, using the sigmoid function for this problem simply causes the weights to be too small and put too much pressure on the biases to give an accurate result.

While the linear function also had the problem of making negative outputs possible, the ReLU function fixed this to give reasonable (or at least not completely invalid) values for both the weights and biases.

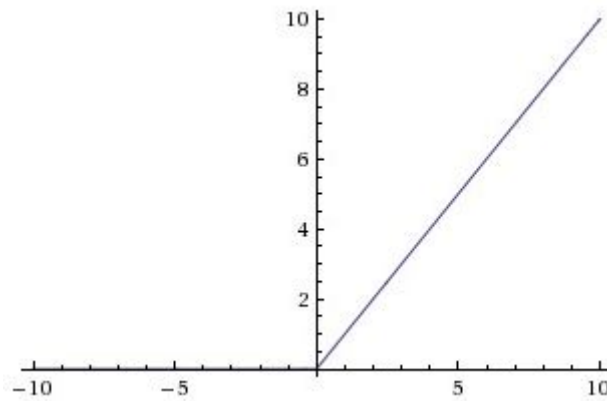


Figure 15: ReLU activation function

The ReLU function is defined as the linear function for all positive x , while negative values returns 0. There are other variants where negative values may be considered on a very small gradient, such as $f(x) = 0.01x^-$ however these were deemed unnecessary for the solution.

5.3 Training

The neural network may be trained for a specific for any number of sessions, as the user wishes. Each session will train the network using the given data but keeping the weight and bias values from past training.

In training the network, the algorithm is first forward propagated with random weights and biases. The output is then compared with the real data output of that time period and the error calculated. This error is then propagated through the network so that each neuron contains an error value. The error is then used to adjust all weight and biases in the network, proportional to a learning rate which is hard coded as 0.00001. This is repeated for each value in the training data until all of it has been processed.

6. Scenarios and Examples of the System Working

Examples of the system working can be found in various previously set up scenarios that are present in the configuration folder and loaded from the main menu. Following are some scenarios,

- 1) Fixed Price Retailer Buying
- 2) Fixed Price Retailer Selling
- 3) Time Dependent Retailer
- 4) Time of use Tariff
- 5) Tit for tat Retailer Selling

7. Critical Analysis of Performance With and Without Negotiation

The home agent, as well as each individual retailer agent, can be set to have a different negotiation strategy. The agents will optimally meet somewhere in the middle, at the point where they can both maximise their costs.

7.1 No Home Agent Negotiation

If the home agent has no negotiation strategy and uses the fixed price method, each round of negotiation with a retailer agent will result in the same offer from the home agent. This means there is potential for no optimal deal to be reached by the home agent and retail agents, were the reservation value to be higher than the agent is willing to pay. If this is the case, the home agent will be forced to accept the offer of the “default” fixed price retailer agent, as not supplying energy to the home is not an option. This is obviously suboptimal for the home agent since they may also potentially miss out on a good deal from another retail agent who has reduced his prices but whose negotiations were unsuccessful due to the deadline being exceeded or time being reached.

7.2 No Retailer Agent Negotiation

If there is absolutely no retailer agent negotiation present in the system the home agent will opt for the cheapest fixed price agent. Once again, since not buying energy isn't an option, the home agent will be forced into a corner by the retailers and have to buy at potentially a high fixed price. In the case that both the retailer and home agents are unwilling to negotiation up or down, the home agent will be forced to buy energy from whatever is available.

7.3 Ideal Negotiation Scenario

If both the home agent and at least some of the retailer agents are willing to negotiation their prices, the home agent will get deals from different agent at different points in time. For example, a time dependant retailer may offer a very reasonable off-peak rate but one that is very expensive during peak, potentially compensated by block agent that offers a cheap price during peak hour because a large amount of energy is being bought at once.

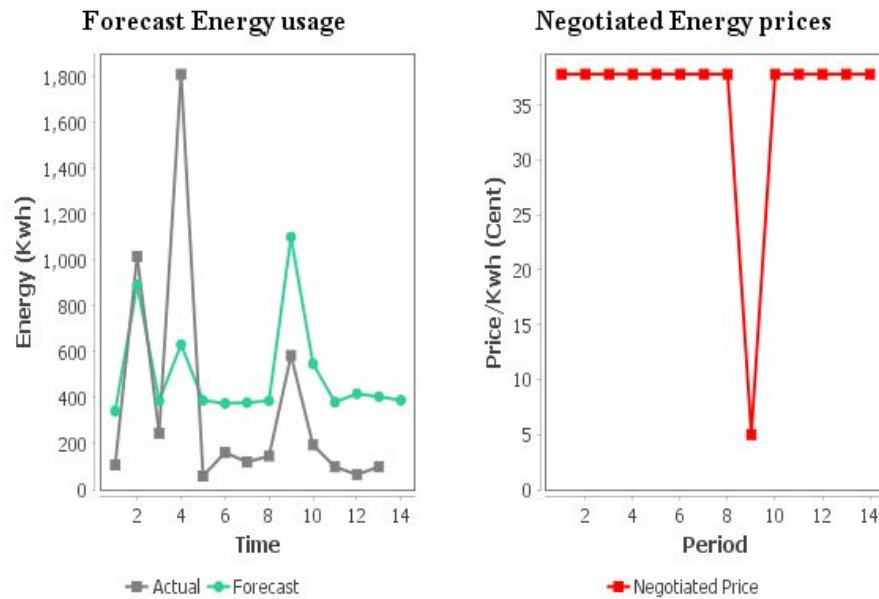


Figure 16: Forecasted energy usage and negotiated energy price graphs

Figure 16 demonstrates a scenario in which there are time-dependant conceder and bouldware retailer agents alongside a fixed price block agent. The block agent will provide energy cheaper than the others, but only if a large amount is needed. It can be seen that in time period 9 that a large amount of energy is predicted to be used and therefore a cheaper price can be gotten for it as a whole, despite all other time periods negotiating successfully with the conceder agent. Not that the negotiation is only based off of the predicted value and had the forecast in period 4 accurately predicted the home agent would potentially have been able to save a lot of money.

8. Conclusion

In this project we created a Home Energy Trading System that is able to accurately forecast its energy demand and then negotiate with different retailers to either sell or buy energy. The goal of the system is to minimize the costs or maximize the profits, respectively, of the homes. For that reason, we explored and implemented various different possible mechanisms: Different forecasting methods, multiple negotiation strategies and different tariffs for the retailer energy pricing.

We created three different methods for the Usage Forecasts of the appliances to be able to compare them to each other and investigate with methods provide the best outcomes for the system. They differentiate in complexity and have a huge impact on the overall performance of the system. The most complex forecasting method even makes use of a Neural Network. Comparing these methods to each other you can clearly see that the system performs better using the complex forecasting method.

Then, in order to buy energy from retailers or sell energy to them, we implemented different negotiation strategies to see how the home agents performance changes by using different strategies. We implemented the Fixed Price, Time-Dependent as well as the Random Tit-for-Tat negotiation strategies. Here, too, we have noticed that the performance of the system heavily depends on used negotiation strategy.

All in all, the Home Energy Trading System is an extremely flexible system that can be used to test various real world scenarios and to explore different options in order to find the system that performs best. Also, the system can be controlled and monitored using a sophisticated and nice-looking GUI and it is even possible to load and save different configurations from or into XML files in order to easily test various scenarios repeatedly.


Appendix

1.1 Team Contribution

Home Energy Trading System	
Team Members	Julian Robert Malaka
Name: Julian David Schaefer Student No: 101985593	Report <ul style="list-style-type: none"> • Overall System Architecture - Logic Package; Integration of Forecasts into Appliances • Implemented Interaction Protocols • Implemented Negotiation Strategies • Negotiation Strategies - The Actual Negotiation • Conclusion System Design/Coding <ul style="list-style-type: none"> • Development of Mock-Ups for the system • JADE infrastructure implementation (agent communication, Directory Facilitator) • JADE interaction protocols (FIPA Request Interaction Protocol and FIPA Contract Net Protocol) • Simple and moderate forecasting methods • Negotiation between home agent and retailers for selling and buying (Fixed Price Strategy, Time-dependent Strategy, Random Tit-for-Tat Strategy) • Retailer tariffs (Random Tariff, Block Tariff, Time of use Tariff) • Automated simulation in the UI • XML import and export of scenarios into configuration files • Graphs in the UI • Git management • Creation of Executable .jar file
Name: Robert Bagnato Student No: 100585471	Report <ul style="list-style-type: none"> • Overall System Architecture - Prediction Package

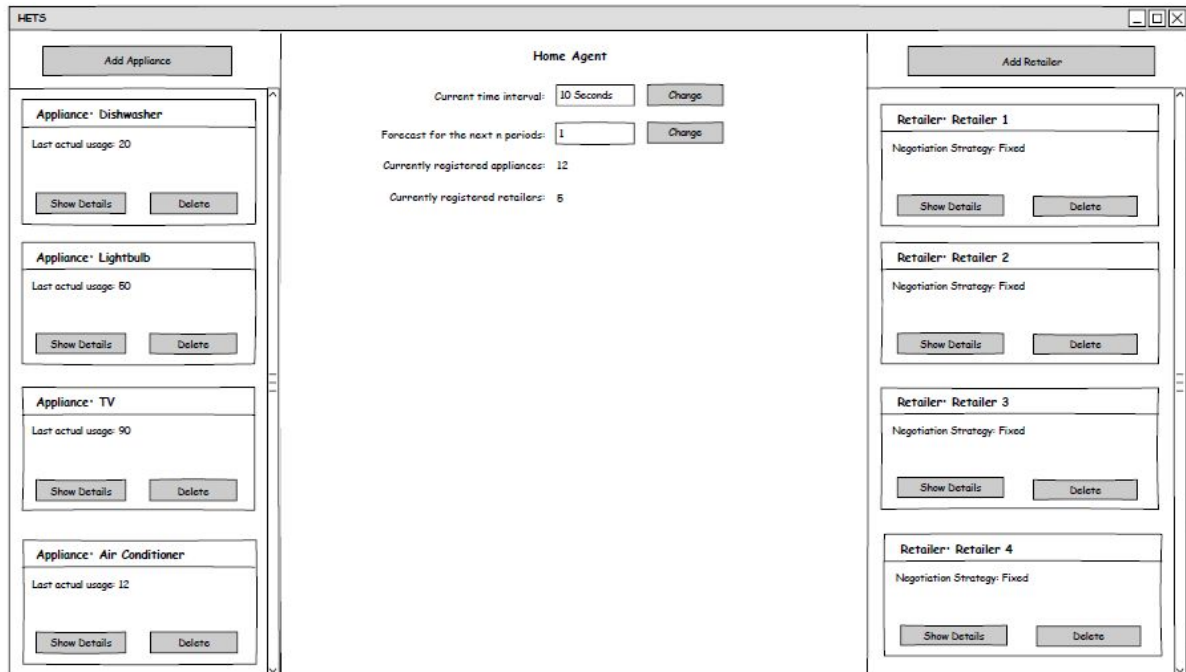
	<ul style="list-style-type: none"> • Implemented Prediction Algorithms (All) • Implemented Negotiation Strategies - Retailer Price Mechanisms (all), Negotiation Strategies (Fixed Price, Time-Dependant) • Report editing, formatting and proof-reading <p>System Design/Coding</p> <ul style="list-style-type: none"> • Prediction package/forecast design and implementation • Data reading and sorting
<p>Name: Malaka Perera Student no: 101065042</p>	<p>Report</p> <ul style="list-style-type: none"> • Overall System Architecture - UI Package • UML class Diagram <p>System Design/Coding</p> <ul style="list-style-type: none"> • Designing the GUI • Implement initial functionality of load/save config file • Implement UI for la settings panel for system • Implementing GUI components for the system • Develop custom JPanel, and button classes for UI

Team Member Name	Signature	Date
Julian David Schaefer	<i>J. Schaefer</i>	02/11/2018
Robert Bagnato	<i>R. Bagnato</i>	02/11/2018

Malaka Perera		02/11/2018
---------------	---	-------------------

1.2 UI Mockups

Main Page



The Main Page UI is a window titled "HETS" with a standard Windows-style title bar (minimize, maximize, close buttons). It is divided into three main vertical sections.

Left Section: Contains an "Add Appliance" button at the top. Below it is a scrollable list of appliances. Each entry shows the appliance name, its last actual usage, and buttons for "Show Details" and "Delete".

Appliance	Last actual usage
Dishwasher	20
Lightbulb	60
TV	90
Air Conditioner	12

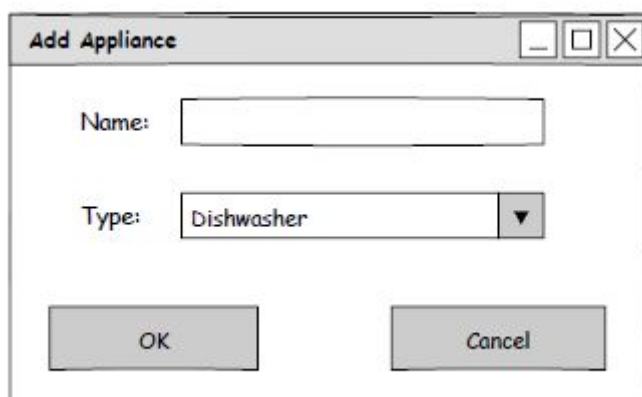
Center Section: Titled "Home Agent". It contains several status lines and interactive elements:

- Current time interval: 10 Seconds (with a "Change" button)
- Forecast for the next n periods: 1 (with a "Change" button)
- Currently registered appliances: 12
- Currently registered retailers: 6

Right Section: Contains an "Add Retailer" button at the top. Below it is a scrollable list of retailers. Each entry shows the retailer name, their negotiation strategy, and buttons for "Show Details" and "Delete".

Retailer	Negotiation Strategy
Retailer 1	Fixed
Retailer 2	Fixed
Retailer 3	Fixed
Retailer 4	Fixed

Add Appliance Window

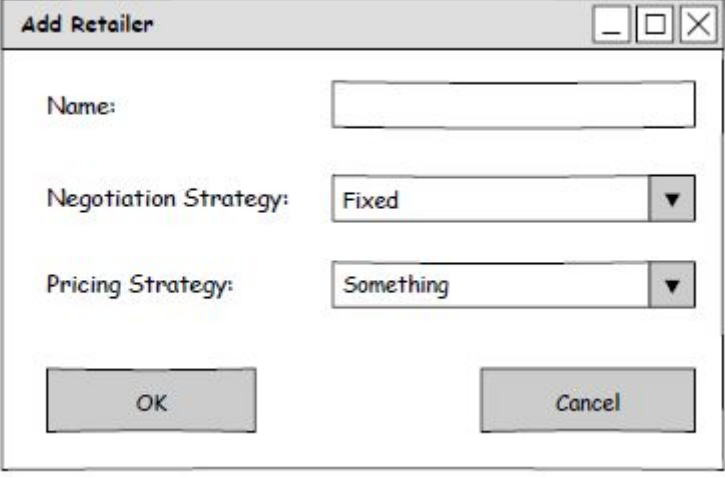


The Add Appliance Window is a small dialog box with a title bar "Add Appliance" and standard Windows-style title bar controls (minimize, maximize, close buttons).

It contains two input fields:

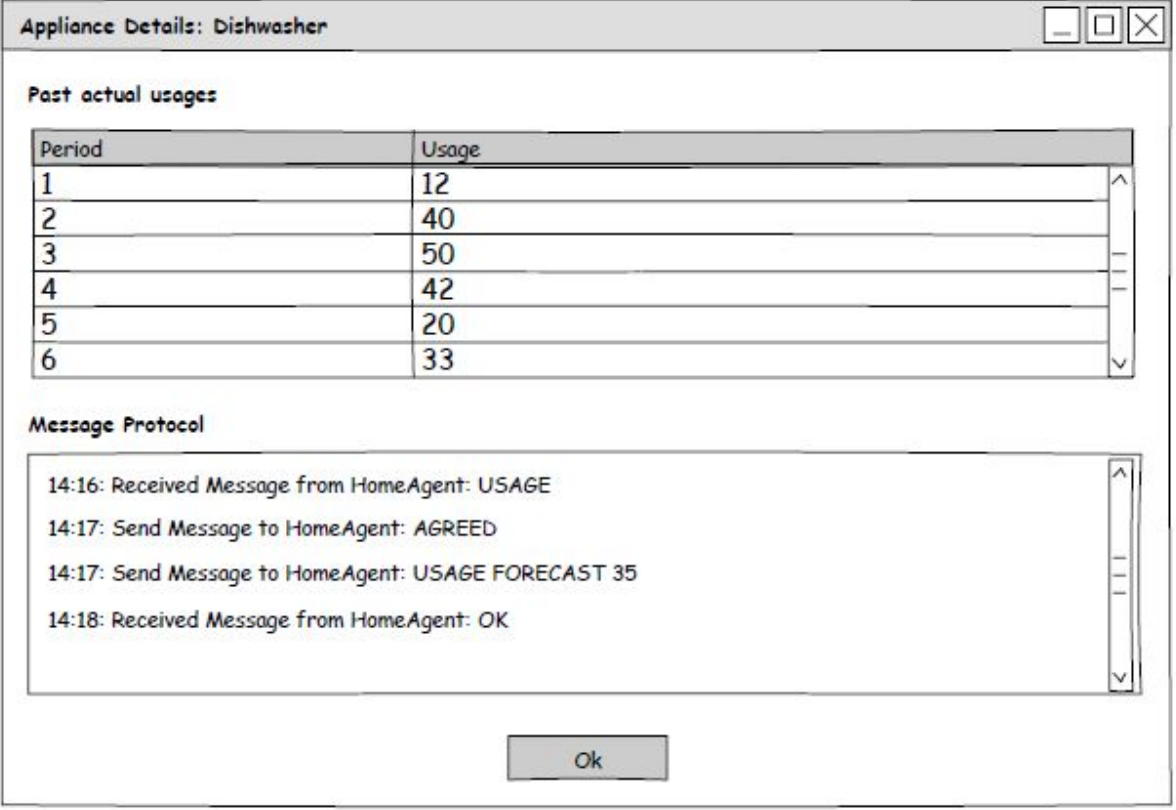
- Name:** A text input field.
- Type:** A dropdown menu currently showing "Dishwasher".

At the bottom, there are two buttons: "OK" and "Cancel".

Add Retailer Window:

A dialog box titled "Add Retailer" with a standard Windows window frame (minimize, maximize, close buttons). It contains three input fields: "Name:" with a text box, "Negotiation Strategy:" with a dropdown menu showing "Fixed", and "Pricing Strategy:" with a dropdown menu showing "Something". At the bottom are "OK" and "Cancel" buttons.

Name:	<input type="text"/>
Negotiation Strategy:	Fixed ▼
Pricing Strategy:	Something ▼
<input type="button" value="OK"/> <input type="button" value="Cancel"/>	

Appliance Details Window:

A window titled "Appliance Details: Dishwasher" with a standard Windows window frame. It displays two sections: "Past actual usages" and "Message Protocol".

Past actual usages

Period	Usage
1	12
2	40
3	50
4	42
5	20
6	33

Message Protocol

14:16: Received Message from HomeAgent: USAGE
14:17: Send Message to HomeAgent: AGREED
14:17: Send Message to HomeAgent: USAGE FORECAST 35
14:18: Received Message from HomeAgent: OK

Ok