Technical University of Crete

Diploma Thesis

Development of a Functional Multi-Agent System Prototype for Offering Integrated Vehicle-to-Grid and Grid-to-Vehicle Services in Smart Cities

Author:

Georgios Kechagias

Committee:

Associate Professor **Georgios Chalkiadakis** (Supervisor)

Associate Professor **Eftichios Koutroulis**

Dr. **Nikolaos Spanoudakis**

A thesis submitted in partial fulfillment of the requirements for the degree of Diploma in Engineering

in the

School of Electrical and Computer Engineering

Technical University of Crete

2018

Technical University of Crete

School of Electrical and Computer Engineering

Development of a Functional Multi-Agent System Prototype for Offering Integrated Vehicle-to-Grid and Grid-to-Vehicle Services in Smart Cities

by Georgios Kechagias

Abstract

In this work, we design a functional prototype for a novel multiagent systems (MAS) services architecture for the important and challenging to engineer vehicle-to-grid (V2G) and grid-to-vehicle (G2V) energy transfer problem. The prototype was developed using JADE, a FIPA (Foundation of Intelligent Physical Agents)-standards compliant multiagent platform. Agent communication is based on the exchange of appropriate FIPA-ACL agent communication language messages, and on well-defined communication protocols specifically tailored to our application domain. As part of our work, we defined two novel design patterns, which allow the developers *(i)* to reuse the protocol parts and logic defined in the framework, and *(ii)* to customize key agent functionalities or capabilities according to their needs/goals. We demonstrate the functionality and effectiveness of our system prototype on a variety of realistic use case scenarios, executed using both real-world and synthetic datasets.

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Ανάπτυξη ενός Λειτουργικού Αρχετύπου Πολυπρακτορικού Συστήματος Παροχής Ολοκληρωμένων Υπηρεσιών Φόρτισης-Αποφόρτισης Ηλεκτρικών Οχημάτων σε Έξυπνες Πόλεις

Γεώργιος Κεχαγιάς

Περίληψη (Abstract in Greek)

Η διπλωματική εργασία ασχολήθηκε με το σχεδιασμό και την υλοποίηση ενός λειτουργικού αρχετύπου για μια καινοτόμο πολυπρακτορική αρχιτεκτονική προσφοράς υπηρεσιών στο πεδίο της φόρτισης/εκφόρτισης ηλεκτρικών οχημάτων (στα αγγλικά: the vehicle-to-grid/grid-to-vehicle -V2G/G2V- problem). Το εξαιρετικά σημαντικό καινοφανές αυτό πραγματικό πρόβλημα, απαιτεί λόγω της φύσης του καθώς και της περιορισμένης σχετικής βιβλιογραφίας και εμπειρίας σχετικών λύσεων την προσεκτική σχεδίαση (πληροφοριακού ή άλλου) συστήματος εκ μέρους του μηχανικού. Το αρχέτυπο αναπτύχθηκε χρησιμοποιώντας το JADE, μια πολυπρακτορική πλατφόρμα που είναι συμβατή με τα πρότυπα του διεθνούς οργανισμού Foundation of Intelligent Physical Agents (FIPA). Η επικοινωνία των πρακτόρων βασίζεται στην ανταλλαγή κατάλληλων μηνυμάτων της FIPA-ACL γλώσσας επικοινωνίας πρακτόρων, και σε καλά καθορισμένα πρωτόκολλα επικοινωνίας ειδικά σχεδιασμένα για το πεδίο εφαρμογής. Ως μέρος της εργασιας μας, προχωρήσαμε στον ορισμό δύο πρωτότυπων σχεδιαστικών προτύπων, τα οποία επιτρέπουν στους προγραμματιστές *(i)* να επαναχρησιμοποιούν το κομμάτια των πρωτοκόλλων και την λογική που ορίζεται στο σχεδιαστικό πλαίσιο, και *(ii)* να προσαρμόζουν βασικές λειτουργίες και ικανότητες των πρακτόρων σύμφωνα με τις ανάγκες και τους στόχους τους. Η λειτουργικότητα και αποδοτικότητα του αρχετύπου μας επιδεικνύεται σε μια σειρά από ρεαλιστικά σενάρια χρήσης του συστήματος, τα οποία εκτελούνται χρησιμοποιώντας συλλογές από πραγματικά και συνθετικά δεδομένα.

Acknowledgements

First and foremost, i would like to thank my supervisor Dr.Georgios Chalkiadakis for teaching me multi-agent systems and algorithmic game theory all these years, for trusting me with this thesis, and for his help and guidance throughout my studies. I am also truly thankful to my co-supervisor Dr.Nikolaos Spanoudakis for always being available to answer my questions, for teaching me how to engineer multi-agent systems, and for the enjoyable time we sent at the office. Moreover, i am grateful to Dr.Charilaos Akasiadis for his guidance and comments throughout this work, the moral support, and encouragement. I would like to thank my dear friend Vassilis Amourgianos for his invaluable help to the preparation of the final presentation of this thesis, and all my friends who in their own way helped me during my studies. I can not thank enough Lydia for all her love, patience and support all these years. Last but not least, i would like to thank my parents and sister; Maria-Christina, Paschalis, and Maria, for always loving and supporting me unconditionally.

Contents

List of Figures

List of Tables

Dedicated to my mother and father

Chapter 1

Introduction

To this day, electricity production has been considerably depended on fossil fuels such as coal, petroleum and natural gas. Many countries have initiated endeavors to reduce their carbon emissions through the reduction of fossil fuels usage. In this context, the usage of electricity in the transportation sector and the progressive independence from the internal combustion engine (ICE) which most contemporary vehicles utilize, is expected to be pivotal. In recent years, pure electric vehicles (EV)s have been gaining momentum and today more than 3 million circulate globally. China is the biggest EV market with more than half of global sales, 40% of the global stock, and 2.2% market share. Norway has the most successful EV deployment in terms of market share, with 39% [28]. Estimations from the International Energy Agency, indicate that the number of pure EVs on the road will reach 125 million by 2030.

Even though the increasing circulation of electric against ICO vehicles is going to reduce, to some extent, the carbon emissions, this fact by itself is not going to reduce our reliance on fossil fuels. EVs require a considerable amount of electricity to charge their batteries, therefore they are expected to increase the electricity demand. Renewable energy sources such as sunlight and wind, are alternative energy sources which do not pollute the environment and are essential in order to complement the benefits of EVs. However, their drawback is that they are highly volatile and intermittent, thus, we need to design special mechanisms that will manage their usage efficiently. EVs could potentially be an asset in the electricity grid by providing ancillary services.

Over the recent years, a number of Demand-Side Management approaches [2] have been proposed, aiming to create appropriate mechanisms which incentivize electricity consumers to shift part of their consumption to time intervals during the day where the renewable energy is plentiful. This is a central aspect of the smart grid agenda as the

direct managing of electricity generation in settings with high renewable energy penetration, is infeasible. EVs are parked on average 90% of the day [36], thus, they could be utilized as a temporary distributed energy storage system. During the time intervals with plentiful renewable energy production, EVs are enabled to charge their batteries, whereas, when the electricity grid presents energy shortages, the fully charged vehicles can provide energy back to the grid in order to cover peak demand (Vehicle-To-Grid Problem, V2G) and potentially create a profit for the EV owner [37, 49, 51].

At the same time, the uncontrolled charging of multiple EVs simultaneously, and the fact that EVs require a relatively large amount of energy to charge their batteries, put the electricity grid under critical strain. Therefore, the design and development of intelligent systems which are going to monitor and schedule the EV charging in real-time considering the grid constraints, such as the imbalance between the local electricity production and consumption prioritizing the utilization of renewable energy, is imperative (Grid-To-Vehicle Problem, G2V) [49, 63].

EVs charge through charging stations that are located either at the house of the driver or they are public. The latter case, given that EV charging preferences and EV's current state vary depending on the day, the month and the year, as well as the fact that some charging stations may be more congested than others, introduces a major difficulty to the EV drivers regarding the selection of the most suitable charging station. The extension of the intelligent systems mentioned before with charging station recommendation capabilities that are going to guide EVs to the most suitable charging stations is pivotal. This may reduce the amount of energy EVs consume searching for charging stations, providing the potential to a smoother incorporation of EVs to the electricity grid and increasing the satisfaction and engagement of EV drivers.

Multi-agent systems (MAS) are an excellent technological solution for smart grid applications such as EV charging management (V2G/G2V), as they enable the communication of complex information and the execution of difficult calculations effectively and in a distributed fashion [57].

1.1 Thesis Contributions

In the work presented in this thesis, we design a functional prototype of a novel multiagent systems (MAS) services architecture for the important and challenging to engineer vehicle-to-grid (V2G) and grid-to-vehicle (G2V) energy transfer problem. For the system design, we make an effort to keep it generic and open in order to be able to incorporate additional future requirements regarding the operation of the smart grid.

The key entities (stakeholders) of the proposed system are illustrated in Figure 1.1:

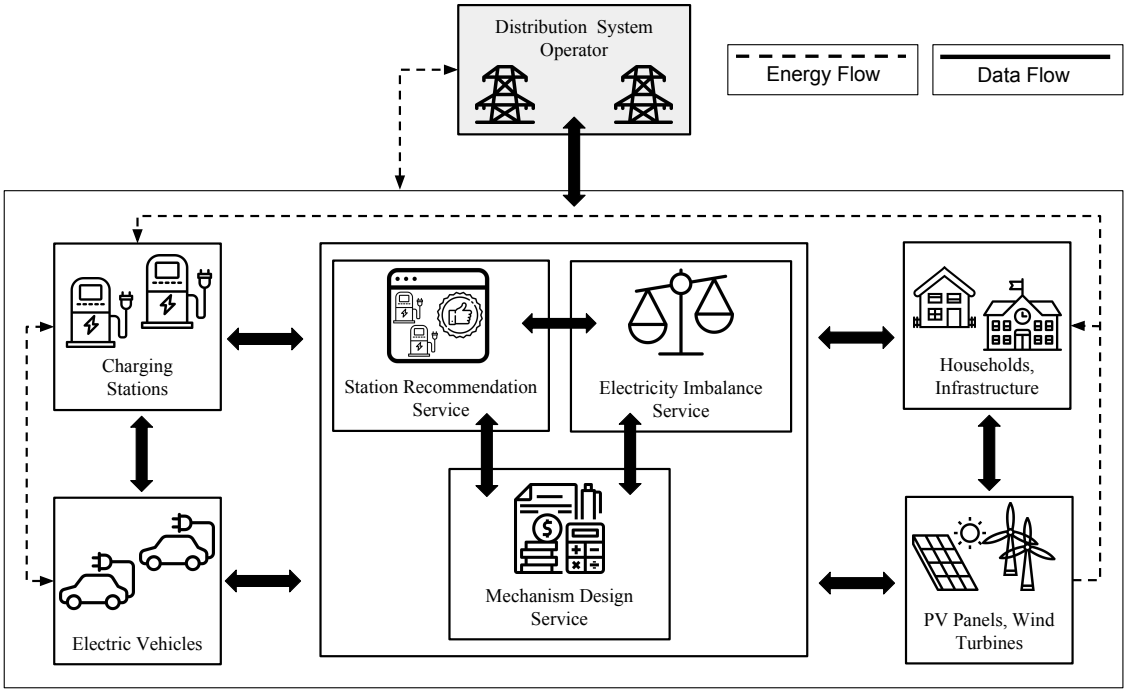

FIGURE 1.1: An overview of the core stakeholders of the proposed system.

- Electric Vehicles (EVs), which are a contemporary type of load in the electric grid whose drivers have very specific requirements and preferences regarding their battery charging.
- Charging Stations, which are the physical gateways where EVs connect to charge their batteries. Charging Stations make a profit from reselling electricity to EVs, while they can provide electricity back to the grid for the batteries of the fully charged EVs in case of an emergency.
- Households and Infrastructure, which incorporate the traditional energy consumers connected to the electricity grid. Contemporary houses can have the infrastructure for harvesting renewable energy which they can offer back to the grid when they have low energy consumption and high renewable energy production.
- Renewable Energy Producers, which are all the wind turbines, PV panels and other infrastructure (Distributed Energy Resource, DER) responsible for harvesting renewable energy and offering this energy for consumption to the grid. The owners of these infrastructures expect to make a profit from selling the produced energy to the electricity markets.
- Distribution System Operator, which is the physical and legal entity responsible for the safe operation, maintenance and development of the electricity distribution and to ensure the system's long-term ability to meet electricity demand.

Furthermore, we assume the existence of a national regulatory service entity for energy, or possibly a profit-making private service that consists of:

- Station Recommendation Service. which is a service designed to propose to EVs the charging stations that match perfectly to their needs, preferences and current mood.
- Electricity Imbalance Service, which is a service responsible for gathering from the various electricity producers and consumers their expected supply and demand for electricity and providing the interested stakeholders with the time intervals of energy excess and shortage.
- Mechanism Design Service, which is a service responsible to calculate and provide the price per unit of energy for each time interval, and to ensure that the reports made from the various stakeholders regarding their future operation in the electricity grid are truthful and accurate. The latter is mathematically guaranteed by controlling the flow of payments, taking into account in these payments the accuracy and truthfulness of the reports of the various entities.

All the stakeholders that were previously mentioned are represented as intelligent agents. This means these agents are not controlled in any way by a central authority, that they are enabled to communicate with other agents in their network in order to coordinate their actions, and that they are capable of independent decision making, in order to achieve their private goals.

The proposed system supports different agent types, i.e. EVs, charging stations, producers/consumers, and, in addition, specific V2G/G2V related services, such as recommendation systems, mechanism design schemes, and grid constraints extraction modules. Each of these agents is allowed to execute different algorithms to match their goals, e.g. achieve minimum charging cost, participate in V2G activities, avoid herding effects, etc.

For the inter-agent communication, we design and implement specific high-level stateful communication protocols, that enable diverse stakeholders to use the services of the agents that comprise our architecture. Furthermore, we develop a general V2G/G2V ontology that can be used in a plethora of smart grid related applications. The establishment of an ontology for the agent communication is pivotal, as agents should agree on the terminology they are going to use in their exchangeable messages. The definition of the communication protocols and domain ontology highlights the open nature of our MAS, where the various participants (agents) are generally able to freely join and leave the system at any time. The agents that we implemented, compose in their intra-agent control the various communication protocols according to their owns needs and goals.

As each agent can be diverse, with specific requirements, goals, and business models, thus, it is natural to need custom algorithms and logic. To this end, we define two new design patterns, that on the one hand allow the developers to re-use the protocol parts and logic defined in the framework, and on the other hand to customize key functionality or capabilities according to their needs/goals.

A fundamental feature of our system is that it enables software reusability. Each entity was designed and developed as an independent agent, by specifying open communication protocols and ontologies, and by applying strong software engineering practices (design patterns). As underlying framework of our implementation, we used the Java Agent Development Framework (JADE). JADE is a very popular, Java-based framework that facilitates the development of agent-based applications and it is compliant with the FIPA specifications for interoperable, intelligent, multi-agent systems.

The Graphical User Interface (GUI) that we developed from various agents, enables the human-agent interaction and makes the agent control, monitoring, and debugging easy and effective.

The system proposed in this thesis is an endeavor towards the consolidation of the theoretical results and algorithms proposed from various computer science fields for tackling V2G/G2V sub-problems in order to achieve efficient, smooth and reliable incorporation of the EV technology to the smart grid.

1.2 Thesis Structure

The rest of the thesis is structured as follows. Chapter 2 presents key concepts from the smart grid, electric vehicle charging, and algorithmic game theory. Chapter 3 introduces the concepts of agent and multi-agent system and then the methodology and tools that are commonly used to design and develop integrated agent-based systems. Chapter 4 present related works. Chapter 5 presents the architecture for the V2G/G2V problem and Chapter 6 contains the design and implementation of this architecture. Chapter 7, contains a detailed presentation of use case scenarios that demonstrate the functionality of the proposed system and the datasets we used for our evaluation. Finally, Chapter 8 concludes this thesis and outlines directions for future work.

Chapter 2

Smart Grid and Electric Vehicles

In this introductory chapter, we review some key concepts from the smart grid, electric vehicle charging and algorithmic game theory that are essential in order to capture the requirements and the assumptions that have to be considered for a realistic system that is going to offer integrated V2G/G2V services. We begin with an introduction to the smart grid, the energy markets, and the typical energy tariffs. Then, we discuss the characteristics of electric vehicles, the challenges arising from their charging and the benefits from their "smart" charging. Next, we introduce some important concepts of coalition formation and discuss how such techniques could assist the efficient incorporation of the electric vehicles into the smart grid. In this context we review basic recommendation system techniques which could assist EV drivers or their agents to explore alternative charging plans and finally, we introduce the mechanism design domain as a solution to incentivize truthful reporting of the preferences in smart grid settings.

2.1 Smart Grid Overview

Electric power is extremely important to modern society. Communities that lack electric power, even for short periods, jeopardize the public health, safety, and economic prosperity of their dwellers [66]. Electricity grids, which are the means of transmission and distribution of energy, were adequate for our old-fashioned usages, but as more sophisticated electricity consumption and production patterns arise, their management becomes more complex. It is evident that for their effective and reliable operation, electricity grids require considerable renovation [49].

Nowadays, we don't consume electricity generated only from bulk thermal power plants which burn fossil fuels, but part of the production is derived from decentralized energy

generators which use renewable energy such as the sunlight and wind and are incorporated to the distribution grid. Targets put forward by 196 countries under the *Paris Agreement* [45] for the reduction of greenhouse gases, indicate that the usage of renewable energy sources is going increase considerably in the near future. However, due to their dependence to weather conditions, RES are by definition intermittent and very unreliable, therefore they introduce a considerable uncertainty to the planning of energy production. Their large-scale incorporation and effective utilization by the contemporary electricity grid becomes even more complicated when taking into consideration that electricity storage is expensive and difficult. Therefore, the amount of electricity generated must be consumed at the time of its generation. If electricity balance (i.e. the difference between production and consumption) is not maintained for large periods, the grid could collapse with intolerable consequences, as entire regions could remain without power for many hours (i.e. blackout) [21, 49]. More formally, given a number of electricity producers I and a number of electricity consumers J, the electricity imbalance imb_t between production and consumption during a time interval t, is defined as:

$$imb_t = \sum_i prod_{i,t} - \sum_j cons_{j,t} \tag{2.1}$$

where, $prod_{i,t}, i \in I$ is the production of i during t and $cons_{j,t}, j \in J$ is the consumption of j during t.

Apart from the evolution of electricity production, customer habits (i.e. electricity consumption) change rapidly. The increasing use of information technologies and consumer electronics has lowered the tolerance for outages, fluctuations in voltages and frequency levels, and other power quality disturbances [66]. However, the large-scale induction of the computers in every aspect of our daily life reshapes the landscape of the energy sector (e.g. smart meters, electric vehicles), and creates opportunities to overcome past difficulties. For instance, in the past, due to the lack of means to communicate complex information, little flexibility has been built into the demand side of the electricity market [8].

Smart grid technology which incorporates bidirectional data flow, allows customers to adjust their energy consumption by using real-time information about the production, consumption, and prices of electricity. This control over electricity usage is called demand-side management (DSM). DSM programs comprise two essential activities: demand response programs which intend to transfer customer load during periods of high demand to off-peak periods where energy generation is less expensive; and energy efficiency and conservation programs, which encourage customers to give up some energy use and convenience in return for saving money [18].

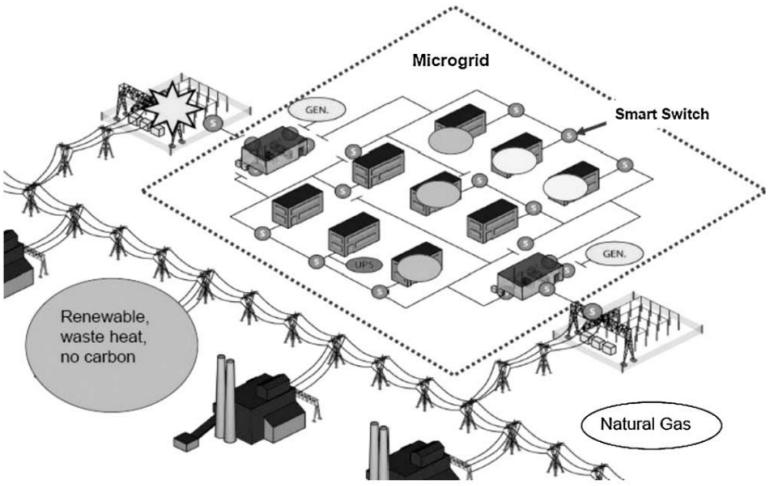

FIGURE 2.1: An overview of a micro-grid [4].

The aforementioned descriptions could be summarized in the following definition of Smart Grid which was given by the U.S. Department of Energy [66]: *A fully automated power delivery network that monitors and controls every customer and node, ensuring a two-way flow of electricity and information between the power plant and the appliance, and all points in between. Its distributed intelligence, coupled with broadband communications and automated control systems, enables real-time market transactions and seamless interfaces among people, buildings, industrial plants, generation facilities, and the electric network.*

A Smart Grid, apart from the high penetration of renewable energy sources, the extensive support of smart appliances and the sophisticated available pricing schemes, is essentially a collection of smaller interconnected grids. These "small grids" which are called micro-grids [4], vary in size from countries and big cities to small villages and neighborhoods. In more detail, a micro-grid (Figure 2.1) is an integrated autonomous energy system which contains distributed energy sources as they were previously described, together with a number of electrical loads as well as storage units (e.g. electric vehicles, batteries).

Electricity grids contain various operators which are responsible for different aspects of their operation. An entity named Transmission System Operator (TSO) is responsible to maintain the security and integrity of the grid by managing the energy transmission network and simultaneously, by operating a number of energy markets that enable energy exchange between the various market players. An entity named Distribution System Operator (DSO)[1], is responsible to maintain and ensure the effective operation of the energy distribution network, which is the infrastructure that carries energy from the transmission system to the individual consumers.

[1]In some countries, TSO and DSO are completely independent business entities with multiple counterparts operating in the same market (e.g. in the USA), while in other, these entities may have some

2.1.1 Electricity Markets

Markets have evolved over the years from mere locations where a few individuals would occasionally gather to trade goods, into virtual environments where information constantly flow and agreements are instantly reached. Electricity markets summarize rather accurately the contemporary meaning of a market and are developed based on the idea that energy can be treated as a commodity and therefore to be traded. However, there are critical differences between the electricity trading and the trading of other commodities (e.g. cubic meters of gas or stocks) [38]. Real world TSOs run multiple power markets which have explicit participation rules and minimum requirements [32]. A classification of these markets is the following:

- **Baseload Power**: The baseload power market is for the power that must be provided continuously throughout the year, usually at a very low cost. Nuclear, coal-fired, hydroelectric and natural-gas power plants are usually responsible to provide this type of power. These power plants are intended to be turned off during designated periods of maintenance and therefore the participation in these markets is highly constrained.
- **Peak Power**: Peak power is generated and purchased from this market at times of exceptionally high demand, usually on hot summer afternoons. Peak power is typically provided by gas generators that can be switched on and off for shorter periods of time, usually three to five hours.
- **Spinning Reserves**: Spinning reserves are these generators which are constantly synchronized to the grid and provide energy in case of unplanned and emergency events e.g. transmission line failures. As these generators are designed for contingencies, they are used limited during a year and even then, for durations up to one hour. The payment scheme of these generators is not depended on the duration of their actual utilization, but it is based on their availability. Therefore, they add a considerable expense to the operation of the electricity grid which is commonly conveyed on the bills of the final electricity consumers.
- **Regulation**: Regulation power is used to regulate frequency and voltage on the grid by matching the instantaneous power supplied by the grid with the instantaneous power demand. We have already mentioned the importance of the supply-demand balance in the grid. To provide regulation services and to participate in the regulation market, the participants must respond to a frequent real-time signal sent by the TSO in order to validate their availability.

independence, but ultimately operate under the same national business "umbrella" (e.g. in Greece). In the literature, the latter case is named independent system operator (ISO).

Markets, besides the players which participate and the products for which they are concerned, operate under a set of proper and strict rules [38]. These rules include the date of the delivery of the agreed products, the mode of settlements and the built-in conditions of the transactions. The aforementioned properties separate the markets in two distinct categories:

- **Spot markets**: In this kind of markets, the sellers deliver their goods immediately and the buyer pays for them "on the spot". When the deal is complete, neither party can back out. In spot markets, there are no conditions attached to the product delivery, as the whole transaction takes place in the same place.
- **Forward markets**: In this kind of markets, the sellers and the buyers sign forward contracts based on their individual interests, with the sellers pledging to deliver a particular product in a designated period and the sellers pledging to pay for the products according to a specific timeline. In this kind of markets, the contracts commonly contain various safeguards in order to protect the involved parties.

2.1.2 Electricity Tariffs

With the advent of technologies which enable the switching to a smarter grid, various pricing schemes have emerged for selling electricity both in wholesale and retailing markets. Some of these pricing schemes utilize the benefits of the smart grid aiming to support its efficient operation, providing incentives to consumers for rational electricity consumption [1]. Other, have been inherited from the old electricity grid and they are still used due to their simplicity and convenience. Some of them are the following:

- **Flat Rate:** In this pricing scheme, a supplier/utility charge a flat fixed price for every kWh consumed or produce which means that a customer's final charging is directly linked to its own energy profile. Typically, this stability offered to the customers comes with a cost, as flat rates typically incorporate service charges. This charging scheme is optimal for customers and grid with low smart metering infrastructure.
- **Time of Use (TOU):** TOU tariffs change according to the season, the day of the week, and in many cases according to the electricity imbalance of the grid. An individual day is segmented in time intervals, and each interval is assigned a fixed price. If energy consumers have knowledge over the peak hours, they can avoid energy consumption during these periods and shift to intervals where the energy is cheaper. For energy producers, peak hours indicate the periods over which their

service can provide an additional profit. Smart metering technology is extremely useful for the effective use of this pricing scheme.

- **Real-time Pricing (RTP):** RTP can be defined as energy prices that are set for a specific time period on an advance or forward basis and which may change according to price changes in the market. Prices paid for energy consumed during these periods are typically established and known to consumers a day ahead or an hour ahead in advance of such consumption, allowing them to vary their demand and usage in response to such prices and manage their energy costs by shifting usage to a lower cost period, or reducing consumption overall [1].

- **Prediction of Use (POU):** In POU tariffs, electricity producers and consumers are asked to predict a baseline for energy profile, and they are charged based both on their actual consumption and their deviation from their prediction (in the sense that units consumed or produced in excess/short of the baseline may be charged different marginal rates) [56, 72].

2.2 Smart Cities

Cities nowadays face complex challenges to meet objectives regarding socio-economic development and quality of life. The concept of "smart cities" is a response to these challenges as they incorporate the Internet of Things (IoT) approach and broadband network technologies to enable e-services which are very important for urban development, and drive the innovation in areas such as health, inclusion, environment, and business [10, 76]. A smart city is the interconnection of key industry and service sectors, such as Smart Governance, Smart Mobility, Smart Utilities, Smart Buildings, and Smart Environment [2].

Smart cities depend on a smart grid to ensure efficient and reliable delivery of energy to supply their many functions, present opportunities for conservation, improve efficiencies and, most importantly, enable coordination between urban officialdom, infrastructure operators, those responsible for public safety and the public health. The smart city is all about how the city "organism" works together as an integrated whole and survives when put under extreme conditions. Energy, water, transportation, public health and safety, and other aspects of a smart city are managed in concert to support the smooth operation of critical infrastructure while providing for a clean, economic and safe environment in which people live, work and play [26].

[2] http://www.smart-cities.eu

2.3 Electric Vehicle Charging

Electric vehicles (EVs) are considered as key technology in the direction of reducing our dependency on fossil fuels, therefore, they are central aspect of the smart grid agenda [49]. Pure EVs emit zero CO_2 as they are completely independent of ICOs, but to charge their batteries they require a considerable amount of electricity, which is multiple times higher than an average household electricity consumption.

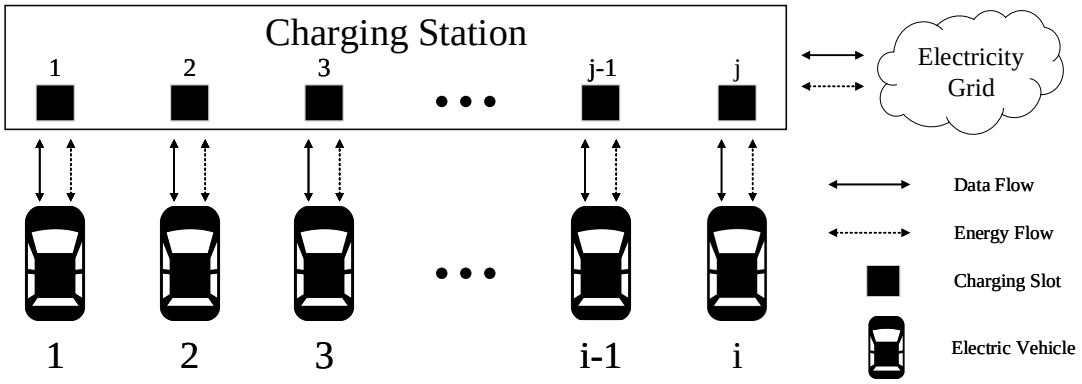

FIGURE 2.2: An overview a charging station with multiple charging slots.

EVs can charge their batteries at charging stations that provide the physical infrastructure to connect to the grid. Each charging station can be equipped with multiple chargers (charging slots), and each charging slot can be of a different type and power output [22]. A general view of a charging station with multiple charging slots is illustrated in Figure 2.2. A charging station is interconnected with the rest of the electricity grid at the power supply level, thus having access to the energy that then resells to the EVs to charge their batteries. In the scenario, a smart grid, the flow of electricity is bidirectional as a charging station can provide power back to the grid from that stored in the batteries of the EVs. At the same time, and in the same scenario, a charging station can communicate in real-time complex information and can offer advanced services to other Internet-enabled devices and stakeholders (Internet of Things, IoT).

There are two categories of charging stations based on their location:

1. **Home Charging Station**: Households with available private garage can easily be equipped with charging stations. However, as a household charging station is coupled with its electricity infrastructure, the type of charger and the charging rate it provides, is constrained from the infrastructure specifications. As a result, the charging duration at home is usually high, but the cost is low. Most EVs can charge during evening hours when their owners return home from work and want to be sure that their battery will be charged enough to allow them to drive the

next day [49, 67]. Therefore, a significant peak on the demand is expected during evening hours, which coincides with the existing afternoon peak on the household electricity demand.

2. **Public Charging Station**: In urban areas where the physical spatial planning do not allow the existence of home charging infrastructure, public charging stations and public garages equipped with chargers are the only charging option for EVs. These public charging stations usually offer a number of charging slots, possible of different connection type and charging rate (e.g. fast, rapid charging), and are owned from different providers. The pricing per kWh typically varies between them due to the different business models which they may have. The arrival/departure pattern to and from public charging stations is relatively different from households. Most EV drivers frequently prefer to connect their vehicle upon arrival at a selected destination (e.g. work, sports facilities, shopping malls) and expect to have their vehicles fully charged upon departure. This behaviour creates a similar problem with charging at home, but with the peak on the demand being shifted at different time intervals. Additionally, public charging includes and the of problem EV parking.

If an EV owner has a charging station at her home garage, then the charging station is reasonable to be equipped with a small amount of charging slots (the most likely one with two). However, if a charging station is public or is owned by a large parking operator, it is expected to be equipped with much more charging slots in order to meet the demand of a larger number of EVs.

Either charging at home or at a public charging station, it is evident that the uncoordinated EV charging may lead to excessive peak demand and potential blackouts. Therefore, the effective EV charging management in a way that individual EV owners will be always able to drive and also the electricity grid will be protected is a very challenging problem [49, 67]. The complexity of the EV charging problem increases if we consider settings where the penetration of intermittent and unreliable renewable energy sources is high. The reason for this is that large EVs populations offer the grid a large distributed battery in which it can potentially store the excessive renewable production which is very expensive to store alternatively. EVs are parked around 90% of the day, thus, with proper management, they can provide the grid very important balancing services [67].

In the literature, the management of EV charging is distinguished in two problems:

- **G2V Problem**: This is the problem of coordinating the simultaneous charging of EVs from the grid. Since they can draw a considerable amount of power, the grid overloads, which in turn leads to blackouts and infrastructure damage.
- **V2G Problem**: In contrast to G2V, the V2G problem is concerned with how EVs can supply power stored their batteries to the grid during power peaks. This can lower or even eliminate the need for spinning reserves. Since the batteries can charge when power is cheap (e.g. at night or during the time intervals with high renewable energy production) and return the power when it is more expensive, this raises an opportunity for profit for EV owners.

The coordination and scheduling mechanisms that are proposed to address the simultaneous electricity demand (herding effect) and the V2G activities can be either decentralized (bottom-up) or centralized (top-down) Each approach has advantages and disadvantages depending on scenario and the electricity market they are applied to [53, 58, 63, 69].

2.4 Coalition Formation

Coalition Formation (CF) is the problem of finding groups of agents that can join forces and act collectively (cooperate) towards a common goal. Each agent has individual strengths and weaknesses and by forming teams (coalitions), they complement each other yielding results that they could not if acted alone [11, 13, 48]. CF is a twofold problem:

- **Coalition Structure Generation:** Coalition structure generation, is the problem of creating a collection of non-empty subsets (i.e. coalitions) over all the available agents such that the union of all coalitions result in the actual agent space and that every agent is part of exactly one coalition.
- **Payoff Distribution:** The payoff distribution problem is concerned with the division of the coalitional outcome to the agents that were part of the cooperative effort. This must be done in such a way that the rewards are fair, and no agent can be motivated to leave his coalition.

The application of CT techniques to the smart grid domain enables the conversion of the various small, unreliable, almost invisible to the electricity grid, power electricity producers and consumers, to aggregated equivalent power entities that have increased

negotiation power and can participate into the electricity markets in a cost-effective manner. CF approaches have been used extensively in smart grid applications and more particular for the formation of cooperatives for electricity consumption shifting [1, 3], for the formation of Cooperative Virtual Power Plants (CVPPs) [14, 55], and for charging and discharging (V2G/G2V) EVs [15, 16, 70, 71]. For the latter case, the problem of assigning EVs to charging stations can be expressed as a CF problem. The first part of the problem is to determine which EVs are going to be assigned to specific stations, and the second part is to determine the value (payoff) each individual EV is going to receive for its contribution.

2.5 Recommendation Systems

The massive amount of information available on the Internet and the rapid introduction of new e-business services (buying products, product comparison, auctions, etc.) frequently overwhelms users, leading them to make poor decisions [6, 50]. Recommendation systems are software tools and techniques which are responsible to provide users with alternatives (suggestions) for items which they may want to use. Music, Movies and Shopping recommendation systems are extremely popular and have proved very effective in real-world settings. Recently, the research community started studying recommendations on the domain of electric vehicle charging [23], but these endeavors are currently in preliminary state.

In their simplest form, recommendation systems output a ranked list of different items which are provided to the user. The ranks represent the likelihood that an individual would like a particular item. Over the years, numerous recommendations approaches emerged which differ in the type and volume of information that they need and in the way they process the available data. Here we briefly review some notable approaches:

- **Content-based:** Content-based approaches try to recommend items similar to those a given user has liked in the past. The basic process performed by a content-based recommender consists in matching up the attributes of a user profile in which preferences and interests are stored, with the attributes of an item, in order to recommend to the user new, possibly interesting, items.
- **Collaborative filtering:** Collaborative Filtering (CF) approaches make suggestions to a user by using the items which other users with similar tastes liked in the past. The intuition of this approach is that people often get the best recommendations from someone with tastes similar to themselves.

- **Hybrid:** Hybrid recommendation systems are based on the combination of the techniques mentioned above. A hybrid system combining techniques A and B tries to use the advantages of A to fix the disadvantages of B. For instance, CF methods suffer from new-item problems, i.e., they cannot recommend items that have no ratings. This does not limit content-based approaches since the prediction for new items is based on their description (features) that are typically easily available. Given two (or more) basic RSs techniques, several ways have been proposed for combining them to create new hybrid systems.

2.6 Mechanism Design

Mechanism design (MD) is a sub-field of game theory that explores how to build systems (viewed as games) that compute optimal system-wide socially accepted solutions despite the self-interest of individual players (or agents) [42]. This means that these agents can not maximize their gains by misreporting their true preferences and intentions, thus, they have an incentive to be truthful. In MD literature, this property is called *incentive compatibility* and means that the design must be such, that actors finally choose willingly to follow a desired social choice function [1].

In the context of the smart grid, the requirements for planning and coordination of the production and consumption make MD approaches considerably effective. The traditional electricity pricing schemes often do not provide the appropriate incentives (or counter-incentives) to the electricity users to assist the reliable and effective operation of the electricity grid. Given that, we need systems that will mathematically guarantee desired social behaviours and rational electricity usage without jeopardizing the comfort and well-being of the end-users. At the V2G/G2V domain, it is imperative to incentivize the truthful reporting of charging preferences of EV's driver such as the arrival and departure to and from a particular charging point [54]. In the literature, several smart grid-related MD approaches have been proposed, and here we will briefly review some representative endeavors that motivated and inspired our own work.

Chalkiadakis et al. [14] propose a payment mechanism that guarantees that CVPPs have an incentive to truthfully report to the grid accurate estimates of their electricity production. In a similar setting, Robu et al. [55] utilize *scoring rules* and more particular the *Continuous Ranked Probability Score* (CRPS) to apply payments that incentivize the reporting of accurate electricity production predictions. Akasiadis and Chalkiadakis [3] present a mechanism for the coordination of large-scale demand shifting which is based on CRPS that has been evaluated using real-world consumption data. Here, consumption shifting agents report their confidence regarding meeting their expected

contributions, and this measure affects both their selectivity and final charging for electricity usage. At the domain of EV charging, Robu et al. [54] propose a dominant strategy incentive compatible online mechanism for the allocation of electricity units to EVs for charging. This is achieved by canceling unit allocations or by discharging over-allocated units before EV departure.

Chapter 3

Agents and Agent-Oriented Programming

In the previous chapter we introduced important background concepts for the smart grid with a focus on the V2G/G2V domain, as well as game-theoretic and AI approaches which are frequently used to tackle the problems arising in this domain. The terms agent and multi-agent system (MAS) are commonly mentioned in the related literature, but in the context of this thesis, we did not properly define their actual meaning and the advantages they provide. In this chapter we begin by briefly introducing the concepts of agent and multi-agent system and then the methodology and tools that are commonly used to design and develop integrated agent-based systems using agent-oriented programming (AOP) techniques. AOP is a relatively new software paradigm that brings concepts from the theories of AI into the realm of distributed systems.

3.1 Agent Definition

The concept of an agent has become important in both Artificial Intelligence (AI) and mainstream computer science. However, during the period of the emergence of the term, there was much debate around the question *what is an agent?*, and many researchers have provided their own interpretations [24]. Today, this debate is not as intense as old, since a considerable part of the agent-based research community seems to have converged to the following characterization[1] [73]:

[1] Without this meaning that other definitions or characterizations are wrong.

An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives.

The general properties which define the term agent (either hardware or software) and could be derived from the aforementioned characterization are the following [74]:

- **Autonomy:** agents operate without the direct intervention of humans or others and have some kind of control over their actions and internal state.
- **Social Ability:** agents interact and communicate with other agents (and possibly humans) via some kind of agent-communication language.
- **Reactivity:** agents perceive their environment, which may be the physical world, a user via a graphical user interface (GUI), a collection of other agents, the Internet, or perhaps all of these combined, and respond in a timely fashion to changes that occur in it.
- **Pro-activeness:** agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative.

Besides these properties, there are several more which describe an agent but under designated conditions. These include the agent mobility, which ability of an agent to move to different nodes of the available network in order to exploit remote resources or to cooperate with other agents, the assumption that agents are not malicious which is called veracity, and the assumption that agents act in order to achieve their private goals in a non-self-destructive manner which is called rationality [74].

When agent-oriented approaches are adopted to model a domain of interest, it comes naturally that this domain may include multiple entities that could be described by the term agent. These agents, as they act on behalf of their user or owner, they represent their interests and goals which may be different from these of other agents in the same environment. However, despite their selfishness, they may be able to cooperate towards a common goal [75] making decisions under uncertainty or lack of trust [47]. Either way, it soon becomes apparent that the majority of real world problems require or involve multiple agents [29]. These distributed systems which contain multiple agents capable of making independent decisions, are called multi-agent systems (MAS). An overview of such a system is illustrated in Figure 3.1. In general, multi-agent systems are categorized into cooperative and competitive depending on the inter-agent relationships.

Multi-agent systems have multiple advantages over single agent or traditional centralized systems. Some of their advantages are the following:

- They are decentralized and thus does not suffer from the "single point of failure" problem associated with centralized systems.
- They model problems in terms of autonomous interacting agents similar to components, which is proving to be a more natural way of representing open environments, task allocation and team planning.
- They provide solutions in situations where expertise is inherently distributed.
- They enhance the overall system performance and inherently provide computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness, flexibility, and reuse.
- They enable the interconnection and interoperation of multiple existing legacy systems. By building an agent wrapper around such systems, they can be incorporated into an existing agent society.

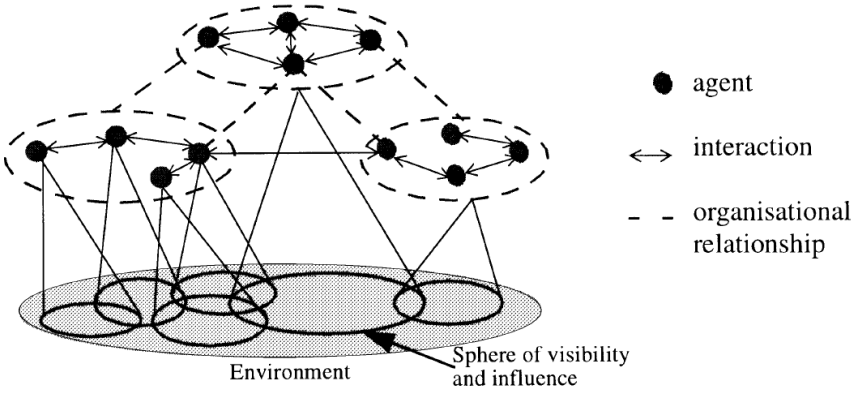

FIGURE 3.1: Generic overview of a MAS [29].

There are numerous real world applications which utilize the agent and multi-agent paradigm. Broadly speaking, these applications are divided into three main categories. The first category is the family of applications which use agents as processing nodes of a fully functional distributed processing system. Each agent, given an input, is responsible for a specific calculation and the combination of the results of these calculations form the final solution of the problem of interest. The second category is concerned with the utilization of agents as personal assistants. Agents represent actual users, act on their behalf and aim to maximize their satisfaction and utility. The final category contains applications which combine the aforementioned applications in order to gain the benefits of both. For example we may have a personal assistant agent like the one we mentioned earlier, but in a more complex environment. This agent consists of multiple agents and each of them has a specific computation to handle. These "worker" agents, report back

to their "manager" about the status of their computation who in turn is responsible to combine the results, utilize them, and assign new tasks to the workers [75].

3.2 Agent-Based Negotiations

Negotiations and more specifically, the automated negotiations (or agent negotiations), are important means for agents to resolve their differences and reach mutually acceptable agreements. The above imply that the agents which participate in a negotiation, are autonomous, self-interested, hold private information, have preferences, but also that they may have interest in cooperation [30, 46]. In a negotiation process, the common message types which agents exchange to denote their intentions are, proposal (or offer), counter-proposal, critique, accept-proposal, and reject-proposal. Despite the fact that negotiations are complex interactions and in many cases, domain specific, the negotiation theory can be summarized to the following three topics: *(a)* Negotiation Protocols, *(b)* Negotiation Objects and *(c)* Agent's Decision Making Models [30]:

3.3 Foundation of Intelligent Physical Agents

FIPA [2] is an IEEE Computer Society standards non-profit organization which was established in 1996 in order to develop a collection of standards and specifications for software agent technology. FIPA provides a framework within which agents which are implemented according to its specification dwell, operate and are managed. This framework defines the model of agent creation, registration, location, communication, migration, and operation, and is supported by a collection of entities with specific roles and purpose and are illustrated in Figure 3.2. In more detail:

- **Agent Platform (AP)** provides all the software and hardware infrastructure where agents are deployed. This includes machines (e.g. Servers, PCs, Mobile Devices), operating systems, as well as other FIPA agent management components (as we describe below), the agents which are implemented by a developer and all the ancillary software (e.g. libraries).
- **Directory Facilitator (DF)**, is an optional component that dwells in the AP and offers yellow page services to other agents. When agents intend to make public one of their services, they send a registration message to the DF agent with information about the service they want to register. Agents can search the DF

[2] www.fipa.org

for services which they are interested in or subscribe with the DF to periodically receive updates about them. Note that the DF component is considered to be a neutral and trusted entity within the AP in the sense that it has not an incentive to provide inaccurate information or mislead other agents. Finally, an AP allows the co-existence of multiple DFs that can form a DF federation.

- **Agent Management System (AMS)** is a mandatory component of an AP and it is responsible for orchestrating the agent activities such as creation, deletion, and overseeing the migration of agents to and from the AP. When an agent is created, it is obligated to register with the AMS in order to obtain an agent identifier. When an agent deregisters with the AMS, her life within an AP terminates. AMS holds a directory of all agents present within an AP, together with their status (e.g. active, waiting, suspended) and an agent description that is provided by the agent herself. An AP can only contain a single AMS and when AP spans to multiple machines, the AMS authority is extended to all of them.

- **Agent** is a computational process that inhabits an AP and typically offers one or more computational services that can be published as a service description. The particular design of these services, otherwise known as agent capabilities, is not the concern of FIPA, which only mandates the structure and encoding of messages used to exchange information between agents. An agent must have at least one owner and must support at least one notion of identity which can be described using the FIPA Agent Identifier (AID) that labels an agent so that she may be distinguished unambiguously.

- **The Message Transport System (MTS)** is a service provided by an AP to transport FIPA-ACL messages between agents on any given AP and between agents on different APs. Messages are providing a transport envelope that comprises the set of parameters detailing, for example, to whom the message is to be sent. For details about agent communication and FIPA-ACL messages, see Section 3.4.

In addition, FIPA proposes an abstract multi-agent architecture, that is, a collection of elements that enable the interoperability and reusability between agent systems possibly of different technology and implementation. These elements include mechanisms for agent registration, agent discovery and inter-agent message transfer (Figure 3.3). The actual implementation of the architecture is called realization and may include all or a subset of these elements. The purpose of the abstract architecture is to introduce a minimum number of required elements which are useful for an integrated multi-agent system but a realization may include additional elements which may not be described by FIPA.

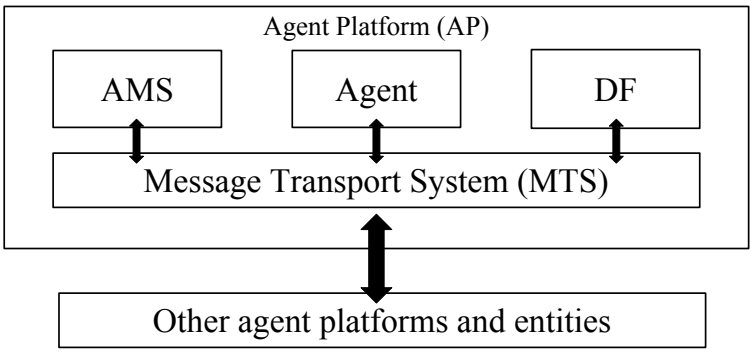

FIGURE 3.2: FIPA's agent management reference model.

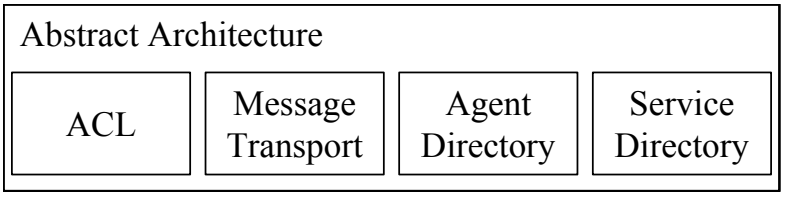

FIGURE 3.3: FIPA abstract architecture.

3.4 Agent Communication

Unlike the object-oriented communication which in reality is method invocations between objects, agent communication is based on the assumption that an agent is an autonomous entity and thus, other agents can not force her to perform an action or write data onto her internal data structures. What agents can do is to influence, motivate or simply request other agents to perform an action by formulating and sending individual semantically meaningful messages [75]. In practice, agents are a form of distributed and heterogeneous code processes, therefore, they are enabled to communicate using mainstream low-level communication protocols (e.g. TPC/IP, HTTP) [7].

From the infrastructure perspective (i.e. computer networks, low-level protocols), agents have means to communicate efficiently, however, in order to understand each other they need some form of communication language just as humans need natural languages (e.g. Greek, Arabic, Chinese). Earlier we mentioned that agents exchange messages in order to communicate, therefore, the definition of an agent communication language requires the description of the properties and semantics of these messages.

The majority of ACLs are considerably influenced by the speech act theory which originates from [5]. Speech act theory is predicated on the assumption that speech actions

TABLE 3.1: FIPA-ACL message parameters [7].

Parameter	Description
performative	Type of the communicative act of the message
sender	Identity of the sender of the message
receiver	Identity of the intended recipients of the message
reply-to	Which agent to direct subsequent messages to within a conversation thread
content	Content of the message
language	Language in which the content parameter is expressed
encoding	Specific encoding of the message content
ontology	Reference to an ontology to give meaning to symbols in the message content
protocol	Interaction protocol used to structure a conversation
conversation-id	Unique identity of a conversation thread
reply-with	An expression to be used by a responding agent to identify the message
in-reply-to	Reference to an earlier action to which the message is a reply
reply-by	A time/date indicating by when a reply should be received

just like other actions are performed by agents intending to fulfill their goals [75]. Performative verbs, which are described and defined in terms of beliefs, desires, intentions, and other similar modalities (e.g. *request*, *inform*, *propose*, *promise*), accompany an agent's message and are used to utter the specific intention that the message is serving. Additional to message performatives, ACLs include definitions about the structure and parameters of the exchangeable messages, as well as the terminology and rules that are allowed for the knowledge representation.

Over the years, many agent communication languages (ACL) have emerged each having their own pros and cons. The two major ACLs which in practice share the same foundation are the Knowledge Query and Manipulation Language (KQML) and FIPA-ACL [39]. Nowadays, FIPA-ACL is enjoying widespread usage through the JADE framework (see Section 3.6) and since it is used in the context of this thesis, we are going to briefly discuss some fundamental FIPA-ACL features.

Table 3.1 illustrates the parameters of a FIPA-ACL message according to the FIPA message structure specification. There are no mandatory parameters as the specification intends to provide flexibility, however, some of them are unofficially mandatory (e.g. the sender, receiver, content). FIPA suggests three different encodings for its messages: String, XML and Bit-Efficient. FIPA-ACL provides a list 22 communicative acts (CA) (i.e performatives) which are based on speech act theory and are fully defined in an independent FIPA specification. Typical communicative acts include the *request*, *inform*, *propose*, *call for proposal* (*cfp*), *refuse*, *subscribe* and are assigned to the *Performative* parameter of a FIPA-ACL message.

In agent communication, the information which the various agents can exchange vary from mere words and numbers to complex data structures with many different variables,

reflecting all the information spectrum in the real world. This complexity requires the adoption of a well-defined syntax and rules so that the information that is sent to be parsed and recognized identically. The aforementioned syntax and rules are defined in the literature as content languages [7, 39]. An ACL message provides the parameter *content* for the exchangeable knowledge. FIPA, do not mandate the usage of a specific content language, but through an independent specification proposes the FIPA-SL language, a human-readable string-encoded first-order modal language which can represent propositions, objects, and actions.

```
(request
    :sender (agent-identifier :name alice@mydomain.com)
    :receiver (agent-identifier :name bob@yourdomain.com)
    :ontology travel-assistant
    :language FIPA-SL
    :protocol fipa-request
    :content
      ""((action
          (agent-identifier :name bob@yourdomain.com)
           (book-hotel :arrival 15/10/2006
                       :departure 05/07/2002 ... )
      ))""
)
```

FIGURE 3.4: FIPA-ACL message with a request performative [7].

Figure 3.4, depicts a FIPA-ACL message whose *content* is encoded as String using the FIPA-SL content language and an ontology named *traver-assistant* (Refer to Section 3.5 for details about ontologies), and which is part of a message sequence that follows the *FIPA-Request-Protocol*. In detail, the message sender (Alice), requests the receiver (Bob) an action (action construct), to book her a hotel room (book-hotel term), for the period between arrival and departure (two constant date terms).

Furthermore in agent communication there exist predefined sequences of messages that can be applied in several situations that share the same communication pattern regardless of the application domain [7]. These sequences of messages are defined as high-level protocols and typical examples include the negotiations and auctions (e.g. English Auction, Vickrey Auction). Based on this assumption, FIPA provides a collection of specifications for general interaction protocols with the most notable examples being the *FIPA-Request-Protocol*, the *FIPA-Contract Net-Protocol* and the *FIPA-Subscribe-Protocol*.

3.5 Ontologies

People, organizations and in our case, software agents, constantly communicate between and among themselves. However, the different preferences and background contexts, create widely varying viewpoints and assumptions regarding what is essentially the same subject matter. The consequent lack of a shared understanding leads to poor communication and difficulties in identifying requirements between these entities [65]. The tackling of this problem requires the establishment of a shared understanding of the domain of interest in order to reduce conceptual and terminological confusion. An ontology is the term used to refer this shared understanding of some domain of interest. An ontology embodies some sort of worldview with respect to a given domain and often is conceived as a set of concepts (e.g. entities, attributes, processes), their definitions and their inter-relationships [65, 75].

In the context of multi-agent systems, if the various agents are to communicate about some domain, it is necessary for them to agree on the terminology they will use to describe this domain. Additionally, if these agents have been engineered by different organizations and developers, the establishment an ontology makes the collective agent development process easier and their communication more effective [68].

In summary, the reasons to develop an ontology are the following [43]:

- To share a common understanding of the structure of information among people organizations or software agents
- To enable reuse of shared understanding and domain knowledge
- To make domain assumptions explicit
- To separate domain knowledge from the operational knowledge
- And to analyze the domain knowledge

3.6 Java Agent Development Framework

JADE[3][7] is a software development framework for the implementation of fully distributed multi-agent systems. It provides a simple yet rich Java API, as well as a collection of features that enable peer to peer asynchronous agent communication, agent service discovery, management of ontologies and context languages, effective debugging and monitoring through graphical tools. In the context of this work, we choose JADE

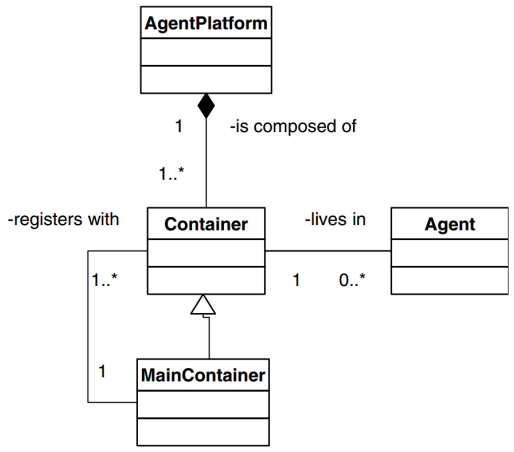

FIGURE 3.5: Relationship between AP, containers and agents [7].

over other alternatives [64] due to the aforementioned properties, its widespread usage from academia and industry, and its open-source nature.

Each running instance of the JADE environment is called container and each of them can contain multiple agents. A collection of active containers is called Agent Platform. In an AP, the first container to start is the Main container and every container that follows is normal. When a normal container starts it must be told where to find their Main container (in terms of IP address and port). These relationships between the AP, the various containers and the agents are depicted in Figure 3.5.

An example JADE system is illustrated in Figure 3.6. It is composed of two platforms which contain three and one containers respectively. *Platform1* contains four developer implemented agents in total, while *Platform2* contains only one. In JADE, the various platforms, containers and agents, communicate by using appropriate message transport protocols [7]. In JADE, both the DF and AMS as defined by FIPA (Section 3.3), are implemented as agents that provide and registration and exploration services to the various other agents.

Figure 3.7 shows the primary JADE graphical interface which is provided by the JADE's Remote Monitoring Agent (RMA) and allows the system administrator to monitor and manage the running platforms. The letter A indicates the root folder which contains all the available APs, each of them being identified by an IP address and a port. The containers of an AP are indicated by D. The Main Container must contain the AMS, DF and RMA agents which are indicated by C. Finally, the agent(s) which are implemented by a developer, are identified by a name and the AP identification and in Figure 3.7 are indicated by C.

[3] `http://jade.tilab.com/`

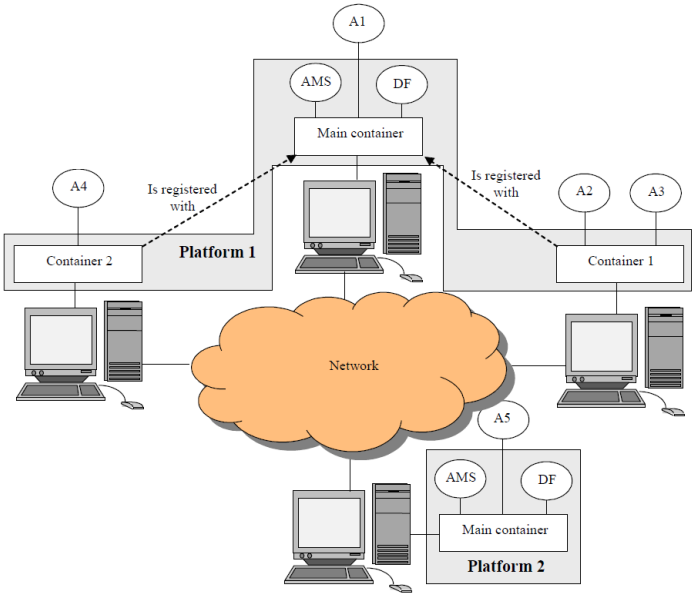

FIGURE 3.6: A scenario of a JADE based system [9].

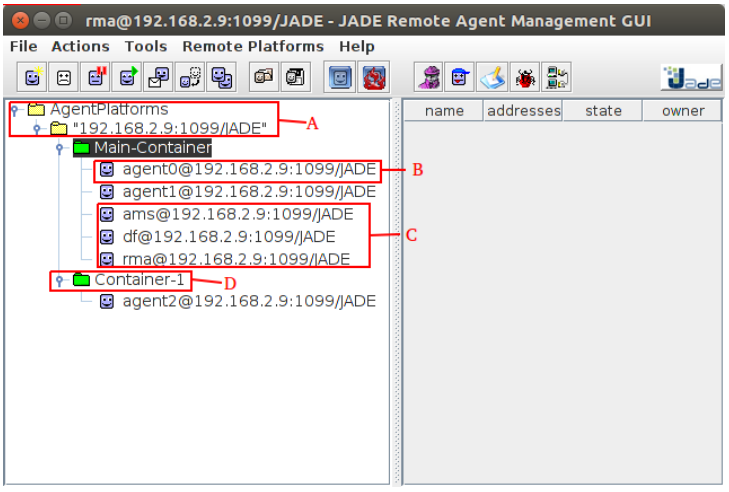

FIGURE 3.7: Example GUI of the JADE RMA showing the basic platform elements.

Apart from the RMA, the framework provides a collection of console and graphical tools which help the developer in the management and debugging phase and are essential since MAS applications are quite complex. Such tools are the *DummyAgent*, which is very useful for sending ACL messages to ping implemented agents; the *SnifferAgent* which is a tool for debugging and/or documenting conversations between agents; and the *IntrospectorAgent* which is a tool that allows the monitoring of an agent's life cycle, and its queues of sent and received messages.

3.6.1 Basic Programming Interface

In JADE, agents are all the Java classes that extend `jade.core.Agent` class. This class provides a collection of basic agent methods and among them, the `setup()` method which the developer typically has to override in order to include custom agent functionality. An agent can delete herself from an AP by using the `doDelele()` method that is provided by the `Agent` class. Before an agent is deleted, the `takeDown()` method is invoked and its responsibility is to carry out clean-up operations (e.g. deregister from the DF, GUI termination). According to FIPA, within an AP, each agent has a unique agent identifier and in JADE it is represented as an instance of the `jade.core.AID` class. `Agent` class enables communication with other agents using an asynchronous communication model based on FIPA-ACL messages.

The activities an agent needs to perform are typically carried out within behaviours which are all these classes that extend the `jade.core.behaviours.Behaviour` class. Each behaviour performs its designated operation by executing the method `action()` that is inherited from the `Behaviour` class. JADE provides a behaviour library that enables the composition of more complex actions by combining different behaviours in various ways. Some representative behaviours are the following:

- `OneShotBehaviour`: An atomic behaviour that is executed exactly once. Atomic is a behaviour that does not contain any other sub-behaviour.
- `CyclicBehaviour`: An atomic behaviour that is executed constantly.
- `TickerBehaviour`: An atomic behaviour that is executed periodically according to a configurable timer.
- `WakerBehaviour`: An atomic behaviour that is executed exactly once after a configurable timeout.
- `FSMBehaviour`: A composite behaviour that executes its sub-behaviours according to a FSM defined by the user. Each sub-behaviour represents a state of the FSM.
- `SequentialBehaviour`: A composite behaviour that executes its sub-behaviours in sequential order and terminates when its last sub-behaviour has ended.

Each agent has a built-in list that contains the active behaviours that are scheduled for execution according to the order of their insertion. New behaviours can be added into the behaviour list by using the `addBehaviour()` exposed from `Agent` class. The round-robin, non-preemptive behaviour scheduling implies that when a behaviour is scheduled for execution, its `action()` method is invoked, it runs until it finishes and

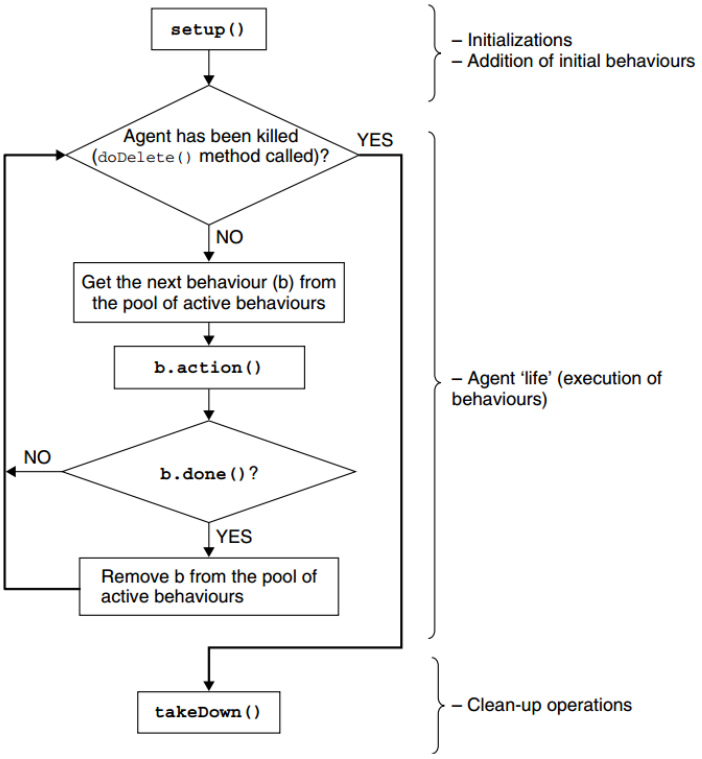

FIGURE 3.8: JADE agent execution policy.

it can not be interrupted. A behaviour returns when its `done()` method returns `true`. A behaviour can release the execution control with the use of the `block()` method, or it can permanently remove itself from the behaviour list in runtime through the `removeBehaviour()` method. The schematic overview of the path of execution of an agent thread is depicted in Figure 3.8.

Note that behaviours and can be spawned, suspended or killed at any given time as a behaviour should be seen as a way to spawn a new (cooperative) execution thread within the agent. In terms of data exchange, sharing and storage, JADE behaviours can use the provided from JADE `jade.core.behaviours.DataStore`. This makes the developed behaviours completely independent from a specific application, enabling the reusability of the behaviour code.

Note that in JADE, an agent is a dedicated Java thread and all her behaviours are always executed by this particular thread. The intention of this design approach makes JADE an effective and robust framework in environments with limited resources such as smartphones or other mobile devices. Nevertheless, as JADE is implemented in Java, developers can spawn additional execution threads but the management of these threads requires special management from the developer. When there are no behaviours available for execution, the agent's thread goes to sleep in order to lighten the CPU load.

JADE agents can send and receive messages by using the `send()` method of the `Agent` class and pick up messages from their message queue using the `receive()` method. JADE provides message filtering mechanisms (through the `jade.lang.acl.MessageTemplate` class), thus, we can specify rules about the properties of the messages that specific agent behaviours will receive and consume.

3.6.2 Agent Communication in JADE

The agent communication is probably the most central JADE feature and it is implemented considering the guidelines of the related FIPA specification for agent communication as we discussed in Section 3.4. It provides a built-in message transport system which facilitates the inter-agent communication in two different cases:

1. Agents which share the same container, communicate using the JADE Internal Message Transport Protocol, which by default is using Java Remote Method Invocation (RMI). This choice was made by JADE developers and it is not contained in a FIPA specification.

2. Agents of different APs communicate via HTTP-based Message Transport Protocols (MTPs). This is an official FIPA specification guideline.

When JADE agents exchange FIPA-ACL messages, they have to parse the FIPA-SL syntax of these messages in order to understand their contents. Additionally, they must have shared understanding (through an ontology), about the domain of discourse in order to ascribe the same meaning to symbols for the communication to be effective. JADE agents are Java-based, thus, the content of their messages is represented using Java objects. When an agent sends a message, she has to convert its internal Java representation into the corresponding ACL content expression while the receiver needs to perform the exact opposite conversion (see Figure 3.9). JADE provides support for content languages and ontologies in order to automatically perform all the above conversions and verification operations. JADE supports the various content languages such as SL, XML, LEAP, but a developer is free to create custom content languages by developing proper codecs for their interpretation.

In order to perform the proper semantic checks on a given content expression, JADE provides a message content ontology which is used to classify all the elements of the domain in which agents operate. The classification is derived from FIPA-ACL specifications and requires the content expression of a message to contain semantics which align with its performative (i.e. communicative act). These semantics include:

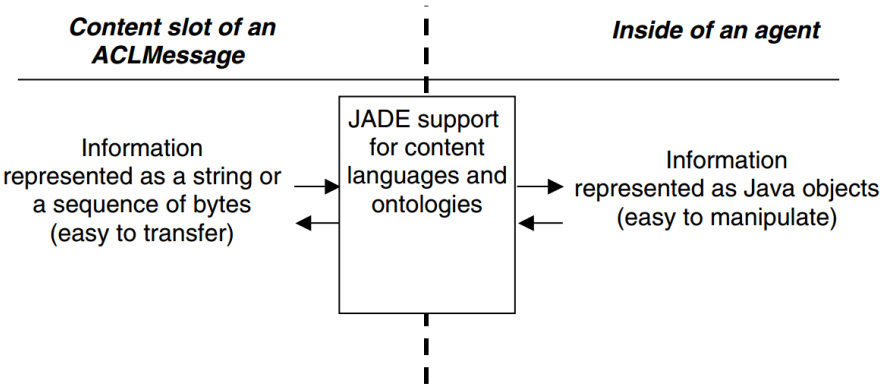

FIGURE 3.9: JADE message marshalling and unmarshalling [7].

- **Predicates**: Represent expressions that say something about the status of the world (e.g. `(Lives (Person :name George) (City :name "Chania"))`) and can be used as content of messages which contain the *inform* performative.
- **Terms**: Represent expressions identifying entities that have real existence in the domain of agent operation (e.g. `(Person :name Nikos :age 20)`).

Terms are further categorized into:

- **Primitives** (i.e. Integers, Strings, Doubles) and **Concepts** (i.e. Complex objects with multiple fields). These categories make no sense if used directly as the content of an ACL message.
- **Agent Actions**: represent special types of concepts that incidate the actions which agents have to perform (e.g. `Buy (Book :title "Game of Thrones") (Person :name Charis)`). This category only makes sense as content of messages which contain the *request* performative.

A domain specific ontology in JADE is a collection of entities which implement one of the following marking interfaces: `jade.content.Predicate`, `jade.content.Predicate` or `jade.content.Predicate`.

3.7 Agent Systems Engineering Methodology

The Agent Systems Engineering Methodology (ASEME) [59] is an Agent Oriented Software Engineering (AOSE) methodology for developing multi-agent systems. It uses the Agent MOdeling LAnguage (AMOLA), which provides the syntax and semantics for creating models of multi-agent systems covering the analysis and design phases of a software

development process. It supports a modular agent design approach and introduces the concepts of intra- and inter-agent control. The former defines the agent's behavior by coordinating the different modules that implement his capabilities, while the latter defines the protocols that govern the coordination of the society of the agents. ASEME applies a model driven engineering approach to multi-agent systems development, so that the models of a previous development phase can be transformed to models of the next phase. Thus, different models are created for each development phase and the transition from one phase to another is assisted by automatic model transformation, including model to model (M2M), text to model (T2M), and model to text (M2T) transformations leading from requirements to computer programs.

Agent Modeling Language (AMOLA [60]) describes both an agent as a unit and agents as part of community (multi-agent system). The concept of functionality is defined to represent the thinking, thought and senses characteristics of an agent. Then, the concept of capability is defined as the ability to achieve specific goals (e.g. the goal to decide which movie to see tonight) that requires the use of one or more functionalities. Therefore, the agent is an entity with certain capabilities, including inter and intra-agent communication. Each of the capabilities requires certain functionalities and can be defined separately from the other capabilities. The capabilities are the modules that are integrated using the intra-agent control concept to define an agent. Each agent is considered a part of a community of agents, i.e. a multi-agent system. Thus, the multi-agent system's modules are the agents and they are integrated into it using the inter-agent control concept. The originality in this work is the intra-agent control concept that allows for the assembly of an agent by coordinating a set of modules, which are themselves implementations of capabilities that are based on functionalities. Here, the concepts of capability and functionality are distinct and complementary. In order to represent system designs, AMOLA is based on statecharts, a well-known and general language and does not make any assumptions on the ontology, communication model, reasoning process or the mental attitudes (e.g. belief-desireintentions) of the agents giving this freedom to the designer.

Capabilities are decomposed to simple *activities*. For example, a decision making capability fulfills the task of finding where to recharge this evening. The problem is decomposed to activities, e.g. of finding which stations are nearby, what are their available charging slots, rank them, etc. Each activity is realized by a specific *functionality*, i.e. a specific technique like an algorithm, invocation of a service, etc. The same capability can be implemented with different functionality types: an agent can have a *decision making task* based on an *argumentation-based functionality*, while another implementation of the same capability could be based on a different functionality, e.g. *multi-criteria-based functionality*.

ASEME (through its IDE) automatically generates portions of the agent code or provides guidelines for the programmers to transform their design models to implementation models. As JADE is used widely from many developers as underlying agent development framework because of its multiple advantages, ASEME authors have chosen JADE and Java as their code generation output [62]. Figure 3.10 briefly illustrates the various ASEME phases, the different levels of abstraction and the models related to each of them. Note that in the context of this thesis, we do not utilized ASEME for code generation, but the design and representation of the inter-agent and inta-agent controls of our system are inspired from ASEME.

Development Phase	*Levels of Abstraction*		
	Society Level	*Agent Level*	*Capability Level*
Requirements Analysis *AMOLA Models*	Actors Actor Diagram	Goals Actor Diagram	Requirements Requirements per goal
Analysis *AMOLA Models*	Roles and Protocols Use case Diagram, Agent Interaction Protocols	Capabilities Use case Diagram, Roles Model	Functionalities Functionality Table
Design *AMOLA Models*	Society Control Inter-agent control model, Ontology, Message Types	Agent Control Intra-agent control model	Components
Implementation	Platform management code	Agent code	Capabilities code
Verification	Protocols testing	Agent testing	Component testing
Optimization	Number of instantiated agents	Agent resources	Code optimization

FIGURE 3.10: ASEME development phases with their outputs [59].

3.7.1 Statecharts

A statechart is a finite collection of states and transitions. A state is either a basic state or a hierarchical state and may contain multiple sub-states [27]. A hierarchical state can be either an AND-state or an OR-state. AND-states have orthogonal components that are related by "and" (executed in parallel) while OR-states have sub-states that are related to each other by "exclusive-or". Basic states are those at the bottom of the state hierarchy, i.e., those that have no substates. Transitions are binary relations between states and are usually triggered by events. Such events can be:

- A sent or received inter-agent message
- A change in one of the executing state's variables (also referred to as an intra-agent message)
- A timeout

- The ending of the executing state activity (no expression transition)

A message event is expressed by $P(x, y, c)$ where P is the performative, x is the sender role, y the receiver role and c the content of the message. The items that the designer can use for defining the state transition expressions are the message performatives, the ontology used for defining the messages content and the timers. An agent can define timers as normal variables initializing them to a value representing the number of milliseconds until they timeout. Timers are initialized in the action part of a transition expression, while the timeout predicate can be used in both the event and condition parts of the transition expression depending on the needs of the designer. The definition of statechart as it is used in AMOLA, makes possible to proceed to the definition of the inter- and intra-agent control.

Chapter 4

Related Work

To date, many simulation tools and real-world prototypes that incorporate a wide range of features related to the V2G/G2v domain have been proposed. These systems either focus on underlying issues related to the EV operation in the smart grid or they are integrated environments (either for simulation or real use) that include many entities that are related to some extent with the V2G/G2V domain.

Rigas et al. [52] present EVLibSim, a Java tool for the simulation of EV charging stations built on their previous work on EVLib [34]. This tool provides a friendly and comprehensive UI for the management of charging stations that allows their creation, modification, and monitoring. The tool simulates the operation of each charging station and provides the results to facilitate the comparison and evaluation of the various approaches. Although proven quite useful by test usage scenarios involving actual domain experts, this approach is restricted to charging stations only and does not incorporate additional stakeholder types.

Jordán et al. [31] propose a MAS to support the decision-making process on the determination of locations of EV charging stations in the city of Valencia. The system integrates a collection of information from heterogeneous data sources such as traffic data, social network data, charging station pricing data, and optimize the charging station locations using a genetic algorithm. Evaluation of experimental results shows that the higher the proposed solutions' fitness values are, the better coverage of the more populated and crowded areas of the city of Valencia is achieved. This approach might come handy for designing charging infrastructure, however, it does not capture what happens next, when it is offered for use.

Kamboj et al. [33] present an approach for forming coalitions of EVs to provide V2G and demand-side management services to the electricity grid. Authors incorporate a MAS

architecture and implement simulations using JADE. The system considers an intelligent agent for each EV, an aggregator agent responsible for forming coalitions of EVs, and a TSO agent that communicates with aggregators and regulate the V2G/DSM process. EVs are selected in a way, such that the minimum energy required to successfully participate in the regulation market is reached. Evaluation, however, involved five EVs only. Moreover, the system does not allow for complex EV selection processes and thus cannot scale.

Papadopoulos et al. [44] propose a MAS implemented in JADE that coordinates the battery charging of EVs considering the individual preferences of the EV drivers, by using specific search techniques and neural networks. The driver preferences include the willingness to participate in V2G, as well as the charging availability of the vehicle. Through a series of experiments, the proposed MAS is shown to have the ability to satisfy autonomous EV owners' charging preferences under both normal, and emergency grid conditions. Then, [35] proposes a MAS EV charging management system that satisfies the energy requirements of a large number of EVs considering owners' preferences. The problem is formulated as a non-cooperative dynamic game [41], that converges to an equilibrium when players are weakly coupled. This means that EVs do not have perfect information regarding all other EVs preferences and states, but can optimize based on the known averages of such values (corresponding to "mass behaviour"). Simulations on a realistic setting show that EV energy requirements can be allocated efficiently, and achieve "valley filling" during off-peak hours, a capability that provides multiple advantages for the Grid. None of these works, however, defines specific protocols and messaging ontologies, nor do they pinpoint how to extend the proposed frameworks by adding different agent types or algorithms, as we do in our approach.

In our work, we present an integrated MAS prototype for the V2G/G2V problem which is based on a high-level architecture to allow the study and evaluation of different approaches of the various modules required in such a setting. We define specific communication protocols and a domain-specific ontology for the inter-agent communication. The proposed protocols are open, in the sense that they can be easily extended or tailored to capture any real-world case that exists since we provide analytic descriptions and semantic schemes for each module that agents in such settings would utilize. Using our system, the designer or researcher can test and compare algorithms of their choice for providing charging recommendations, calculating the charging schedules for large numbers of EVs, and the effect of different pricing schemes, in real-world use-cases. To show the applicability of our approach, we implement a functional prototype based on the proposed architecture and execute combinations of different algorithms for various use-cases to compare their results.

Chapter 5

System Architecture

In this chapter, we present the architecture for the V2G/G2V problem. It follows a multi-agent architecture in the sense that the various stakeholders have been modeled as intelligent agents who can act autonomously without interventions or dictations of other, centralized third parties. The decision regarding which the stakeholder agents are was taken considering the related literature. The communication of these agents is based on the exchange of appropriate messages that adhere the FIPA-ACL, an agent communication language introduced by FIPA (see Section 3.4), and by using well-defined communication protocols specifically tailored to our application domain.

5.1 V2G/G2V System Architecture

The various stakeholders and services that were briefly presented in Section 1.1, are discussed here in much more detail, as agents who compose an integrated multi-agent V2G/G2V architecture. We assume that these agents live and operate in a smart grid infrastructure that can be interconnected with the other parts of the electricity grid through Distribution and Transmission Networks. When the grid under study requires power that can not be generated locally, it is capable of importing it, while when it has a local energy surplus, it can export it to other grids and create additional profits for the electricity producers. As we have already mentioned in Chapter 3, agents are distributed distributed software processes that can make decisions independently, thus other agents do not have control over their internal state and private information. Agents can only communicate using agent communication languages and specific interaction protocols. Figure 5.1 illustrates an overview of the agents of our architecture, their interactions, as well as their basic components.

The agents that participate in the proposed MAS architecture are *(a)* Electric Vehicle agents (EV), *(b)* Charging Station agents (CS), *(c)* Electricity Producer agents (EP), and *(d)* Electricity Consumer agents (EC). Furthermore, we assume the existence of a national regulatory service for energy, or possibly a profit-making private service, that consists of *(i)* the Station Recommender agent (SR), *(ii)* the Electricity Imbalance agent (EI), and *(iii)* the Mechanism Design agent (MD). We chose to refer to such a service by its three distinct modules separately, since our focus in this thesis is on the technical details of the functionality provided by each, and not on the business and regulatory aspects.

In Figure 5.1, each agent of our architecture is depicted as a rectangle and contains a collection of modules, i.e. the computation and storage sub-systems that are associated with the particular agent. Each module is private in the sense that only the particular agent that contains that module can access it. The agents that have multiple counterparts in our architecture are denoted with a star superscript near their name. The grey coloured elements denote required functionality that, however, has not been implemented in the context of this thesis.

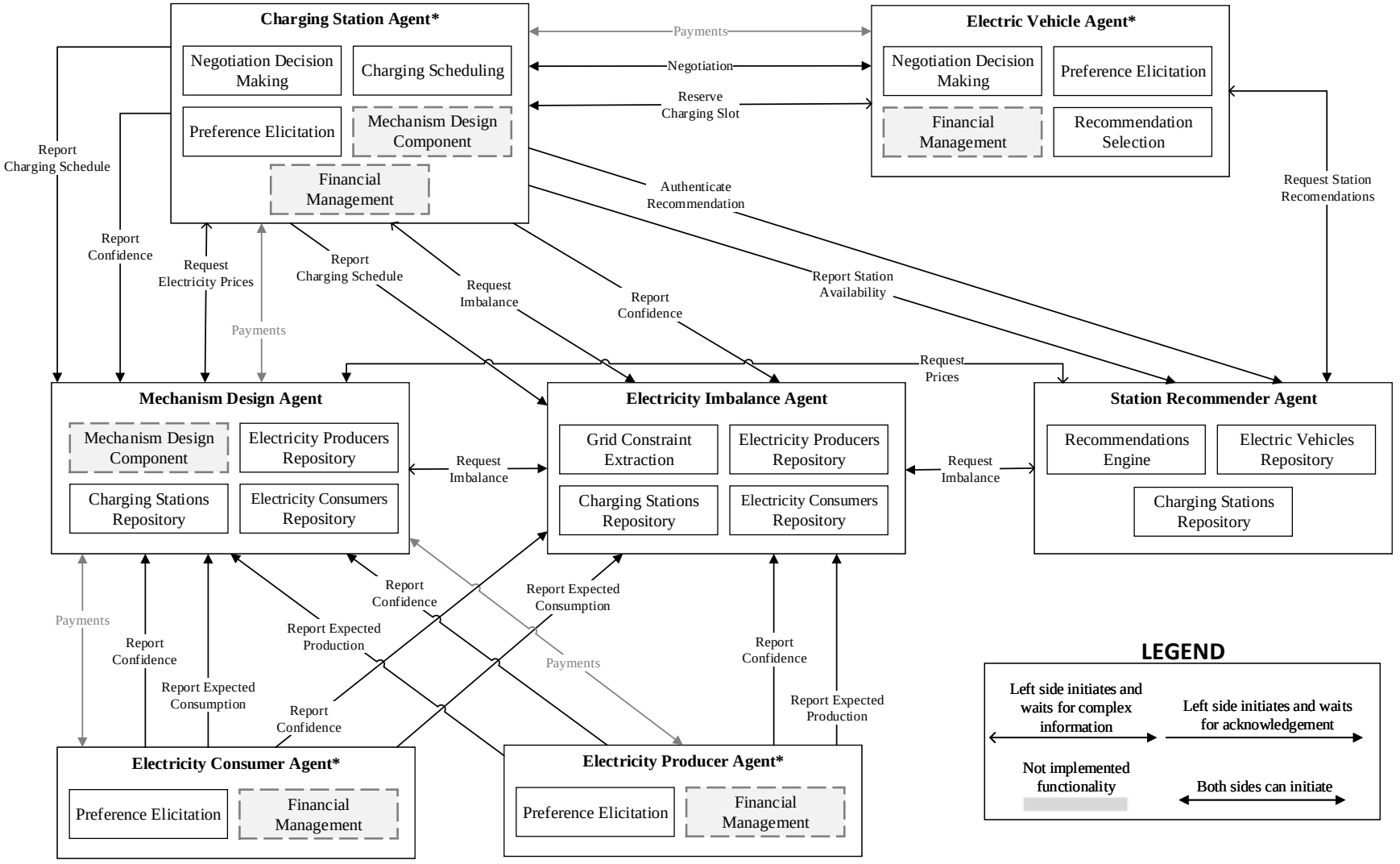

FIGURE 5.1: An overview of the proposed architecture for the V2G/G2V problem. With a star we noted the agents that can have multiple counterparts while in gray color we have note the functionality that is out of the scope of this thesis.

5.2 Agents

Electric Vehicle Agent: Each EV has an intelligent agent which we assume to be installed and executed on a computational infrastructure that is part of the EV, or on a mobile device that its driver possesses. The goals of an EV agent is to ensure that the vehicle will always have enough charge for driver's next trip. It monitors the activities and behaviour of the driver, models and predicts his future behavior and needs, and then contracts a charging station to schedule its battery charging. It also aims to create a profit to the driver by participating in V2G activities when the vehicle is parked for long periods of time. If one of the driver's needs changes and he does not want to charge under the active agreement with the charging station as originally decided by the agent, the agent can initiate a negotiation regarding the initial agreement, or cancel it and make a new arrangement to accommodate driver's latest mood and preferences. An EV agent contains a Preference Elicitation module that is responsible to monitor the driving habits and behaviour of the driver, and predict its future charging preferences; a Financial Management module that is responsible for making the payments regarding the charging of the EV and receiving payments when the EV participates in V2G activities; a Negotiation Decision Making module that contains the algorithmic components and the logic according to which the agent conducts negotiations; and a Recommendations Selection module that contains the algorithmic components for the agent to determine which charging recommendation to select. An EV agent communicates with the Station Recommendation agent and the Charging Station agents (see below).

Charging Station Agent: The goal of a Charging Station agent is to look after the best interests of the owner of the charging station, by providing the physical gateways (i.e. connectors, parking slots) to EVs that connect to the grid and create profit by charging the EV batteries. Another goal of the Charging Station agent is to negotiate with Electric Vehicle agents an already made charging agreement, in order to change some of its parameters and in this way to create space in its charging schedule for more vehicles to charge. This leads to a better utilization of the station infrastructure and maximizes the profit for its owner. A Charging Station agent contains a Charging Scheduling module that is the algorithmic components responsible to schedule the charging and discharging of EVs over a predefined planning horizon (e.g. 24 hours ahead); a Negotiation Decision Making module that contains the algorithmic components and the logic according to which the agent conducts negotiations; a Mechanism Design module that calculates the EV payments in such a way to incentivize rational behaviour; a Preference Elicitation module that monitors charging slots usage and updates the prices for each of them according to the needs of the station owner; and a Financial Management module that is responsible for making the payments regarding the operation the charging station.

A Charging Station Agent communicates with the Electric Vehicle agents, the Station Recommender agent, the Electricity Imbalance agent and the Mechanism Design agent.

Station Recommender Agent: The goal of this agent is to recommend the charging stations and charging slots that match the most with the EVs' preferences and current battery state, while ensuring that herding effects do not occur and allowing the EVs to participate in V2G activities. The Station Recommender agent contains a Recommendations Engine module that makes charging stations recommendations, an EV Repository module where it stores information about the past EV behaviour in order to utilize it for future recommendations, a Charging Station Repository of registered charging station. It communicates with the Electric Vehicle agents, the Charging Station agents, the Electricity Imbalance agent, and the Mechanism Design agent.

Electricity Imbalance Agent: The goal of this agent is to aggregate information from the Electricity Producers, Electricity Producers and Charging Stations agents, regarding their expected energy profiles, and to calculate the periods of electricity shortage and surplus. Then, it provides the electricity imbalance to all interested parties, for them to plan their electricity consumption and production activities. It employs a Grid Constraints Extraction module that can incorporate various measures and methods that could be relevant in a Smart Grid scenario, and calculates electricity imbalance over a predefined planning horizon, and a Charging Stations Repository, an Electricity Producers Repository and a Electricity Consumers Repository, where it stores information about the registered agent types, respectively.

Mechanism Design Agent: This agent represents an intermediate trusted third party entity which is responsible for the prices calculation and the payment flows. In the general case, whoever communicates with this agent, does so for anything related to payments. The goal of this agent is to charge producer and consumer agents with appropriate, possibly personalized fees, so that they are incentivized to be truthful regarding their statements for expected values, and in this way promote the reliable and smooth operation of the system. By collecting expected profiles and by observing the actual production and consumption for the corresponding time intervals, the Mechanism Design agent calculates the various payments. These payments are such, so that to reward the agents who provide accurate and truthful reports regarding their expected energy profiles, while, at the same time, inaccurate agents suffer penalties. We assume that agents who register with the Mechanism Design agent, are informed of the method by which payments are calculated and they have to agree to comply with it. The Mechanism Design agent contains a Mechanism Design Component module which contains the algorithmic components to calculate the various payments and repository modules

to store the expected energy profiles of the various producers and consumers. It communicates with the Electricity Producer and Consumer agents, the Charging Station agents, the Electricity Imbalance agent, and the Station Recommender agent.

Electricity Producer Agent: The goal of an Electricity Producer agent is to predict and periodically report the expected energy output of the energy generators it corresponds to, as well as the confidence of such predictions. These reports are then used to plan the aggregate consumption patterns, possibly by introducing variable prices for consumption, according to if renewable energy is abundant or in shortage. This can lead to significant exploitation of the renewable energy produced, as without coordination, this energy may have been left unexploited. An Electricity Producer agent contains a Preference Elicitation module which is responsible to predict the electricity production of the producer over a specific planning horizon (e.g. 24 hours ahead) and calculate its confidence for that prediction, and a Financial Management module which is responsible for the payments and transactions of the agent. This agent communicates with the Electricity Imbalance agent and the Mechanism Design agent.

Electricity Consumer Agent: The goal of this agent is to predict and periodically report the expected energy consumption of the consumer it represents, as well as the confidence of such predictions. An Electricity Consumer agent, contains a Preference Elicitation module which is responsible to predict the electricity consumption of the consumer over a specific planning horizon (e.g. 24 hours ahead) and calculate its confidence for the prediction, and a Financial Management module which is responsible for the payments and transactions of the agent. This agent communicates with the Electricity Imbalance agent and the Mechanism Design agent.

5.3 Communication Interfaces

The agents of the architecture, communicate using FIPA-ACL messages as they were presented in section 3.4. The connections that appear in Figure 5.1 and connect the agents of our architecture indicate which agents communicate with each other. The arrows of these connections denote the order in which these messages are exchanged. A connection with two solid arrowheads, indicates that both the agents can initiate the communication. An example of such a communication is the connection *Negotiation* between an Electric Vehicle agent and a Charging Station agent. A connection with a solid arrowhead and a simple arrowhead indicates that the side from which the solid arrowhead leaves, initiates the communication requesting complex information, and waits for a response containing this information. An example of such a communication is the connection *Request Station Recommendation* between an EV agent and the Station

Recommender agent. The connections with only one arrowhead, indicates communication where the side from which the arrowhead leaves, initiates the communication, while the other side sends a simple acknowledgement to inform the sender about a result. An example of such a communication is the connection *Report Expected Production* between the Electricity Producer agent and Electricity Imbalance agent.

In our architecture, the aforementioned connections represent communication protocols whose details are going to be described in Section 6.1 where we present the inter-agent control of our system. Here, we will present in high level the goals of these protocols:

- *Request Charging Recommendations*: This protocol is used from the Electric Vehicle agents to request charging recommendations from the Station Recommender agent.
- *Reserve Charging Slot*: This protocol is used from the Electric Vehicle agents to request a reservation to particular charging slot of a charging station.
- *Negotiation*: This protocol is used both from Electric Vehicle agents and Charging Station agents in order to negotiate the details of an already existing charging reservation to a particular charging slot of the station.
- *Request Imbalance*: This protocol is used from all agents who want to get informed about the electricity imbalance of the grid over a predefined planning horizon (e.g. 24 hours ahead).
- *Request Prices*: This protocol is used from all agents who want to get informed about the prices of electricity consumption over a predefined planning horizon.
- *Report Expected Production*: This protocol is used from the Electricity Production agents to report their expected electricity production over a predefined planning horizon.
- *Report Expected Consumption*: This protocol is used from the Electricity Consumption agents to report their expected electricity consumption over a predefined planning horizon.
- *Authenticate Recommendation*: This protocol is used from Charging Station agents to verify that a charging recommendation made to a particular Electric Vehicle agent is genuine.
- *Report Charging Schedule*: This protocol is used from Charging Station agents to report their charging schedule over a predefined planning horizon.

- *Report Station Availability*: This protocol is used from Charging Station agents to report their updated availability regarding their charging slots.
- *Report Confidence*: This protocol is used from all the agents who want to report their confidence to their expected energy profile over a predefined planning horizon. These profiles can be the expected production, expected production or the charging schedule of the respective agents.
- *Register Charging Station*: This protocol is used from the Charging Station agents in order to register with the all the service agents Electricity Imbalance agent, Mechanism Design agent and Station Recommender agent which need their profile and type in order to consider them in the their activities. We do not illustrate this protocol in 5.1 to increase readability.
- *Register Electricity Producer*: This protocol is used from the Electricity Producer agents in order to register with the service agents Electricity Imbalance agent and Mechanism Design agent which need their profile and type in order to consider them in the their activities. We do not illustrate this protocol in 5.1 to increase readability.
- *Register Electricity Consumer*: The same as *Register Electricity Producer* protocol.
- *Payments*: This protocol is used from all the agents who perform monetary transactions (e.g. payments for electricity consumption, payments for electricity production). This protocol, as already mentioned, in out of the scope of this work.

5.4 V2G/G2V Domain Ontology

The FIPA communication model is based on the assumption that communicating agents share an ontology of communication defining speech acts and protocols. In order to have fruitful communication, agents must also share an ontology of their domain of application, thus, we developed a simple ontology domain for the V2G/G2V domain (see Figure 5.2). This ontology is used by the agents in order to form the content of the messages they exchange as the shared understanding and the reduction of conceptual and terminological confusion are essential for successful communication. A detailed description of the concepts of the ontology in alphabetical order is the following:

Battery: This concept describes the battery which is equipped to an EV. It contains the capacity of the battery, which is the maximum energy it can store; the current State of Charge (SOC) of the battery, which is the amount of energy it has currently stored; the State of Health (SOH), which represents the condition of the battery compared to

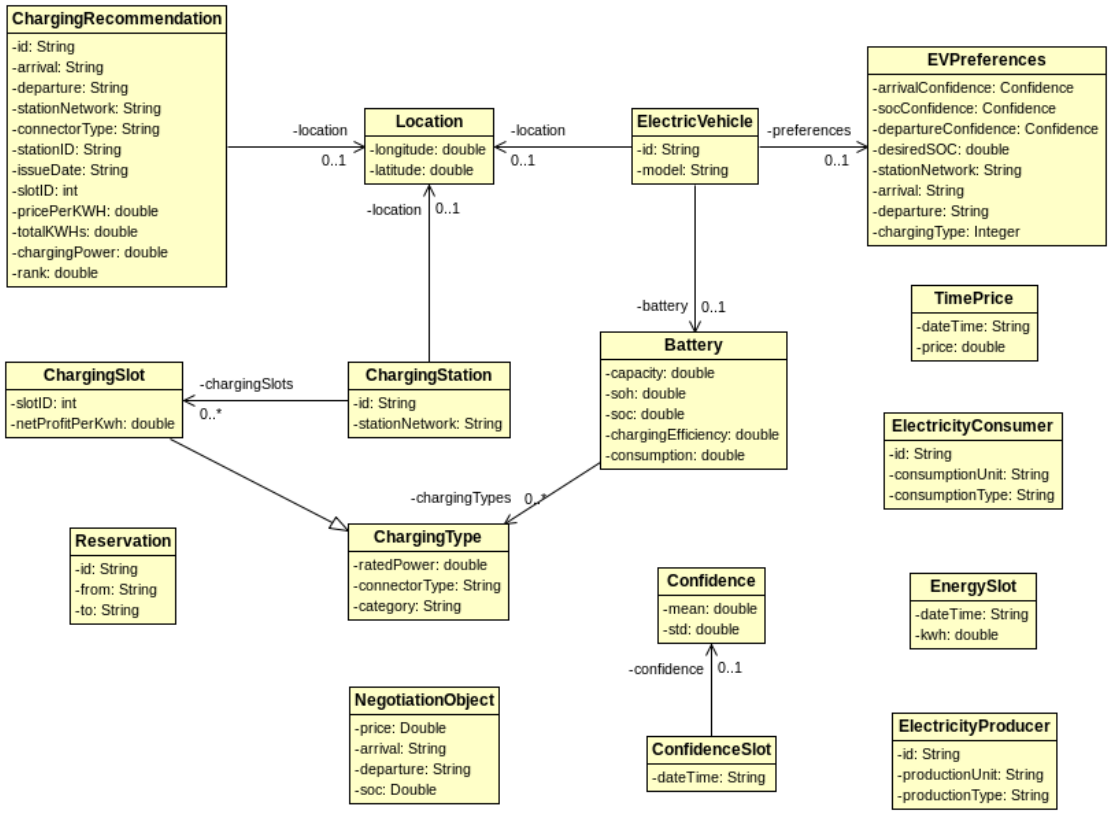

FIGURE 5.2: The concepts of the V2G/G2V Ontology given as a UML class diagram.

its ideal conditions expressed as percentage; the charging efficiency, which denotes the amount of power that the battery receives compared to the nominal value of the input power; the energy consumption (kwh/100km) and a list of charging types.

Charging Recommendation: This concept represents a charging recommendation that is made to an EV given its type (i.e. battery state, charging preferences). It contains information about the recommended arrival and departure to and from the charging station, details about the station itself (e.g, station id, charging slot id, location), as well as, the price for charging at a particular station during the period mentioned before, the amount of energy that the vehicle will receive and the rate of charging. Finally, the recommendation system assigns to each recommendation made a rank, that is, the likelihood that the EV will "like" a particular recommendation.

Charging Slot: A charging station can contain one or more charging slots. These slots are the ports where the EVs connect and derive power to charge their batteries. Each slot is coupled with a parking slot that we assume its in front or near it. At a given charging station, different charging slots can have different payment tariffs for each electricity unit consumed.

Charging Station: This concept describes an EV charging station. A charging station contains the identifier of the station, its location, the network of stations that she belongs,

as well as the charging slots available in this particular charging station.

Charging Type: EVs can be equipped with one or more inlets which allow different maximum power inputs for their battery to charge. The charging power determines the actual charging time of the battery. Given these, this concept contains a field that denotes the maximum rated power for charging and discharging the battery, the inlet type, and a label to categorize the type charging (e.g. Normal, Fast, Rapid).

Confidence: This is a concept that contains information about the confidence of an entity regarding its expected goals. The confidence is expressed using two values; the mean and standard deviation and indicate the deviation from the reported expected energy profile.

Confidence Slot: This concept associates a Confidence object to a specific date and time. It is used to declare our confidence over a specific time interval. Its parameters are a Confidence concept as described above, and a date-time value with format `YYYY-MM-DD HH:MI:Sec`.

Electricity Consumer: This concept contains all the information which describe an Electricity Consumer. It includes the identifier of the consumer, the units of measurement consumption (e.g. kWh) and its type (e.g. Residential, Industrial).

Electricity Producer: This concept contains all the information which describe an Electricity Producer. It includes the identifier of the producer, the units of measurement of production and its type (e.g. PV Panel, Wind Turbine, VPP).

Electric Vehicle: The Electric Vehicle concept contains all these parameters which describe a real-world EV. These parameters include the battery of the vehicle, the charging preferences of the driver, its current location, as well as the model and an identifier to uniquely recognize it.

Energy Slot: This concept contains information about the amount of energy which is expected to be available at a specific time interval. It contains a parameter about the amount of energy (kWh) available and a data-time parameter with format `YYYY-MM-DD HH:MI:Sec`

EV Preferences: This concept, contains a set of parameters which capture the current desires and needs of the EV driver concerning the charging of his vehicle. These parameters include the expected arrival and departure to and from the charging station expressed as a `YYYY-MM-DD HH:MI:Sec` formatted String, the desired SOC at the departure time, the preferred network of charging stations, the preferred charging type from those which the vehicle supports and the confidence to his preferences.

Location: This concept contains a set of geographic coordinates (latitude and longitude) that uniquely determine the position of an object in the world.

Negotiation Object: In our system, we allow negotiations between EVs and charging stations. The negotiation concerns a set of variables that are available for negotiation and are denoted using this object. These variables include the arrival and the departure to and from a particular charging station, the amount of energy that the EV is going to receive, as well as the price charged per energy unit (kWhs).

Reservation: This concept contains information about a reservation made to charging slot of a charging station. Its parameters are the period of the reservation expressed as a set of date-time values (`YYYY-MM-DD HH:MI:Sec`), as well as a unique reservation identifier.

Time Price: This concept contains information about the price of each unit of energy available at a specific time interval. Its parameters are a price, and a date-time value with format `YYYY-MM-DD HH:MI:Sec`.

Chapter 6

System Design and Implementation

In this chapter, we present the design and implementation of the V2G/G2V architecture presented in Chapter 5. The communication protocols (i.e. inter-agent control) that we presented there, and each agent's internal operation (i.e. intra-agent control), are described here in much more detail. We define two new design patterns, that on the one hand allow the developers to re-use the protocol parts and logic defined in the framework, and on the other hand to customize key agent functionalities or capabilities according to their needs/goals. For our implementation, we utilized JADE (see Section 3.6), a framework which respects the standards set by the FIPA and provides a friendly and comprehensive Java API for agent development and management.

The ontology presented in Section 5.4, here is going to be enriched with additional entities that connect the various concepts presented there. In Section 3.6, we mentioned that an ontology in JADE is a collection of *Concepts*, *Predicates* and *AgentActions* which represent the definitions and relationships of these definitions that describe a particular subject domain (in our case, the V2G/G2V domain). The *Concepts* of our ontology are already presented in Section 5.4 and in this section we will present the *Predicates* and *AgentActions* of ontology through the inter-agent control.

The modeling of both the intra- and inter-agent controls of our system follows the ASEME paradigm and rules (see 3.7), and are illustrated using statecharts, which are part of the Unified Modeling Language (UML). Then, these statecharts are transformed into the appropriate JADE communication logic and behaviours, using a similar methodology to [61] and [40]. All statecharts presented in this chapter were designed using YAKINDU [1].

[1] https://www.itemis.com/en/yakindu/state-machine/

6.1 Inter-Agent Control

Inter-agent control refers to the communication protocols that are used by the various agents to communicate. Each protocol is defined by *(i)* its purpose, which is a textual description of the nature of the interaction, *(ii)* the initiator, which is the role(s) responsible for initiating the communication protocol, *(iii)* the responder, which is the role(s) with which the initiator interacts, and *(iv)* the protocol inputs and outputs depending on the role. The two participating roles are shown in the grey area of each statechart that represents parallel (orthogonal) executing components e.g. see Figure 6.1. Having this definition, in the general case, a concrete agent can inherit the part of the statechart corresponding to the role it wants to play in its intra-agent control model.

In this section, we define 13 communication protocols for inter-agent communication. The roles that participate in these protocols are played by agents of our V2G/G2V architecture. Depending on their goals/needs, some agents play many different roles, while others only specific ones. Despite the fact that we specify at each protocol description which agents of our architecture utilize it, this does not imply that these protocols are agent specific. On the contrary, they can be utilized by different agents which possibly are not part of our current architecture.

For the implementation of these protocols, we used JADE's `FSMBehaviour`, which is a composite behaviour with Finite State Machine (FSM) based children scheduling, and *SequentialBehaviour* which is a composite behaviour with sequential children scheduling. The various inputs and outputs of each protocol are accessed through JADE's `DataStore`, a data structure that all the behaviours of a particular role of the protocol have to share (see Section 3.6).

The different color (light yellow) of a state for specific communication protocol statecharts, denotes the state (in our implementation correspond to JADE behaviours) that is not part of the protocol package but is expected as a parameter in the instantiation of protocol role. These states contain custom functionality that is provided by the developer using an appropriate design pattern (see Section 6.3). Thus, each protocol becomes open in the sense that the developer can program the agent to handle the received messages according to her own goals and needs.

Note, the definition of the communication protocols presented in this section, do not impose a specific way of interpreting the exchanged messages or a technology of exchanging them. This means that they could be developed with alternative frameworks and programming languages than these we used in this thesis (i.e. JADE and Java).

6.1.1 Charging Recommendation Protocol

This protocol is used by EV agents in order to request charging station recommendations from the Station Recommender agent and it is illustrated in Figure 6.1. Here, an EV agent plays the *Electric Vehicle* role (*ev*, the left side of the statechart), while the Station Recommender agent plays the role Station Recommender (*srec*, right side of the statechart).

For the *ev* role, the protocol starts by sending a message to request charging recommendations. The event of sending this message, which must have the Request performative, the *ev* as sender, the *srec* as receiver and the message content being the *RequestChargingRecommendations* as the Request(ev, srec, *RequestChargingRecommendations*) event suggests, causes another transition, targeting the *ReceiveRecommendations* state. The same transition executes the action t1 = TIMEOUT that sets the timer t1 to timeout in TIMEOUT milliseconds (TIMEOUT is a configurable variable). The receipt of an inter-agent message with performative Inform and content *ChargingRecommendations*, or Failure and content *Problem*, or the timeout of the timer t1 can cause the final transition to terminate this protocol.

For the *srec* role, the protocol starts at the *ReceiveRecommendationsRequest* state that contains an activity that waits for a request message by an EV. When a request is received (the *srec* transition to the next state has the same expression with the *ev* transition from the "*SendRecommendationsRequest*" to the *ReceiveRecommendations*), she gets to the next state *CalculateRecommendations* (is not implemented in the protocol package) which contains the charging recommendations calculation activity, and sets her timer (t2) to timeout in TIMEOUT milliseconds. From this state, *srec* has two transition options for making a transition. The first option is that a charging recommendations result is successfully calculated, while the second is that the t2 timer timeouts. In the first case, there is a transition to the *SendChargingRecommendations*, while in the second case the protocol terminates for this role. From the *SendResponseMessage* state *srec* sends an Inform with content *ChargingRecommendations* or a Failure message with content *Problem* to her counterpart and the protocol for this role terminates.

- *RequestChargingRecommendations* is an *AgentAction* with properties:
  - *ElectricVehicle*: Ontology *Concept* (see Section 5.4)
- *ChargingRecommendations* is a *Predicate* with properties:
  - *List of ChargingRecommendation*: Ontology *Concept*
- *Problem* is a *Predicate* with properties:

– *String*: *Primitive* (The type of the problem occurred in human-readable format, this is Primitive in our ontology as presented in Section 3.6)

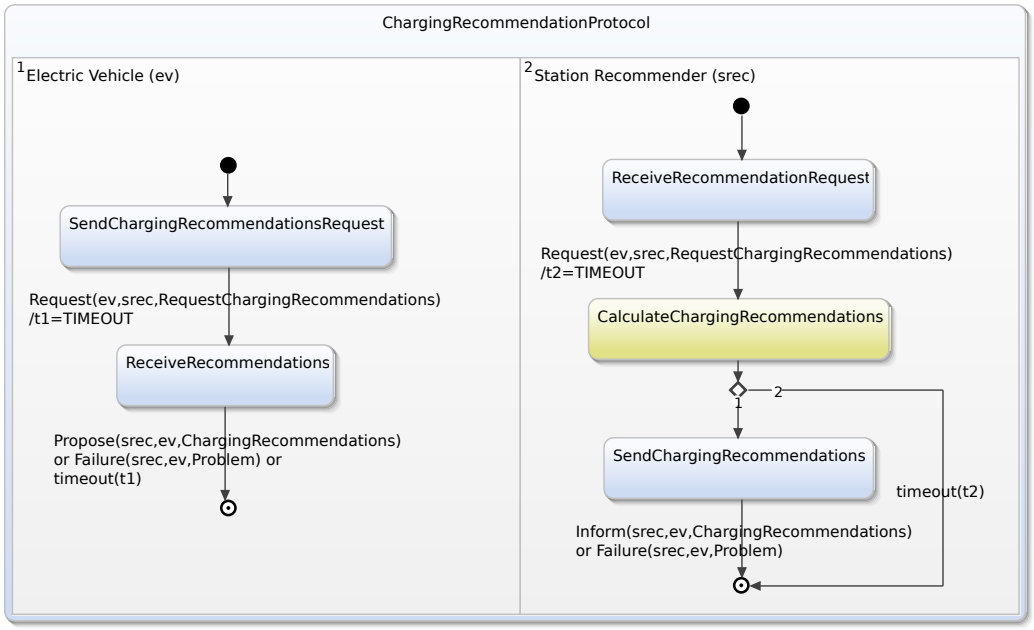

FIGURE 6.1: The model of the Charging Recommendation Protocol.

6.1.2 Charging Station Reservation Protocol

This protocol is used from EV agents to make a charging reservation to a specific charging slot of a charging station. The request is sent to the corresponding Charging Station agent. Note that in order to make a charging reservation, the EV agent should have received charging recommendations. The communication protocol is depicted in Figure 6.2. The execution logic of this protocol is identical with that of the *Charging Recommendation Protocol*. The only difference is the content of the various messages.

- *ReserveChargingSlot* is an *AgentAction* with properties:
  - *ChargingRecommendation*: Ontology *Concept* (see Section 5.4)
  - *ElectricVehicle*: Ontology *Concept*
- Problem is a *Predicate* with properties:
  - *String*: Ontology *Primitive*, the type of the problem occurred in a human-readable format.
- Done is a *Predicate* (part of JADE's `jade.content.onto.basic` package) with properties:

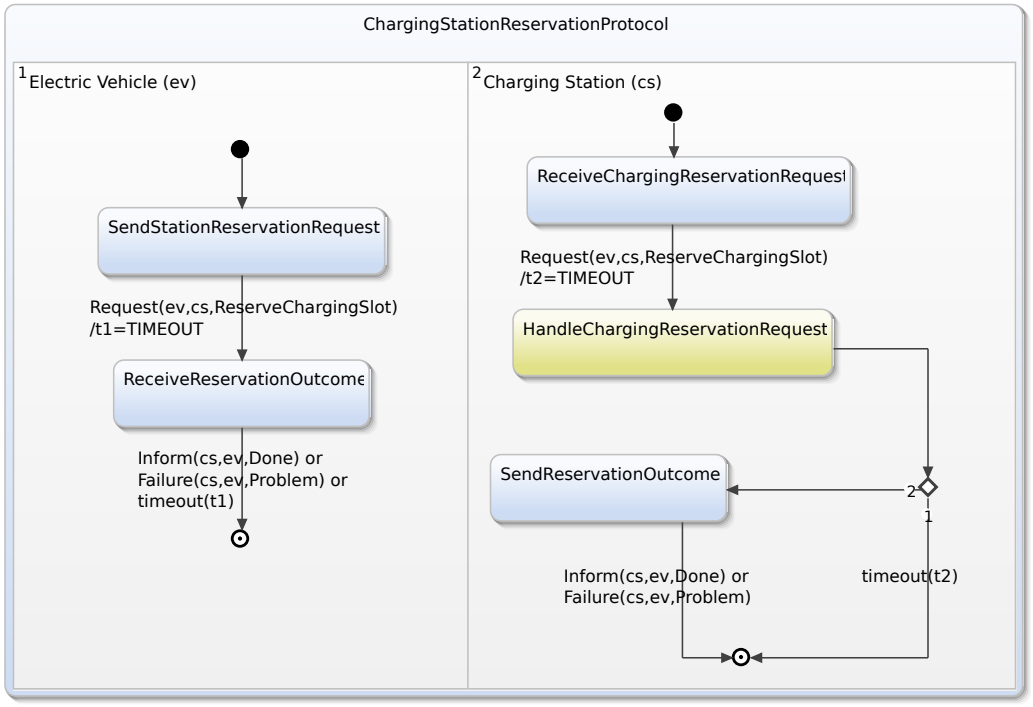

FIGURE 6.2: The model of the Charging Station Reservation Protocol.

– *ReserveChargingSlot:* Ontology *AgentAction*, if the requested action was performed successfully.

6.1.3 Negotiation Protocol

This communication protocol (Figure 6.3) is used from EV and Charging Station agents to negotiate about an existing charging reservation. In this protocol, we have only one role, i.e. the *Responder* to a proposal. Both agents play the same role. The protocol starts as soon as a protocol-related message arrives. This, however, implies that one of the agents is responsible to initiate the negotiation by sending a proposal.

The state *ReceiverNegotiationMessage*, contains an activity that waits to receive a message. As soon as one such arrives, there are two possibilities: *(a)* if its performative is *Accept* or *t1* has timed-out indicating that there is no response of the counterpart indicating inability to reach a solution (the negotiation ends unsuccessfully), and then the protocol terminates for this role; *(b)* the performative is *Propose* or *Reject*, we have a transition to the *NegotiationDecisionMaking* state. In this state, the message received is processed and a result is reached based on the negotiation decision-making algorithm. This algorithm can be inserted with the functionality design pattern (its implementation of the action method just calls the *Template* method).

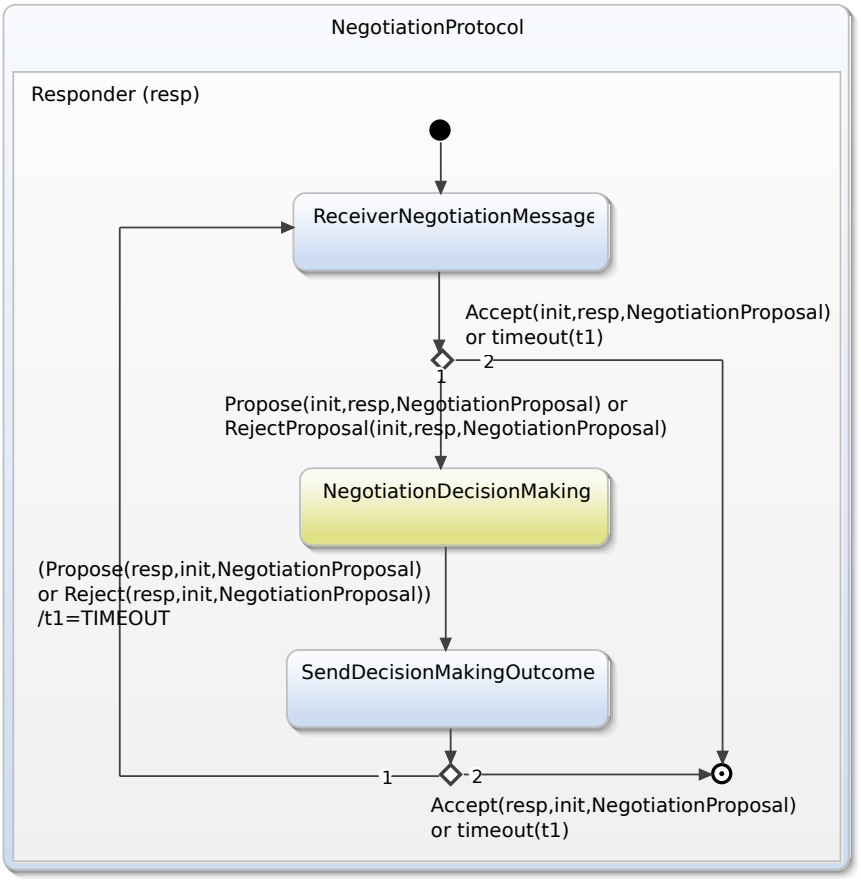

FIGURE 6.3: The model of the Charging Negotiation Protocol.

- *NegotiationProposal* is a *Predicate* with properties:
  - *NegotiationObject*: Ontology *Concept* (see Section 5.4)

6.1.4 Charging Station Registration Protocol

This protocol is used from the Charging Station agents, which plays the *Charging Station* role, in order to register with the various service agents of our system. Currently, these agents are the Mechanism Design agent, the Electricity Imbalance agent and the Station Recommender agent, and all play the *Service Provider* role. A Charging Station agent should be registered with these agents in order to facilitate better individual operation, as well as the operation of the grid. The communication protocol is depicted in Figure 6.5.

- *RegisterChargingStation* is an *AgentAction* with properties:
  - *ChargingStation:* Ontology *Concept* (see Section 5.4)

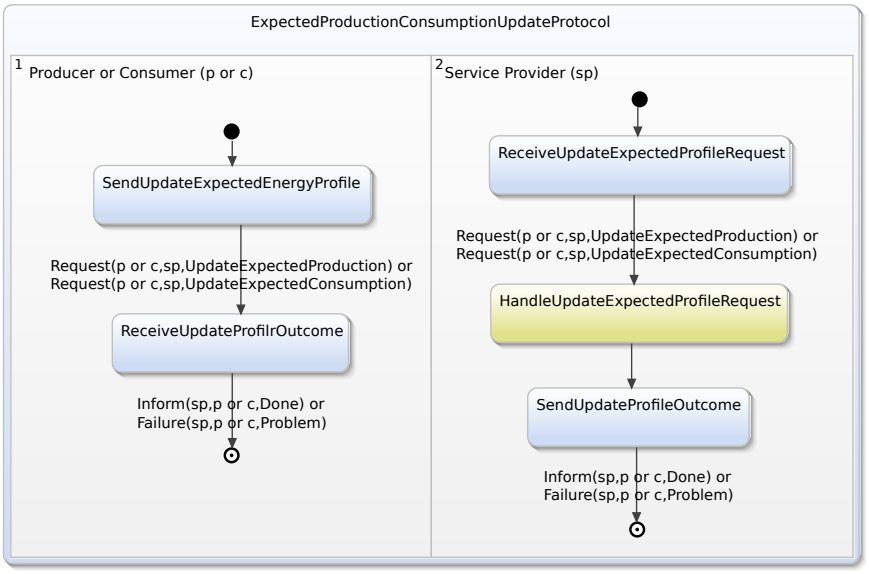

FIGURE 6.4: The model of the Update Expected Production/Consumption Protocol.

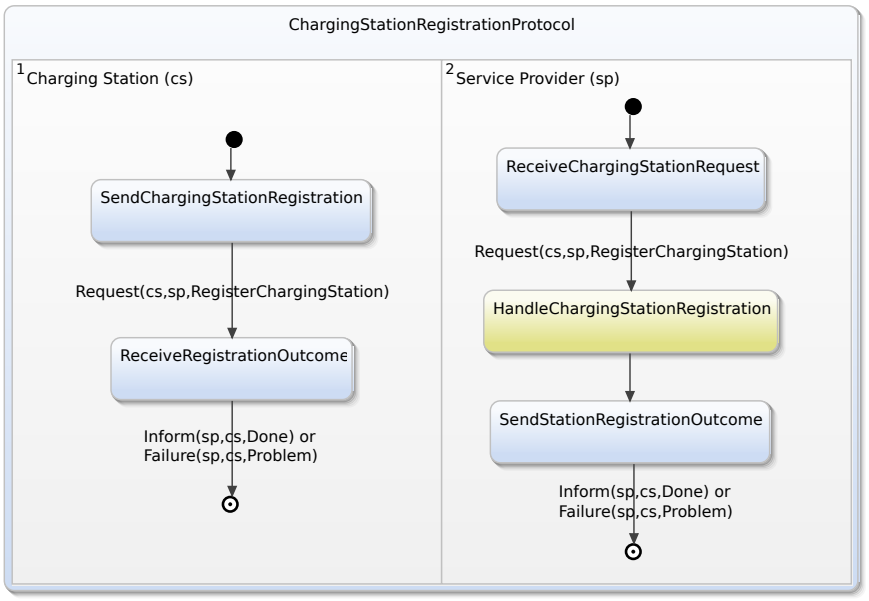

FIGURE 6.5: The model of the Charging Station Registration Protocol.

- Problem is a *Predicate* with properties:
  - *String*: Ontology *Primitive*, the type of the problem occurred in a human-readable format.
- Done is a *Predicate* with properties:
  - *RegisterChargingStation:* Ontology *AgentAction*, if the requested action was performed successfully.

6.1.5 Authenticate Recommendation Protocol

This protocol is shown in Figure 6.6 and it is used from Charging Station agents, in order to validate that a recommendation made to a particular EV agent is genuine or not. The Station Recommender agent can make personalized recommendations to EVs that may contain special prices and offers with an aim to incentive specific charging behaviour and/or better distribution of the EVs to the charging stations. Thus, to protect the recommendation process from malicious EV agents who aim to modify the details of a received recommendation before contracting a charging station, we provide to Charging Station agents (or other agents who want to play one of the roles) this protocol. The *Charging Station* role forms a message with the *Query-If* performative and receives either a message with the Confirm or *Dis-confirm* performative.

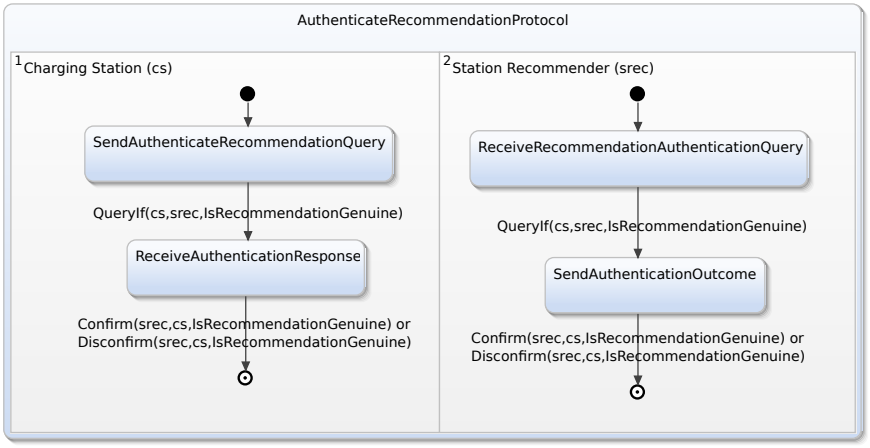

FIGURE 6.6: The model of the Authenticate Recommendation Protocol.

- *IsRecommendationGenuine* is a *Predicate* with properties:
  - *ChargingRecommendation*: Ontology *Concept* (see Section 5.4)
  - *String:* Ontology *Privitive*, the ID of the vehicle that received the recommendation

6.1.6 Electricity Prices Request Protocol

This communication protocol is used from the Station Recommender agent, in order to learn the current prices of electricity over a predefined planning horizon, and utilize this information for its own goals and needs. This agent plays the *Service User* role of the protocol while the Mechanism Design agent plays the *Mechanism Design Service* role. The latter agent always waits to receive requests regarding the electricity prices, and when such a request is received, it calculates the current prices and sends the results.

The statechart of this protocol is shown in Figure 6.7. The message sent by the agent who plays the role *Service User*, has null content, as the semantics of the message (i.e. the protocol name, the ontology, and the content language), provide enough information to the *Mechanism Design Service* role to identify the purpose of the message.

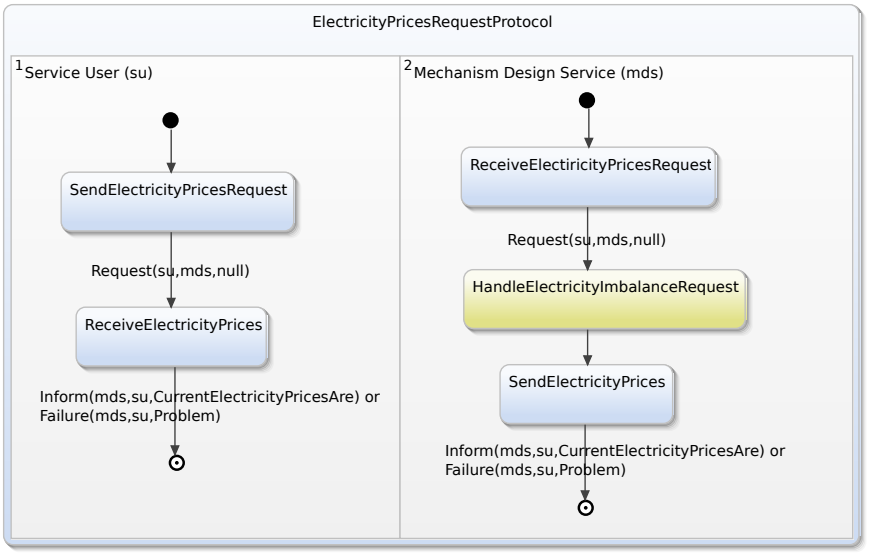

FIGURE 6.7: The model of the Electricity Prices Request Protocol.

- *CurrentElectricityPricesAre* is an *Predicate* with properties:
  - A list of *EnergySlot*: Ontology *Concept* (see Section 5.4)
- Problem is a *Predicate* with properties:
  - *String*: Ontology *Primitive*, the type of the problem occurred in a human-readable format.

6.1.7 Electricity Imbalance Request Protocol

This communication protocol is used from the Station Recommender agent, the Mechanism Design agent and the Charging Station agent, in order to learn the electricity imbalance of the grid over a predefined planning horizon, and utilize this information for their own goals and needs. These agents play the *Service User* role of the protocol while the Electricity Imbalance agent plays the *Electricity Imbalance* role of the protocol. The latter agent always waits to receive requests regarding the electricity imbalance of the grid, and when such a request is received, it processes and sends the results. The statechart of this protocol is shown in Figure 6.8. The message sent by the agent who plays the role *Service User*, has null content, as the semantics of the message (i.e. the

protocol name, the ontology, and the content language), provide enough information to the *Electricity Imbalance Service* role to identify the purpose of the message.

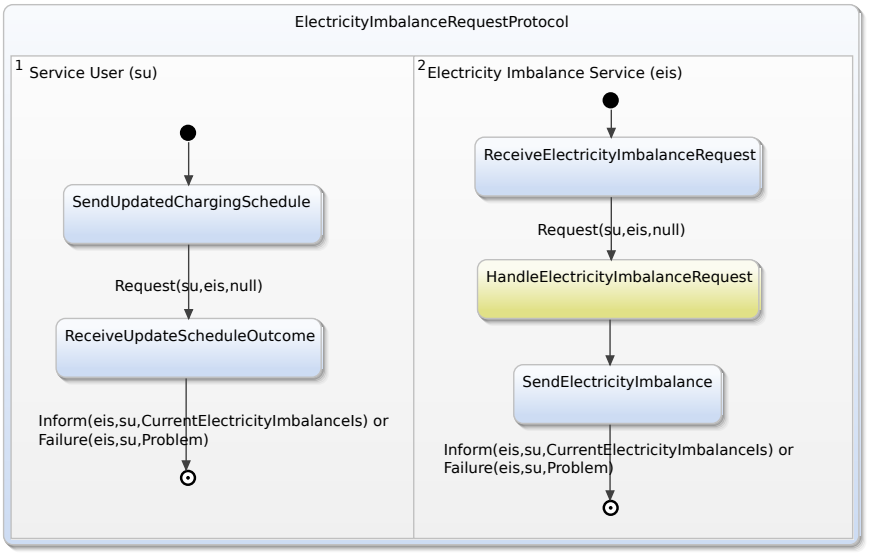

FIGURE 6.8: The model of the Electricity Imbalance Request Protocol.

- *CurrentElectricityImbalanceIs* is an *Predicate* with properties:
  - A list of *EnergySlot*: Ontology *Concept* (see Section 5.4)
- Problem is a *Predicate* with properties:
  - *String*: Ontology *Primitive*, the type of the problem occurred in a human-readable format.

6.1.8 Charging Station Update Schedule Protocol

This communication protocol is used from the Charging Station agents, in order to update their charging schedule over a predefined planning horizon by playing the *Charging Station* role of the protocol. The *Service Provider* role, which receives the charging schedule updates, handles them and send results, is played from the Mechanism Design agent and Electricity Imbalance agent. An illustration of the statecharts of this communication protocol is shown in Figure 6.9.

- *UpdateStationSchedule* is an *AgentAction* with properties:
  - *String*: Ontology *Primitive*, the ID of a charging station
  - List of *EnergySlot*, Ontology *Concept* (see Section 5.4)

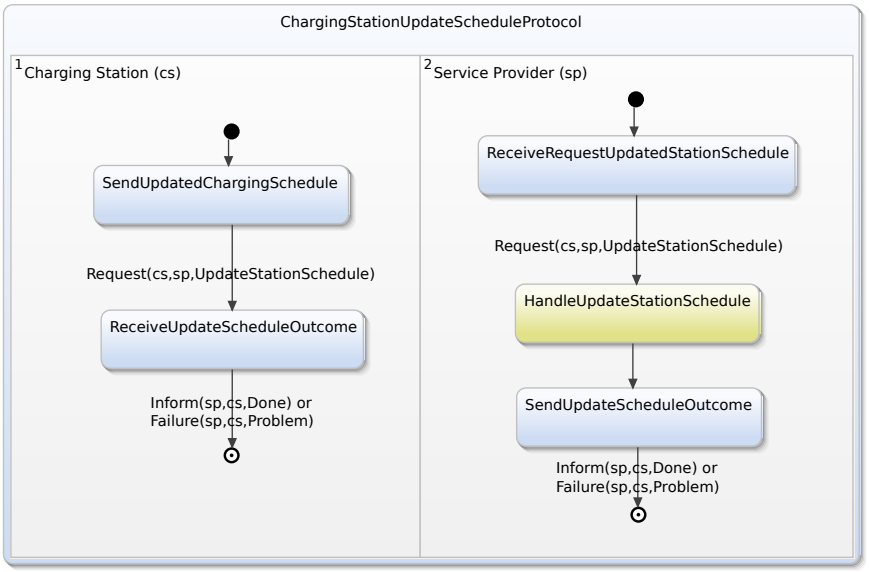

FIGURE 6.9: The model of the Charging Station Update Schedule Protocol.

- Problem is a *Predicate* with properties:
  - *String*: Ontology *Primitive*, the type of the problem occurred in a human-readable format.
- Done is a *Predicate* with properties:
  - *UpdateStationSchedule:* Ontology *AgentAction*, if the requested action was performed successfully.

6.1.9 Producer Consumer Registration Protocol

This communication protocol is used from Electricity Producer and Consumer agents in order to register with the various service agents of our system. Currently, these agents are the Mechanism Design agent and the Electricity Imbalance agent which play the *Service Provider* role of the protocol. The protocol is illustrated in Figure 6.10.

- *RegisterElectricityProducer* is an *AgentAction* with properties:
  - *ElectricityProducer*: Ontology *Concept* (see Section 5.4)
- *RegisterElectricityConsumer* is an *AgentAction* with properties:
  - *ElectricityConsumer*: Ontology *Concept*
- Problem is a *Predicate* with properties:

– *String*: Ontology *Primitive*, the type of the problem occurred in a human-readable format.

- Done is a *Predicate* with properties:
  - *RegisterElectricityProducer or RegisterElectricityConsumer:* Ontology *AgentAction*, if the requested action was performed successfully.

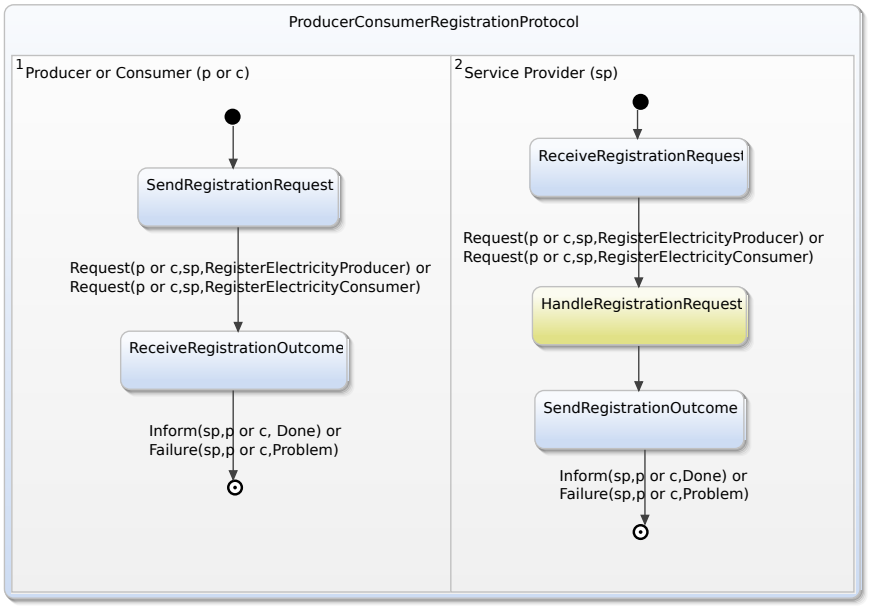

FIGURE 6.10: The model of the Producer/Consumer Registration Protocol.

6.1.10 Update Expected Production/Consumption Protocol

The Electricity Producer and Consumer agents which are registered with the various services of our system should periodically send updates for their expected levels of production or consumption over a predefined planning horizon. This is achieved using the protocol illustrated in Figure 6.4. Currently, the agents which play the *Service Provider* role of this protocol are the Mechanism Design agent and the Electricity Imbalance agent.

- *UpdateExpectedProduction* with properties:
  - List of *EnergySlot*: Ontology *Concept* (see Section 5.4)
- *UpdateExpectedConsumption*
  - List of *EnergySlot*: Ontology *Concept*
- Problem is a *Predicate* with properties:

– *String*: Ontology *Primitive*, the type of the problem occurred in a human-readable format.

- Done is a *Predicate* with properties:
  - *UpdateExpectedProduction or UpdateExpectedConsumption:* Ontology *AgentAction*, if the requested action was performed successfully.

6.1.11 Update Energy Profile Confidence Protocol

This protocol is used from Charging Station agents, Electricity Consumer agents and Electricity Producer agents, in order to update their confidence of their reported expected energy profiles over a predefined planning horizon. These agents play the *Service User* role of the protocol. The Mechanism Design agent and the Electricity Imbalance agent, play the *Service Provider* role, and wait to receive update requests from the aforementioned agents. The communication protocol is depicted in Figure 6.11.

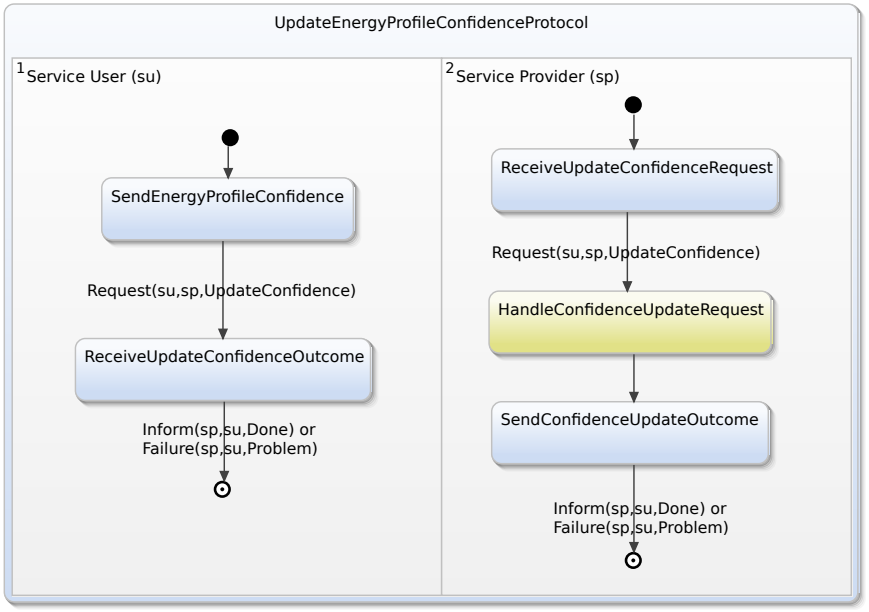

FIGURE 6.11: The model of the Update Station Availability Protocol.

- *UpdateConfidence* is an *AgentAction* with properties:
  - List of *ConfidenceSlot*: Ontology *Concept*, (see Section 5.4)
- Problem is a *Predicate* with properties:
  - *String*: Ontology *Primitive*, the type of the problem occurred in a human-readable format.

- Done is a *Predicate* with properties:
  - *UpdateConfidence:* Ontology *AgentAction*, if the requested action was performed successfully.

6.1.12 Update Station Availability Protocol

This communication protocol is used from the Charging Station agents (which play the *Charging Station* role) in order to update their charging slot availability. The Station Recommender agent, plays the *Station Recommender* role and waits to receive update availability requests. When such a request is received, it handles it the results. of this protocol is depicted in Figure 6.12.

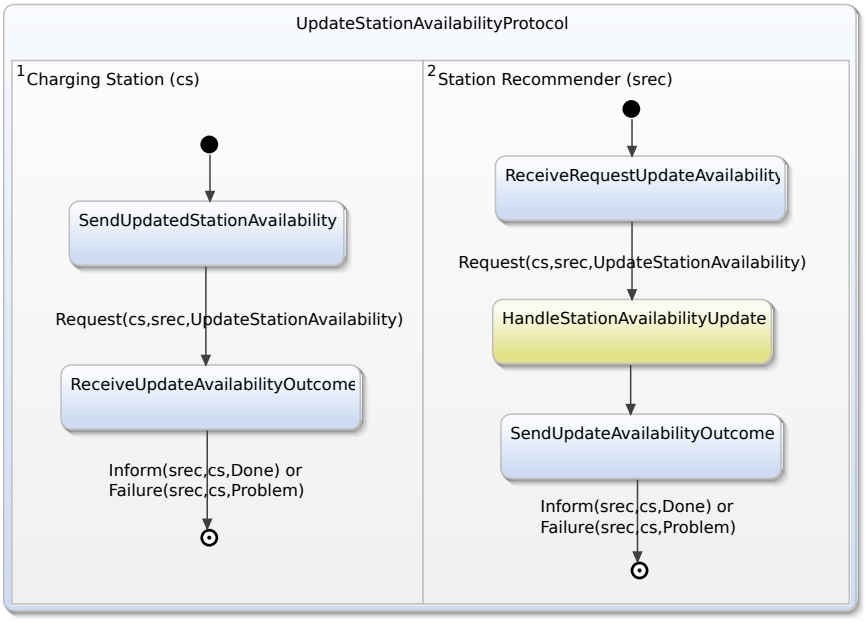

FIGURE 6.12: The model of the Update Station Availability Protocol.

- *UpdateStationAvailability* is an *AgentAction* with properties:
  - *String:* Ontology *Primitive*, the ID of a charging station
  - *Integer:* Ontology *Primitive*, the ID of a charging slot
  - *Reservation:* Ontology *Concept* (see Section 5.4)
- Problem is a *Predicate* with properties:
  - *String*: Ontology *Primitive*, the type of the problem occurred in a human-readable format.
- Done is a *Predicate* with properties:

– *UpdateStationAvailability:* Ontology *AgentAction*, if the requested action was performed successfully.

6.1.13 Time Synchronization Protocol

This protocol (Figure 6.13) is used from all the agents which want to synchronize their internal clocks. Currently, in our system, all the agents should be synchronized, thus, they all use this protocol playing the Service User role. First, the initiator of the protocol requests from the DF agent a list with all the agents which offer a time service. In our system, we assume the existence of a single Time agent. Then, the initiator sends a request which has no content. The reason for this is that the semantics of the message (i.e. the protocol name and the performative) have enough information for the Time agent to understand the intention of the message. When the Time agent receives the message, it responds with the synchronization information.

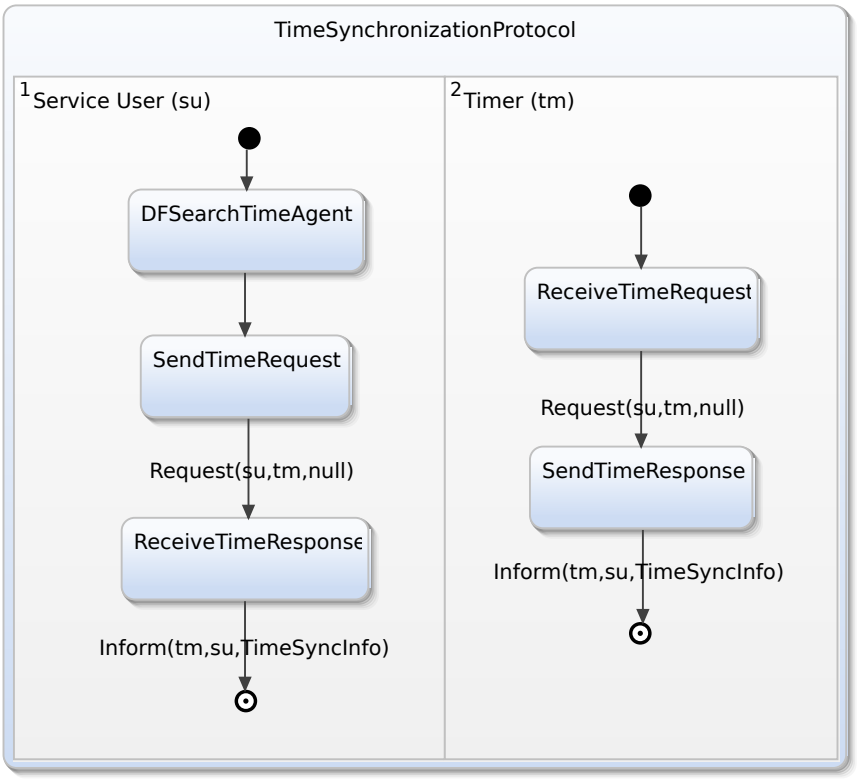

FIGURE 6.13: The model of the Time Synchronisation Protocol.

- *TimeSyncInfo* is a *Predicate* with with properties:
  - *Long*: Ontology *Primitive*, the real date-time when the execution of the system started expressed in milliseconds

- *Long*: Ontology *Primitive*, the real time step of the system expressed in milliseconds
- *String*: Ontology *Primitive*, the simulation date-time when the execution of the system started format `YYYY-MM-DD HH:MI:Sec`
- *Integer*: Ontology *Primitive*, the simulation time step of the system expressed in minutes

6.2 Intra-Agent Control

Intra-agent control refers to each agent's internal data receiving, processing and management model. Each agent of our architecture, depending on its goals, uses a composition of the communication protocols presented in Section 6.1. Through these protocols, it receives information essential for its decision making and/or acts by sending information messages to other agents with purpose of influencing them and modify the environment of operation. Furthermore, each agent can contain private states, which contain agent-specific functionalities that are inserted using specific design patterns (see Section 6.3).

Each agent maintains a collection of data structures where it stores its operational details. These data structures as already mentioned, are private for each agent, thus, other agents can not access and modify them. For each agent we summarize the most important data structures for their operation, providing the name of the data structures, which in our case corresponds to the name of the Java implementation, a description to its purpose, and the state of the intra-agent control that utilizes it or modifies it.

6.2.1 Electric Vehicle Agent

Then intra-agent control model for the Electric Vehicle agent is shown in Figure 6.15. Note that for simplicity of representation, the protocol roles that the agent realizes are shown as basic states. These can be expanded to the relevant roles in the protocols presented earlier in inter-agent control (see Section 6.1).

When it starts its operation, the agent performs initialization activities (enters the *Initialization* state). In this state, it publicizes its service (a V2G / G2V service) to other agents using the DF agent (see Section 3.3, DFRegisterService state) and then, it plays the *Service User* role of the *TimeSynconizationProtocol* in order to receive time information and synchronize its simulation clock. The purpose of the simulation clock is to make easier the experimentation on the developed prototype by accelerating the time of execution of the various agent activities while ensuring that all the agents have the

same clock readings (sec Section 6.4 for more information). Then, the agent enters both the *RecommendationReservation* and *Negotiation* rectangle components. In the *RecommendationReservation* component it makes a transition to the *DecideNextAction* basic state.

In this state, the agent makes decisions regarding the charging of the EV. There are three different possibilities for the *EV agent*: *(a)* it uses a specific algorithm to monitor and predict the battery state and driver preferences and to decide autonomously when and how to arrange the EV charging; *(b)* it gets the aforementioned information from predefined datasets;[2] and *(c)* it provides a comprehensive GUI where the user can insert her charging preferences and manually initiate the protocols. These possibilities are three different implementations for the *DecideNextAction* state activity. From this state, we have two possible transitions. The first transition is caused by the event "Initiate Recommendations-Reservation" and leads to the *RecommendationReservationControl* state, while the second transition is caused by the event "Initiate Negotiation" which, however, has a condition in order to be executed as it requires the variable t1 to be greater than the current time of the system. For now, this condition is not true, thus, the transition is not possible.

In state *RecommendationReservationControl* the EV agent, first plays the *Electric Vehicle* role of the *ChargingRecommendationProtocol* where it sends information about its battery state, preferences and location requesting for charging recommendations. When the response is received, we have two transition options. The first transition is caused by the event "No Charging Recommendations" which means exactly that, that is, no charging recommendations are currently available for the EV. However, if recommendations are available, the agent evaluates them and selects the recommendation which better serves its needs and preferences using a specific utility function (state *EvaluateRecommendations*). This utility function is inserted using the functionality design pattern (see Section 6.3).

Then, the EV agent plays the *Electric Vehicle* role of the *ReserveChargingStationProtocol* in order to communicate with the selected Charging Station agent and make a reservation to the recommended charging slot. If the reservation is successful, the *RecommendationReservationControl* state terminates. If the reservation is unsuccessful due to the fact the initial details of the selected recommendation are outdated (i.e. another EV occupied the requested charging slot), the agent requests new charging recommendations, and the aforementioned procedure is repeated. In both cases, we have a transition back to the *DecideNextAction* state where the agent determines its next action. If the

[2]This is very useful for experimentation with large agent populations.

transition to this state was triggered from the "Successful Reservation" event, the parameter t1 is set to be equal with the arranged arrival time to the selected charging station. This means that a transition from state *DecideNextAction* to *InitiateNegotiation* is now possible, thus, the EV agent can negotiation the active charging reservation with the selected Charging Station agent (optionally).

If the EV agent wants to initiate a negotiation, it just sends a proposal, and if the Charging Station agent replies, the rest of the negotiation process will be taken care by the *Negotiation* orthogonal component. There, the agent enters the *NegotiationProtocol:Responder* (using the Responder role of the respective protocol) and when the negotiation is over, it handles the results (i.e. update its data structures). The *Negotiation* orthogonal component is executed always in parallel to the *RecommendationReservation* component, as a Charging Station agent could itself initiate a negotiation at any time. The EV agent used the functionality design pattern to define the *NegotiationDecisionMaking* action of the *NegotiationProtocol*.

An example of the implementation of the functionality design pattern is illustrated in Figure 6.14. The agent state of the EV agent *EvaluateRecommendation*, has access to the implementation of the utility function from the *myAgent* reference. Thus, the agent is responsible to decide which utility function is going to use. To do this, it utilizes the *UtilityFunctionFactory* (*Factory1* of Figure 6.23), providing as input, only the class name of the utility function it wants to use. In this thesis, we have implemented three different utility functions (i.e. *Instance* subclasses) *(i) MaxEnergyRecSelector*, which selects the charging recommendation with the maximum energy, *(ii) MinDistanceRecSelector*, which selects the charging recommendation that contains the closest charging station, and *(iii) MinPriceRecSelector*, which selects the recommendation that offers the lowest charging price per kWh. The aformentioned subclasses extend the *EVUtilityFunctionTemplate* (*Template1*).

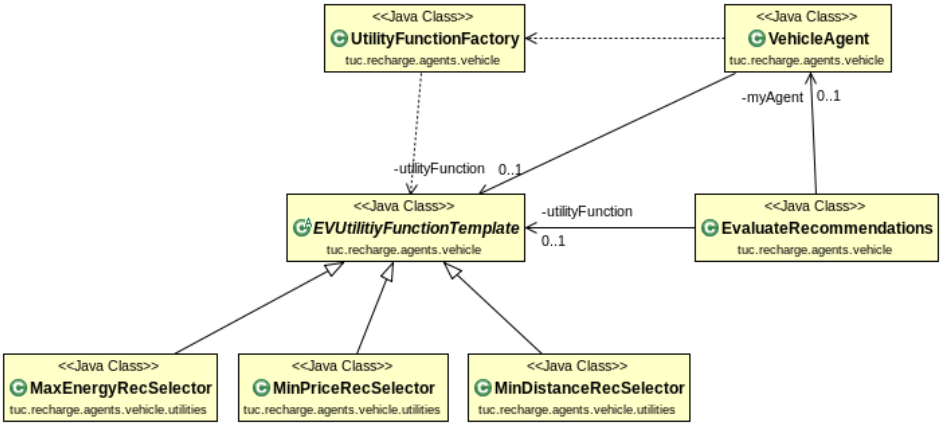

FIGURE 6.14: The impementation of the functionality design pattern for the utility function of the EV agent.

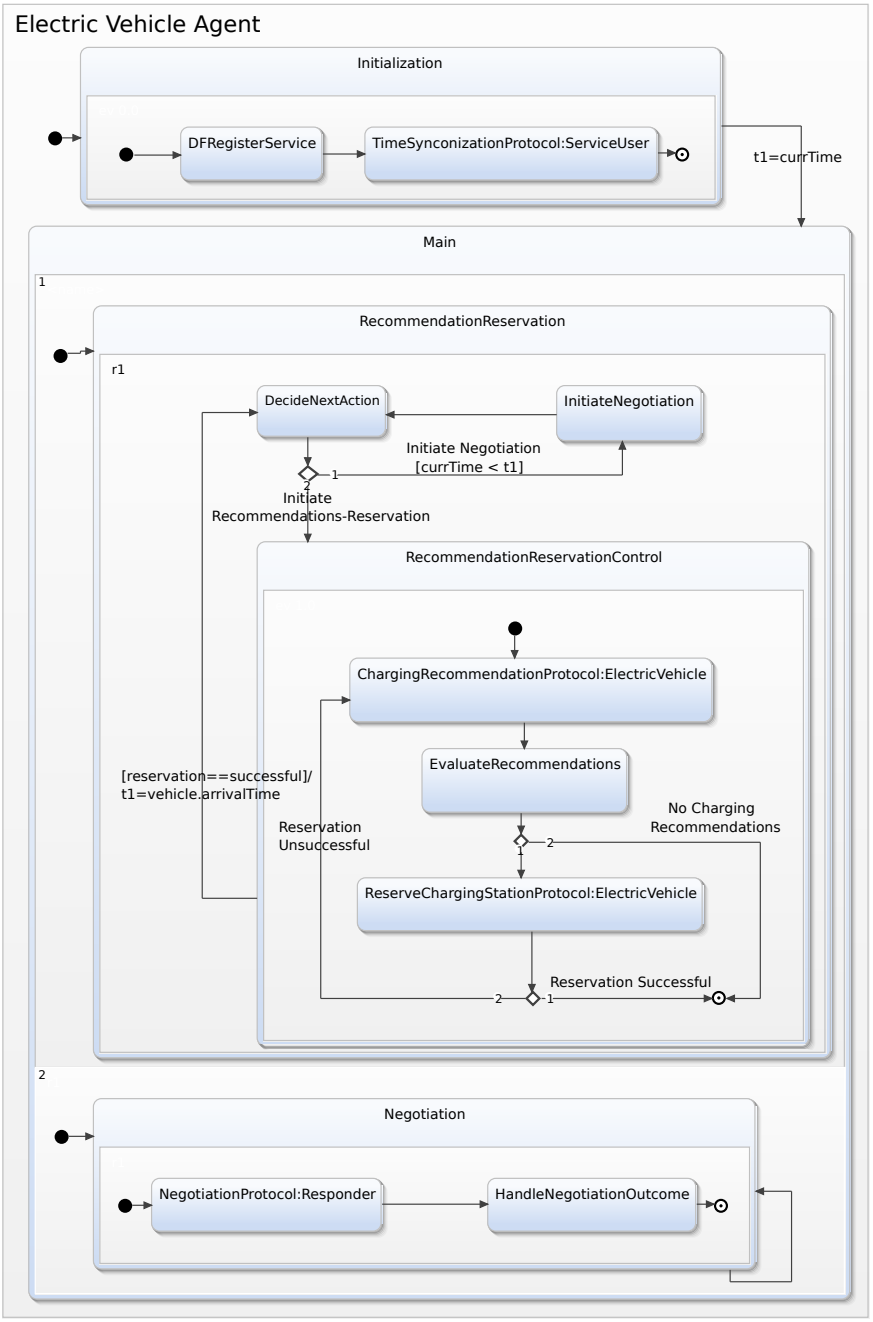

FIGURE 6.15: The intra-agent model of the Electric Vehicle agent.

6.2.2 Charging Station Agent

The intra-agent control of the Charging Station agent is depicted in Figure 6.16. The agent first executes the same initialization presented before for Electric Vehicle agent, that is, the registration of its service with the DF, and the synchronization of its simulation clock. Then, the agent enters both the *ChargingNegotiationControl* and *ChargingStationReservationProtocol:ChargingStation* rectangle components.

In the *ChargingStationReservationProtocol:ChargingStation* component, the agent participates in the *ChargingStationReservationProtocol*, playing the Charging Station role. When in this protocol, it waits to receive a request message from EV agents for a reservation to a charging slot of the charging station (in the *ReceiveChargingReservationRequest* basic state). When such a message is received, the composite state *HandleChargingReservationRequest* is responsible to handle it. First, the *AuthenticateRecommendationProtocol* is used to verify the source of the recommendation which the requester (EV agent) is using to make the reservation. If the answer is negative (Disconfirm), the state directly terminates. If the answer is positive (Confirm), the agent request the electricity imbalance using the *ElectricityImbalanceRequestProtocol* in which it plays the Service User role.

After receiving the electricity imbalance, the agent calculates the charging schedule of the EV and makes a reservation to the requested charging slot. If the reservation is successful, the agent sends its updated charging schedule and confidence to the Mechanism Design and Electricity Imbalance agents using the *ChargingStationUpdateScheduleProtocol* and *UpdateEnergyProfileConfidenceProtocol*, sends its new availability to the Station Recommender agent using the *UpdateStationAvailabilityProtocol* and the state terminates. If the reservation is unsuccessful, the state directly terminates. The outcome of the reservation is sent to the EV agent that requested a reservation. Note that the details of the messages are presented in the inter-agent control of our system (see Section 6.1).

In the *ChargingNegotiationControl* component, the agent participates in the *ChargingNegotiationProtocol*, playing the Responder role (We remind that a negotiation is used when an already made charging reservation and the calculated charging schedule has to be reconsidered in order to capture the updated mood and preferences of the involved stakeholders). When a negotiation is over, the agent reviews the negotiation outcome (in the HandleNegotiationOutcome state), and based on the result, makes the appropriate updates to the corresponding agents. If the charging schedule and confidence were updated, the agents sends the updated quantities to the Mechanism Design and Electricity Imbalance agents using the *ChargingStationUpdateScheduleProtocol* and

UpdateEnergyProfileConfidenceProtocol. If the reservation details (i.e. the arrival and the departure of the EV to the charging station) were updated, the agent updates the Station Recommender agent using the *UpdateStationAvailabilityProtocol*. It is possible that one or none of the aforementioned quantities is updated, thus, it is not necessary for all the corresponding agents to be informed.

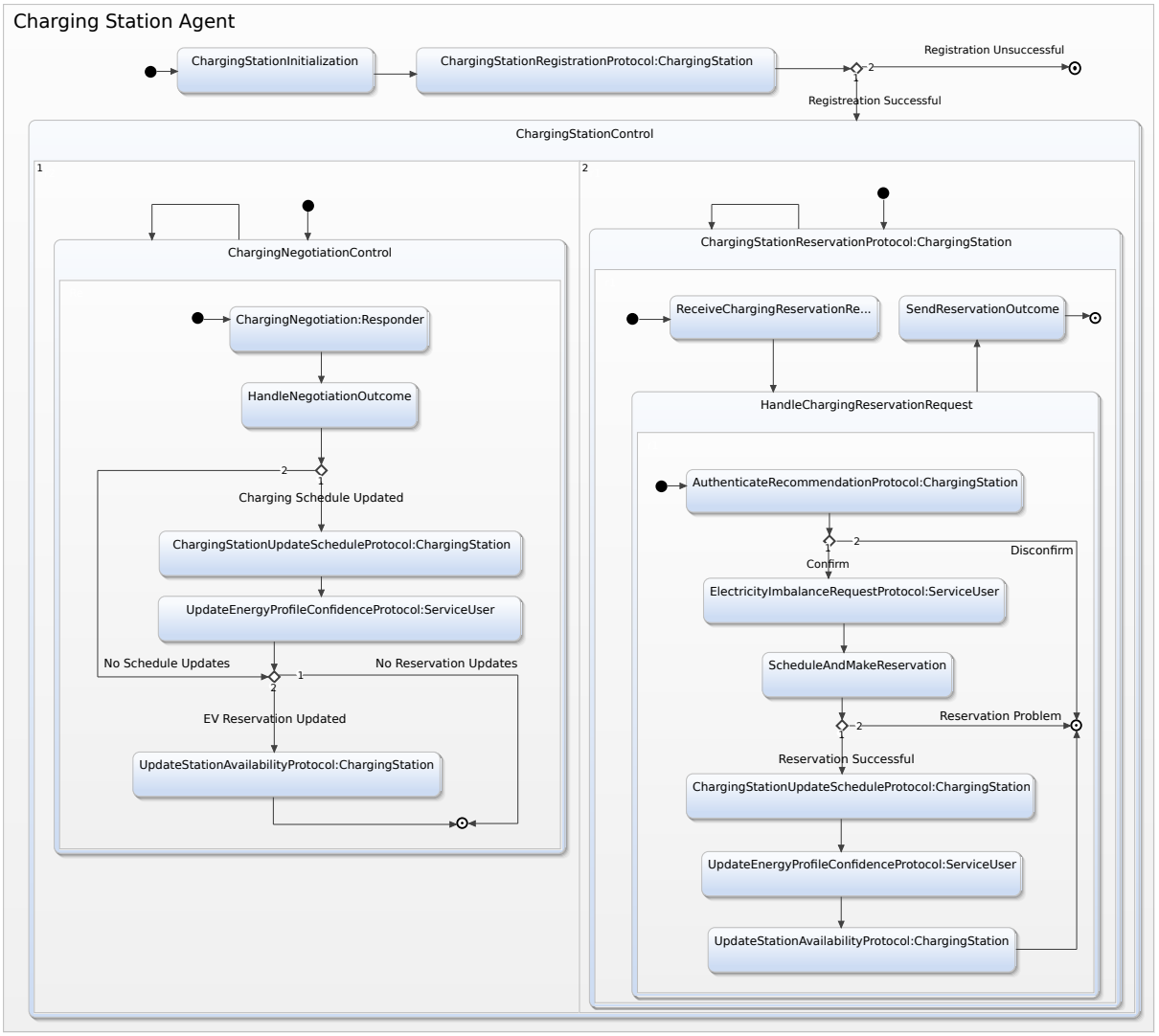

FIGURE 6.16: The intra-agent model of the Charging Station agent.

6.2.3 Station Recommender Agent

The intra-agent control of the Station Recommender agent is shown in Figure 6.17. When this agent is launched, it enters its initialization state where it registers its service with the DF, and synchronizes its internal simulation clock. Then it makes a transition to the *StationRecommenderControl* state. In this state there exist four parallel (AND) composite states that are executed indefinitely, waiting to handle specific requests.

The first composite state is about receiving the requests of Charging Station agents who wish to register with the Station Recommender agent. To receive these messages the agent plays the Service Provider role of the *ChargingStationRegistrationProtocol*. The second composite state is about receiving queries asking if a specific charging recommendation that was made to an EV agent, is genuine or not. To receive these messages, the agent plays the Station Recommender role of the *AuthenticateRecommendationsProtocol*. Next, the at the third composite state the agent receives requests from Charging Station agents who wish to update their charging slot availability. playing the Station Recommender role of the *UpdateAvailabilityProtocol*. The last, and the most interesting, composite state contains the activities of the Station Recommender agent to calculate and send charging station recommendations to EV agents.

When in this state, the agent waits to receive a request message for charging recommendations (in the *ReceiveRecommendationRequest* basic state). As soon as one such arrives, the agent requests the electricity imbalance and the electricity prices for the predefined planning horizon, using the *ElectricityImbalanceRequestProtocol* and *ElectricityPricesRequestProtocol* respectively. The information received, as well as the information provided from the EV agent about EV's current battery status and the driver's preferences, are input to a recommendations algorithm which calculates the charging recommendations (in the CalculateChargingRecommendations basic state). This algorithm is inserted into our agent using the functionality design pattern (see Section 6.3). Finally, the *SendChargingRecommendations* state sends the results to the EV agent who requested the recommendations.

6.2.4 Electricity Imbalance Agent

The intra-agent control of the Electricity Imbalance agent is shown in Figure **??**. When it is launched, it performs the initializations described for the others agents and enters the *ElectricityImbalanceControl* AND-state. In this state, six orthogonal components are executed in parallel, each containing a specific basic state that expresses a capability of the agent that corresponds to a specific role of a communications protocols.

As we can see, for all communication protocols, the agent plays the *ServiceProvider* role, that is, waits to receive messages, and depending on the protocol, to provide the appropriate answer. More specifically, the agent handles registration messages from Electricity Production, Consumption, and Charging Station agents; updates regarding the confidence of the various agents to their reported expected energy profile; from Electricity Producers and Consumer agents, updates about their expected production and consumption profile over a predefined planning horizon; from Charging Station

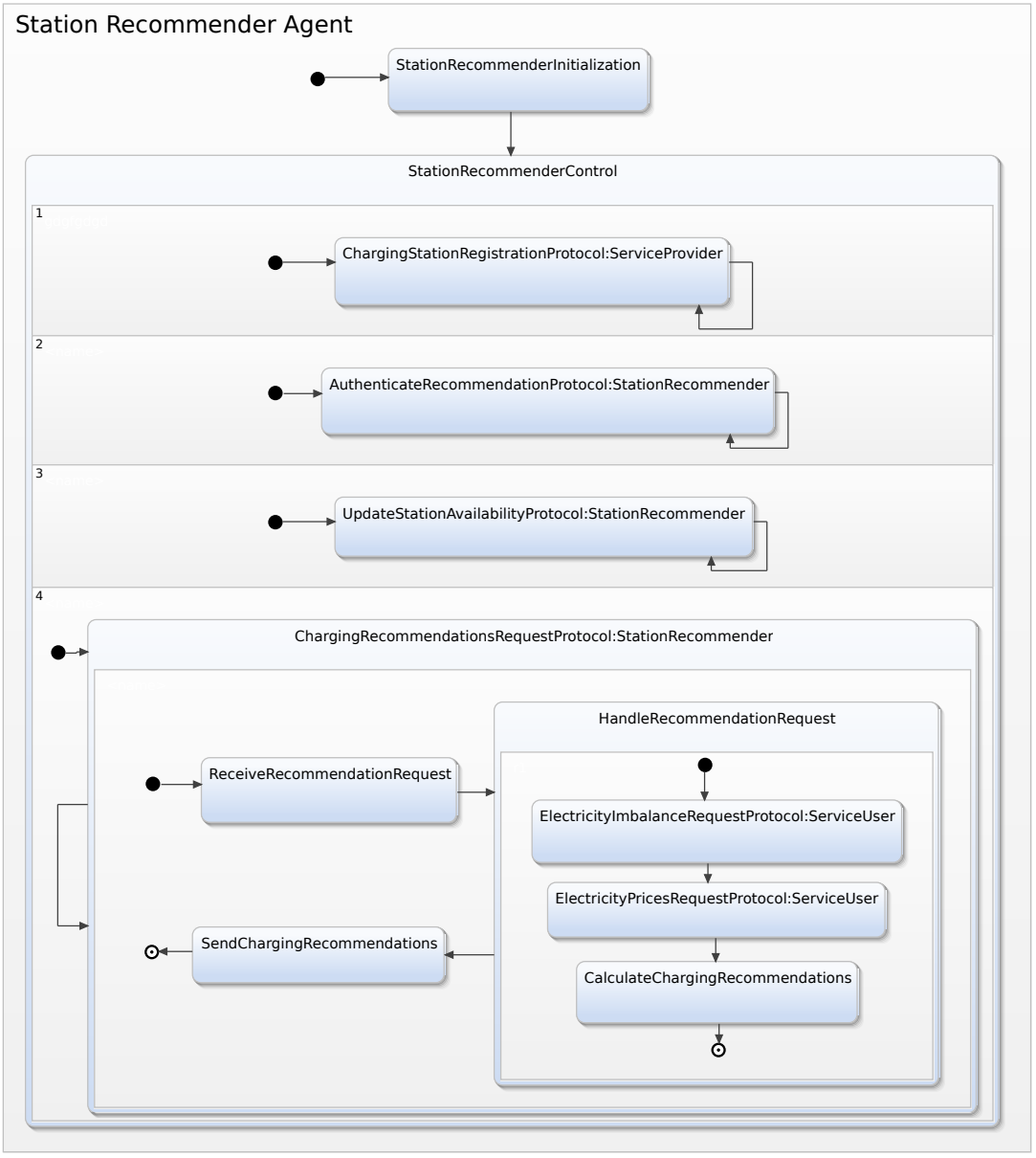

FIGURE 6.17: The intra-agent model of the Station Recommender agent.

agents, updates about their charging schedule; and finally, requests from various agents for the electricity imbalance of the grid over a predefined planning horizon.

6.2.5 Mechanism Design Agent

When it starts its operation, the Mechanism Design agent performs initialization activities (enters the *MechanismDesignInitialization* state, need to mention where it is Figure...). Then, it enters simultaneously in the six orthogonal components illustrated in Figure 6.19. For instance, in state *ChargingStationRegistrationProtocol:ServiceProvider* (using the *ServiceProvider* role of the respective protocol), it waits to receive messages

FIGURE 6.18: The intra-agent model of the Electricity Imbalance agent.

from Charging Station agents which request registration. In state *ElectricityPrices-RequestProtocol:MechanismDesignService*, the agent waits to receive requests for the current price of electricity. When such a request is received, it makes a transition to the state *ElectricityImbalanceRequestProtocol:ServiceUser* and requests the electricity imbalance. Then in state *CalculatePrices*, it calculates the prices and then sends the results to the agent who sent the request.

As we can see, all the depicted states inside the orthogonal components are re-executed after they finish indefinitely. This means that the Mechanism Design agent, depending on the protocol it uses, always waits to receive the appropriate requests. However, each state can not handle multi requests at the same time, only when the state has finished its handling with a request can handle another.

6.2.6 Electricity Producer and Consumer Agents

The intra-agent control of the Electricity Producer agent is shown in Figure 6.20. After the agent has registered its service with the DF and has synchronized its clock (both are activities of the *ElectricityProducerIntialization* state), it then enters the state *DF-SearchAgent*, where it requests and receives from DF the agent identifiers (AIDs) of the Electricity Imbalance and the Mechanism Design agents. Then, it plays the role *Producer* of the *ProducerConsumerRegistrationProtocol* in order to register with them. If the registration is unsuccessful the agent terminates, while, if the registration is successful, it enters the state *UpdateExpectedProductionControl* and sets a timer t1 to TIMEOUT hours. In this state, it calculates its expected production over a predefined planning horizon (*CalculateExpectedProduction* state) and then its confidence (*CalculateConfidence* state). The calculation of these quantities is part of the Preference Elicitation module of this agent. Then, the agent sends its expected production using the *UpdateExpectedProductionConsumptionProtocol* where it plays the Producer role, and its confidence using the *UpdateEnergyProfileConfidenceProtocol* where it plays the Service User role. The agent remains in the state *UpdateExpectedProductionControl* indefinitely, and every time the timer t1 timeouts, the computation resumes. In our implementation, the TIMEOUT period is set to 24 hours, thus, the Electricity Producer agent will calculate and send its expected production and confidence every 24 hours.

Note that the intra-agent control of the Electricity Consumer agent, as well as its description, is almost identical (excluding the state names) with that of the Electricity Producer agent, thus, we will avoid the repetition and we will not review it separately.

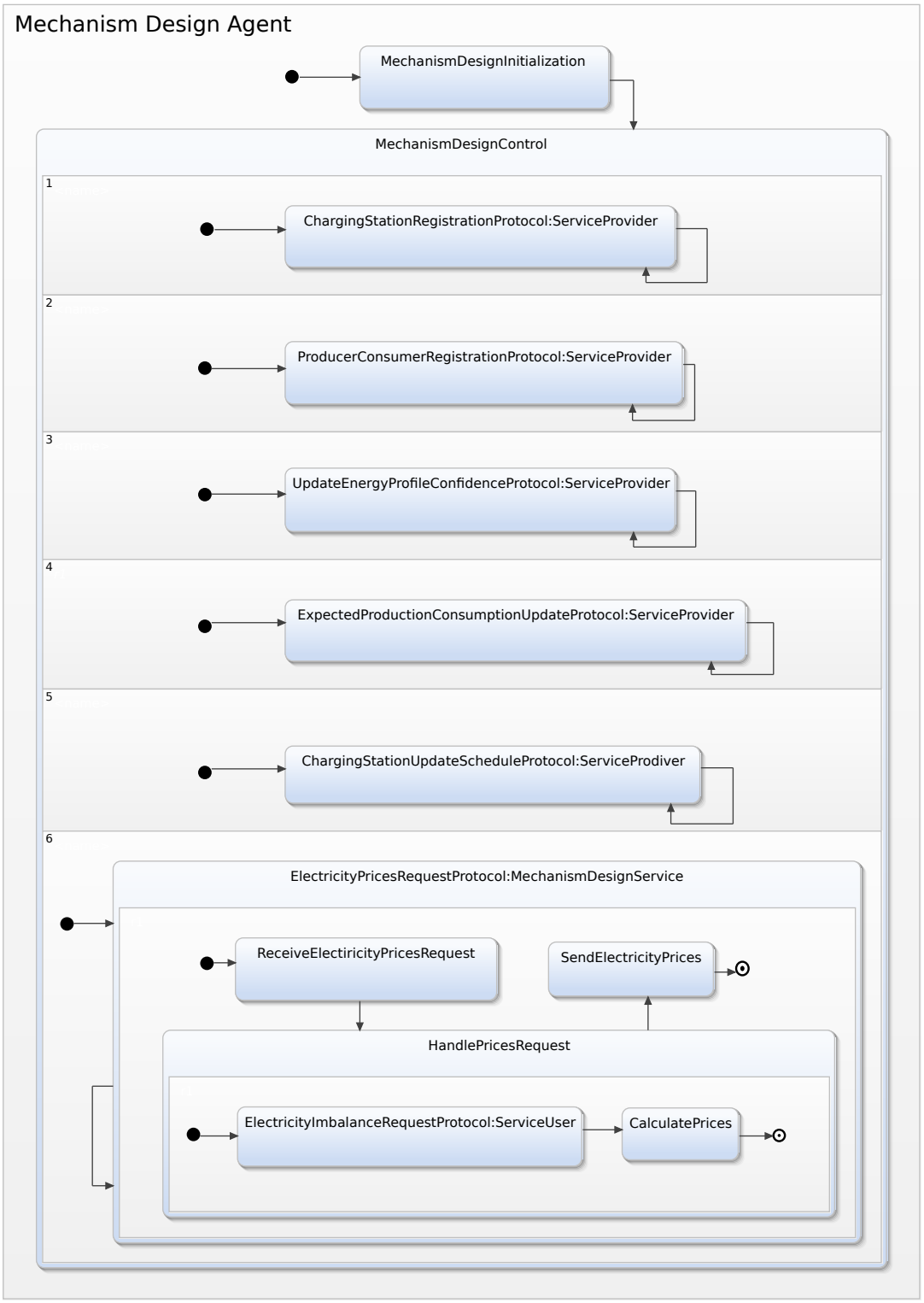

FIGURE 6.19: The intra-agent model of the Mechanism Design agent.

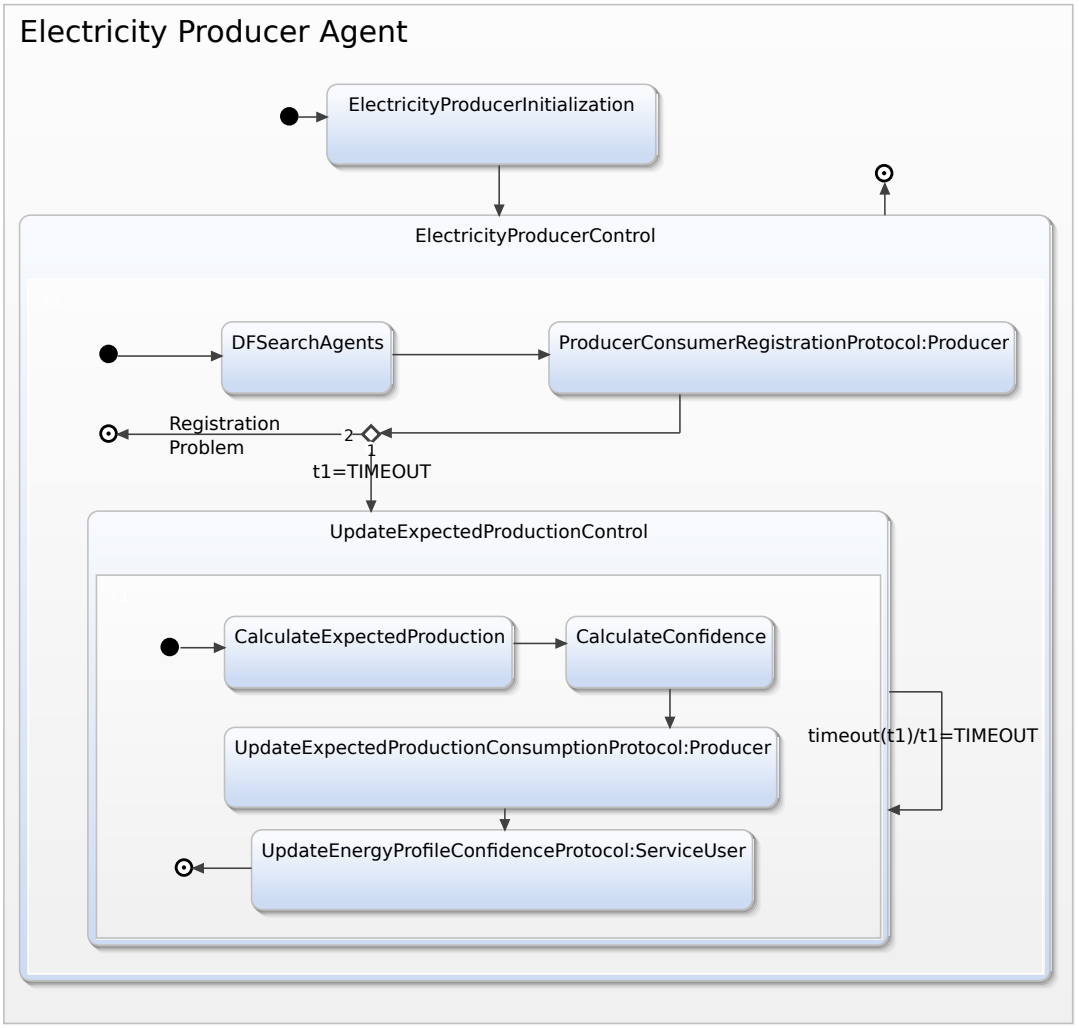

FIGURE 6.20: The intra-agent model of the Electricity Producer agent.

6.3 Design Patterns for Open Protocols

The architecture we presented in Section 5 poses several challenges for the rest of the development process. First, there is a need to accommodate different methods for decision-making based on user preferences, or on the business model of a stakeholder, or on the agent that implements a protocol role. As an example, both EV agents and Charging Station agents have the capability to negotiate (see Figure 5.1), however, it is obvious that they will most likely employ different algorithms to do so.

Thus, we needed to cater for agents following protocols to realize their goals in the system, while being able to define their own algorithms, policies or business rules. These cannot be foreseen at design time of the prototype.

In the first case, we had a decision-making behaviour, e.g. in a negotiation module, where the objects of the negotiation are quite clear, however, there are different strategies for

the agents to employ. We would like to allow the diverse agent developers to develop their own strategies aiming to make more competent agents. Thus, a specific function with clear parameters needs to be able to be supplied by the developing team.

In the second case, e.g. in service provider agents, it was clear that a service provider gets a request, processes it and then replies with an appropriate response. The process part, however, can be very different and changes not only based on policy but also based on the data structures and architecture of an agent. In this case, the agent developer needs to use a behaviour that is tailored to the agent's needs. This time it is not just an algorithm that changes, it is also the agent data structures - that are typically part of a behaviour but also the algorithm.

6.3.1 Capability Pattern

This pattern is based on developing an activity that will take place within the relevant state in the protocol. This activity is not implemented in the protocol package, it is rather passed as a parameter. Thus, although all agents use the same protocol package, they each develop their own specific state activity, which we will call the *handler behaviour*. The only requirement for this design pattern to work is the handler behaviour to have access to the agent's data structures, and the data structures of the protocol itself.

This section aims to define a design pattern for catering for the first case, where the developers need to associate a specific algorithm to a behaviour's action method. This is based on developing a behaviour that will replace the one in the protocol. The abstract behaviour is not implemented in the protocol package, it is rather passed as a parameter. Thus, although all agents use the same protocol package, they each develop their own specific process handler behaviour. The skeleton of this design pattern is illustrated in Figure 6.21, while the instaniation of this design pattern is depicted in Figure 6.22.

6.3.2 Functionality Pattern

In this section, we define a design pattern for catering for the first case, where the developers need to associate a specific algorithm to a behaviour's action method. To remedy this situation we relied on the well-known by practitioners *factory design pattern* [25]. According to this pattern we use a factory method that returns an instance of a method respecting an API which can be dynamically selected, even at run-time. The latter case can be extremely useful in emergency situations where the typical algorithm fails. This can also be used for self-healing of an autonomous - e.g. vehicle agent.

```
public class <ProtocolRoleName> extends FSMBehaviour {

  /*auto-generated state String representations should go here, such as the following
    line*/
  private final String HANDLER = "handler";

  public  <ProtocolRoleName>(Agent a, Behaviour handler, DataStore ds) {
    setDataStore(ds);
    handler.setDataStore(getDataStore());

    /*auto-generated transitions registrations should go here*/

  /*auto-generated state registrations should go here, such as the following line*/
    registerState(handler, HANDLER);
  }
}
```

FIGURE 6.21: Capability Pattern skeleton.

```
    <Capability0>Behaviour b = new <Capability0>Behaviour(customAgent, Object[] params);
    addBehaviour(new <ProtocolRoleName>(customAgent,b,dataStore));
```

FIGURE 6.22: Capability Pattern instantiation skeleton.

Figure 6.23 illustrates the class diagram of this design pattern. The superclass *Template1*, contains a standard and generic API that the various subclasses (e.g. *Instance1, Instance2, Instance3*) use, providing their own implementation for the abstract methods of this API. This template can be the utility function used by the *EV agent* to select recommendations. The Agent (*EV*) implements the *Factory1* static class providing the full class name of the algorithm it wants to use. The factory is responsible for creating the requested algorithm and providing it to the Agent. The various agent behaviours which have access to the Agent's class data have the necessary access to utilize the algorithm according to their needs. An implementation of this design pattern for the EV agent is illustrated in Section 6.2.1, where we present its the inta-agent control.

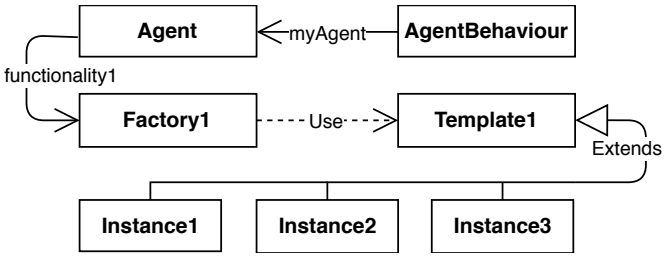

FIGURE 6.23: The abstraction of the Functionality Design Pattern.

6.3.3 Discussion

These patterns are agent methodology- and platform-independent. They can be applied with any statecharts-based method and influence code generation for diverse platforms. Both patterns are compatible with object-oriented development, which, as a norm, is supported by most methodologies for Engineering MAS (e.g. [17, 19])

The pattern in Figure 6.21 shows how an Xpand technology [20] code generation template could be defined for the Eclipse platform (following the ASEME IDE). Similar templates can be defined for other object-oriented implementations of statechart execution engines.

6.4 Synchronization and Time Agent

In a system with multiple agents who have to communicate and coordinate, the synchronization of actions is a crucial factor to their effectiveness. The Time Agent is a utility agent responsible to provide time services to all the other agents of the system. When the various agents initiate, they communicate with this agent in order to receive date and time information that is going to enable the synchronisation of their internal clock with the clocks of the various other agents in the same environment. Note that the transfer of the system to the "wall clock" will make the existence of the simulation clock redundant.

When the Time Agent starts its execution, it stores the real date and time expressed in milliseconds of that particular moment t_{start}, which we assume its the time zero of the system. Other stored attributes are the real duration of each time interval expressed in milliseconds t_{slot}; a value denoting the simulation date-time starting point (which can be different from the real date-time) t^{sim}_{start}; and finally a value which contains the duration of a time interval in simulation time t^{sim}_{slot}. For example, the real time zero of the system can be the current date-time expressed in milliseconds between that date-time and `1/1/1970-00:00:00:000`; the real duration of each time interval expressed in milliseconds can be 3000 milliseconds; the simulation start date-time can be `30/12/2017 00:00:00:000` and the simulation time interval can be 5 minutes. Thus, every 3000 miliseconds in the real world are translated to 5 minutes simulation time.

In JADE, we can represent the clock of an agent using a `TickerBehaviour`. The synchronisation is achieved by combining a `WakerBehaviour` and a `TickerBehaviour`. When the synchronization information is received, we can calculate the specific date and time of the next tick of the clock, t^{tick}_{next} (i.e. of the `TickerBehaviour` of all agents). Then, we

set the `WakerBehaviour` to be executed at time t_{next}^{tick} (see Equation 6.2), and when it will be executed, it will add to the list of active behaviours the `TickerBehaviour` with tick period, t_{slot}. In this way we synchronize within milliseconds the clocks of the agents that share the same Java Virtual Machine (JVM) (the `TickerBehaviour`s of all agents, will tick at the same time). Whenever we want, We can convert the real current time to simulation time, as we have available the start date-time and the time interval of the simulation.

An agent can calculate the current real time of the system using the following formula:

$$t_{curr} = t_{start} + t_{slot} \times k \tag{6.1}$$

where k is the number of time slots elapsed from system's time zero. However, t_{curr} is not unknown, as we can use time APIs such as `System.currentTimeMillis()` in Java to obtain it. Thus, we can utilize this information to calculate $k = \frac{diff}{t_{slot}}$, where $diff = t_{curr} - t_{start}$. We can calculate the date-time of the next clock tick using the next equation:

$$t_{next}^{tick} = t_{start} + (k \times t_{slot}) + t_{slot} \tag{6.2}$$

Chapter 7

Evaluation

The aim of this study, besides the design and the development of an integrated multi-agent V2G/G2V architecture for the evaluation of theoretical models and algorithms, is to answer the question whether a potential solution will be effective and truly useful in the real world. In order to evaluate this, given that the deployment of software agents to real-world charging stations and electric vehicles is incredibly expensive, we first must conduct simulations that are based on real-world scenarios and datasets. In this chapter, we discuss the details of the implemented simulator, the datasets we utilized and the usage scenarios that we executed.

7.1 Implementation Details

To evaluate and prove the functionality of our system, it is required to implement specific algorithmic components that will produce results in order to populate the data structures of the various agents. Such an algorithm is about the charging station recommendations which is used from the Station Recommender agent. In this section we will briefly present this algorithm for completeness purposes, however, we will not present the utility functions used from the EV agents to select charging recommendations and the charging scheduling algorithm which is used from the Charging Station agents to schedule EVs due to their simplicity (e.g. EV charges immediately upon connection). Note that the proper implementation of the aforementioned algorithms is out of the scope of this work.

For our experiments, we derived valuable data from Zap-Map[1], a very popular EV platform in the UK with more than 80.000 visitors each month. The public data available in this platform include electric vehicle and charging station specifications, charging

[1] https://www.zap-map.com

station locations, reviews from each station, availability information and many more. The available information in this platform is crowdsourced from real EV drivers. We combined the aforementioned real-world datasets, with real-world datasets from other sources, as data for our specific use case are not currently available. The data what our use cases required, but however we did not have available, were created synthetically using appropriate distributions with properties derived from the related literature.

The various dataset generators that we developed to combine the aforementioned datasets are implemented in Java, and they are easily configurable. This means that the interested user can easily modify them using her own configurations and datasets. Furthermore, the majority of the real-world datasets used in the context of this thesis, are documented using the JSON [2] format, thus, their management and extension are very simple.

7.1.1 Station Recommendation Algorithm

In order to make recommendations to EV agents regarding the charging station that match better with their preferences and needs, we implemented a very simple recommendation algorithm which was inserted to the Station Recommender agent using the functionality design pattern as presented in Section 6.3. The implementation of the algorithm in Java can be found in Appendix A. Given an EV's type, which contains its preferences (e.g. arrival and departure time, charging type) and its battery's state (e.g. soc, soh, charging efficiency), the recommendation algorithm, first checks if for the given period of connection, that is, between EV's arrival and departure, charging slots are available (i.e. without reservations from other EVs). For the available charging slots, the algorithm filters these which are compatible with EV's charging inlets. If the driver prefers a specific charging type, then this step of the algorithm returns only the corresponding charging slots. For the selected charging slots, the algorithm calculates the energy which the EV can receive during its connection and the actual requested energy as described from the preferences of the driver. Depending on these values, it makes a decision about the amount of energy that will recommend. If the requested energy is more than the available energy that the EV can receive during its connection, then the latter is recommended, otherwise, the requested energy. The recommended prices are calculated using both the pricing policy of the charging station for a specific charging slot, as well as the electricity prices as received from the Mechanism Design agent. Finally, the algorithm checks if the EV preferences contain a specific charging station network, and if this is the case, it filters the stations of the preferred network. Note that the implementation of a more sophisticated recommendation algorithm is something we consider in our future work.

[2] https://www.json.org

TABLE 7.1: Typical charging station slot specifications.

Connector	Power	Current	Price
Type 2	$\leqslant$ 3.7kW	1Φ AC	0.01 €/kwh
Type 2	$\leqslant$ 7.0kW	1Φ AC	0.015 €/kwh
Type 2	$\leqslant$ 22kW	3Φ AC	0.02 €/kwh
CSS	$\leqslant$ 50kW	DC	0.03 €/kwh
CHAdeMO	$\leqslant$ 50kW	DC	0.03 €/kwh

7.1.2 Charging Stations Dataset

To create realistic charging stations, we used both synthetic and real-world data which are freely available. Table 7.1 summarizes the charging station slots specifications that are derived from the *Zap-Map* platform. The *Power* column, denotes the maximum power that a particular charging slot can provide, the *Current* column denotes the electric current type of each charging slot and the *Price* column denotes the amount for charging at a particular charging slot has to pay. These prices are arbitrary and are based on the assumption that the faster you charge, the more you have to pay. Each charging station can be equipped with a number of charging slots of each charging standard and power output. For our experiments and we assume that all the charging slots have equal probability to be available at a charging station, however, the probability values are easily reconfigurable. Inherently, each charging station is placed at a specific location in a given area that is described by a set of coordinates (latitude and longitude). In this study, we assume that charging stations are located where traditional gas stations are located. To find real-world gas station locations, we used the free version of GoogleMaps Places API [3] to download a dataset which contains the locations of 60 real-world gas stations in the city of Chania (Figure 7.1).

7.1.3 Electric Vehicles Dataset

To model EVs that are as realistic as possible, we used real-world data that are freely available in combination with synthetic that are based on empirical statistical properties. Table 7.2 summarizes the specifications of a selection of five real world EVs that were used for our evaluation from the *Zap-Map* platform. The consumption profiles of each EV, was derived from US EPA[4]. To determine the current SOC of an EV, we sampled a beta distribution $\mathcal{B}(4, 17)$ and for the target SOC of EV's battery upon disconnection, we sampled a normal distribution $\mathcal{N}(0.6, 0.1)$ bounded to the interval $[SOC, 1]$. The SOH

[3] https://cloud.google.com/maps-platform/places/
[4] https://www.epa.gov/

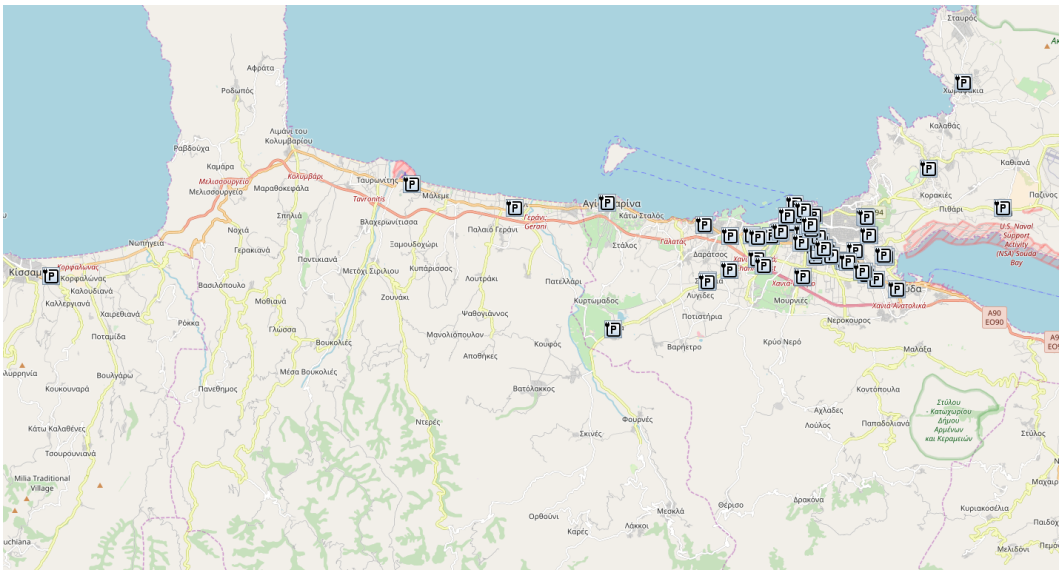

FIGURE 7.1: Locations of real-world gas stations in the city of Chania, Crete, Greece

TABLE 7.2: Battery and charging specifications of some popular EV models.

EV Model	Battery Capacity	Consumption	Charging Methods	
			Inlet	Max Power
BMW i3	33 kWh	18.01 kWh/100km	Type 2 CCS	11 kW 50 kW
Hyundai Ioniq	28 kWh	15.53 kWh/100km	Type 2 CCS	6.6 kW 50 kW
Nissan Leaf	40 kWh	18.64 kWh/100km	Type 2 CHAdeMO	6.6 kW 50 kW
Renault Zoe	41 kWh	10.25 kWh/100km	Type 2	43 kW
Smart ForTwo ED	17.6 kWh	20.5 kWh/100km	Type 2	7 kW

of the battery is sampled from a beta distribution $\mathcal{B}(30, 1)$ and the charging efficiency is set to 0.95 plus some noise sampled from a normal distribution $\mathcal{N}(0, 02)$.

The exact location of an EV at the time it requests charging recommendations is described by the coordinates longitude and latitude. To generate these coordinates for each EV, we selected and used five square sectors in the Chania region. Each sector is bounded by a north and south latitude and an east and west longitude (Table 7.3). Special care was given to avoid areas that are covered by the sea. These values are exported from *OpenStreetMaps*[5], an open data, community-driven, maps platform. Figure

[5]https://www.openstreetmap.org/

TABLE 7.3: Coordinates of the areas in which EVs are located when request charging recommedantions.

Location	NorthLati	SouthLati	WestLongi	EastLongi
Polytexneio-Kounoupidiana	35.5394	35.5254	24.0615	24.0882
Chania City Center	35.5157	35.4952	24.0094	24.0463
Southern Chania City	35.4995	35.4722	23.9841	24.0482
Western Chania City	35.5109	35.4983	24.0112	23.9448
Souda-Tsikalaria	35.4888	35.4735	24.0765	24.0484

7.2 illustrates the method we used to extract the values for the case of *Polytexneio-Kounoupidiana* sector, using the *OpenStreetMaps*'s area export tool. Then, we used the derived sector data to sample two uniform distributions $\mathcal{U}(southLati, northLati)$ and $\mathcal{U}(westLongi, eastLongi)$, for latitude and longitude respectively. We use uniform distribution to preserve generality by not making any assumption regarding the actual EV concentration at a particular location. For our experiments, we assume that the probability of an EV to be located at the *Chania City Center* sector is higher than the other sectors, and for the requirements of future experiments, we can easily modify the corresponding probabilities. The arrival of an EV to a charging station is created synthetically by adding the current time of the system and a random number of minutes sampled from a normal distribution $\mathcal{N}(120, 40)$. For the charging duration of an EV (i.e. its departure time), we used a real-world dataset that contains information about the charging duration of EVs at public charging stations, part of the Low Carbon London (LCL) project [6]. From 1588 charging events, the average connection duration time is 285 minutes and the standard deviation is 259 minutes, thus, to generate the charging duration of our EVs, we sample a normal distribution $\mathcal{N}(285, 259)$, bounded to the interval [15,1440] i.e. from 15 minutes to 24 hours. Note that, all probability distributions can be configured to other types, according to the needs of the user for each case.

7.1.4 Electricity Production and Consumption Datasets

To generate the expected production of the various DESs and the expected consumption of entities such as households and industries, we sampled normal distributions with arbitrary mean and standard deviation values. The reason is that if we used real-world statistical properties to calculate the hourly production and consumption, we would have to aggregate a considerable number of producers and consumers to create a meaningful electricity imbalance and effectively evaluate our system, and that was out of the scope of this thesis. Here again, the replacement of our statistical values with realistic hourly

[6] https://www.europeandataportal.eu/data/en/dataset/low-carbon-london-electric-vehicle-load-profiles/resource/e607bf5b-d71d-428c-ab62-97a828c1643f, accessed November 3, 2018,

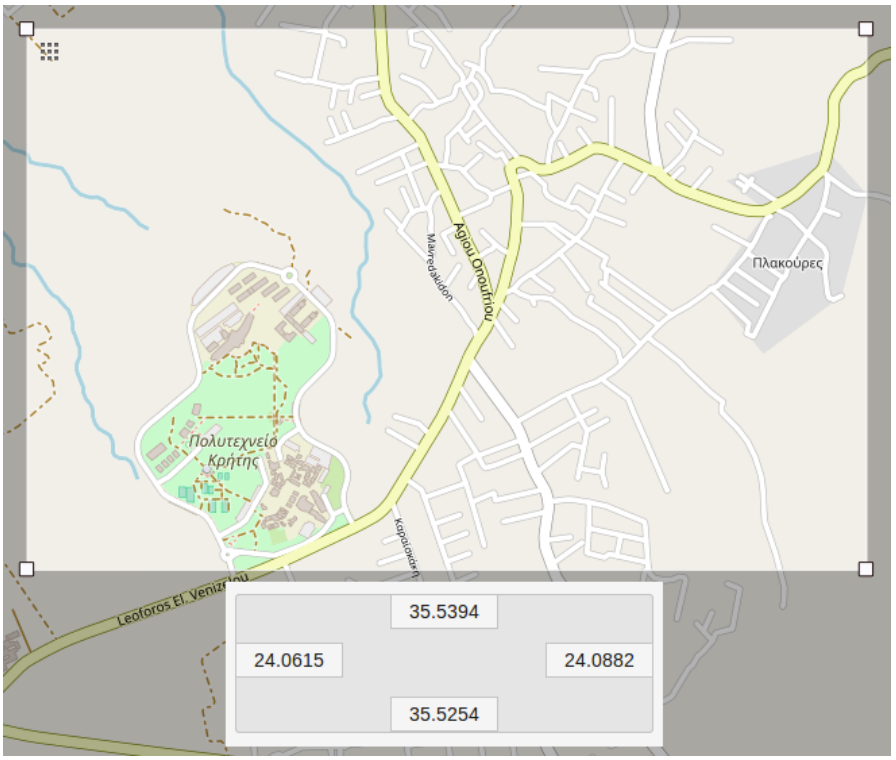

FIGURE 7.2: The area selection tool of *OpenStreetMaps* which was used to extract the *Polytexneio-Kounoupidiana* sector (the bright area). The coordinates box depicts its bounds.

mean and standard deviation data for each day and each month is effortless. Such data are already available in the literature, thus, for realistic datasets of electricity production and consumption, we refer the interested reader to the works of Akasiadis and Chalkiadakis [3] and Chalkiadakis et al. [12].

7.2 System Evaluation

In order to demonstrate the functionality of the MAS prototype proposed in this thesis, we show use case scenarios for the various implemented agents and their interactions. For the visualization of the results, we used graphical user interfaces that we implemented in Java, as well as JADE's Sniffer agent. The latter agent marks with the same color the conversations that share the same conversation id. In our implementation, the instances of the various communication protocols have same conversation id, and thus, the same color.

The message exchanges presented below match with the inter-agent controls (the communication protocols) presented in Section 6.1 and the sequence that the agents use these protocols is defined in the intra-agent control of each agent presented in Section 6.2.

The execution of the use case scenarios conducted using JADE version 4.5 on a 4GB RAM, i5 CPU/2.67 GHz laptop and Ubuntu 18.04.1 LTS.

7.2.1 Electricity Producer and Consumer Agent: Registration and Updates

The execution of this scenario can be seen in Figure 7.3. The agents time-lines shown in the figure are the Mechanism Design agent *(MD)*, the Electricity Imbalance agent *(IM)* and a Electricity Producer agent *(EP_13)*. *EP_13* requests to register with both the *MD* and *IM* agents using the *ProducerConsumerRegistrationProtocol* (lines 1-4, REQUEST and INFORM messages exchange). If the registration is successful, then the *EP_13* agent sends a request to update its expected production for a predefined planning horizon. This request is sent both to the *MD* and *IM* agents using the *ExpectedProductionConsumptionProtocol* (lines 5,9 and 6,8). Together with the expected production, the *EP_13* agent sends and its confidence regarding its expected production using the *ConfidenceUpdateProtocol* (lines 7,10 and 11,12). The latter two protocols are executed periodically as the producer can have updated beliefs about its expected production profile and its confidence to it. The exact same procedure with the *EP_13* agent, is performed by the Electricity Consumer agent *EC_18* and it is depicted in Figure 7.4.

The details of some messages of the conversations between *EP_13* and *IM* are shown in Figure 7.5. To increase readability, we reduced the amount of information of the field *content* of the second REQUEST message, as it is a list of items with repeating pattern i.e. `(EnergySlot :dateTime <dateTime> :kwh <kwhs>)` (each item for a specific time interval).

Figure 7.6 illustrates a GUI that we developed to monitor the electricity that is produced and consumed in the grid. The left panel of the figure shows the imbalance between the local electricity production and consumption as reported by the corresponding agents, while the right panel shows the energy profiles of the aggregated production and consumption over a 48-hour planning horizon.

7.2.2 Charging Station Agent: Registration

The execution of this scenario can be seen in Figure 7.7. The agents time-lines shown in the figure are the the Electricity Imbalance agent *(IM)*, Mechanism Design agent *(MD)*, the Station Recommender agent *(SR)* and a Charging Station agent *(CS_73)*. *CS_73* requests to register with *MD*, *IM* and *SR* using the *ChargingStationRegistrationProtocol* (lines 1-6, REQUEST and INFORM messages exchange). If something goes wrong

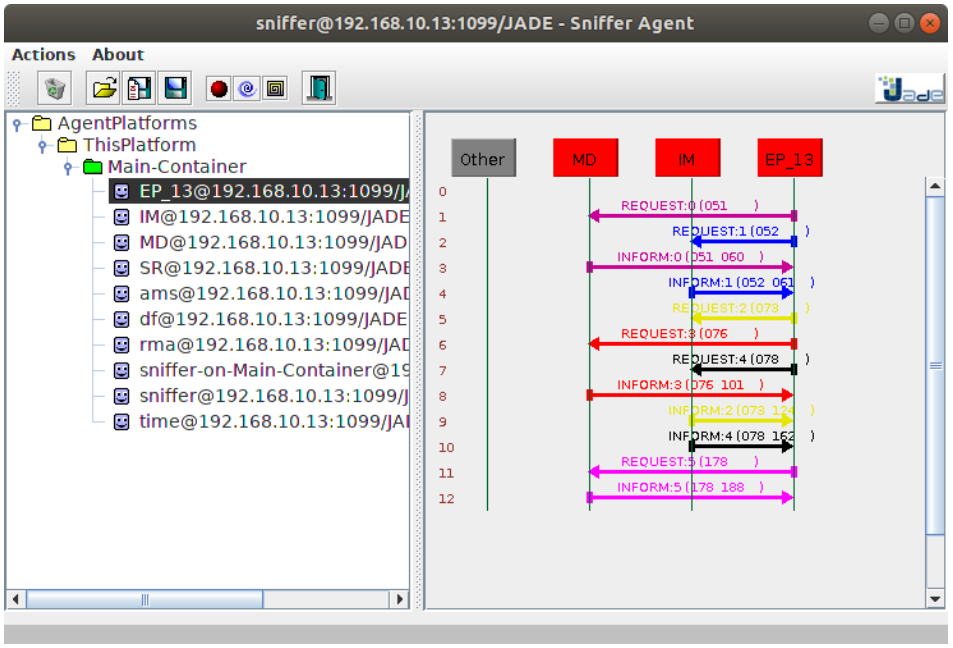

FIGURE 7.3: A screenshot of the JADE sniffer agent that illustrates the registration process and the updates of an Electricity Producer agent.

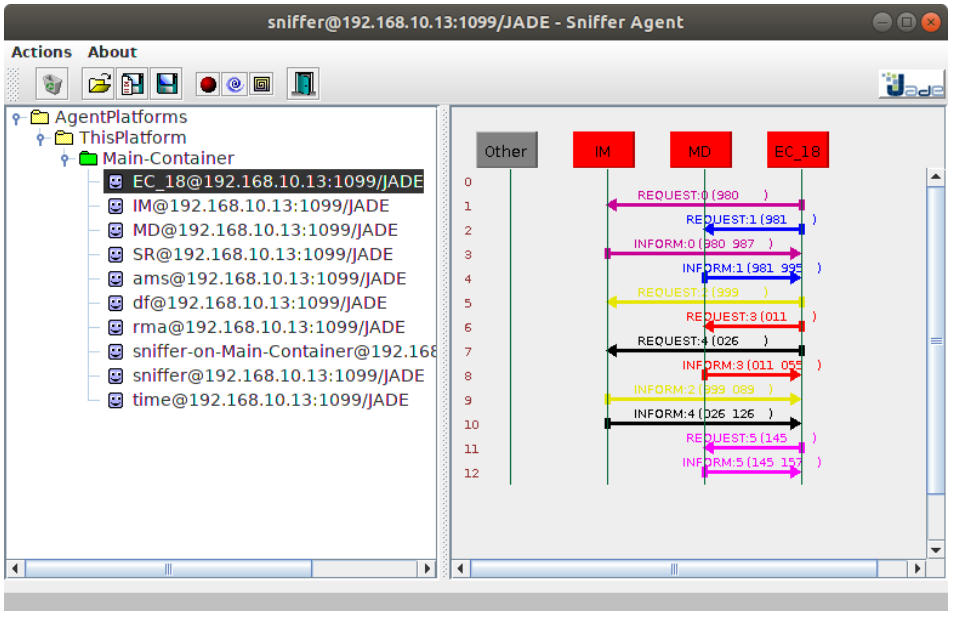

FIGURE 7.4: A screenshot of the JADE sniffer agent that illustrates the registration process and the updates of an Electricity Consumer agent.

FIGURE 7.5: Some representative messages of the conversations between *EP_13* and *IM*.

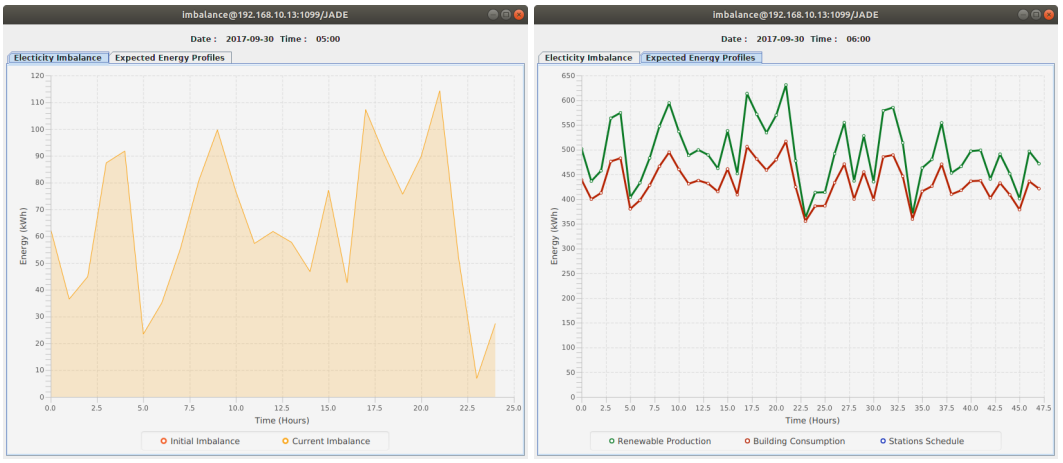

FIGURE 7.6: A screenshot of the GUI of Electricity Imbalance agent that depicts the electricity imbalance (left) and the profiles of the aggregated electricity production and consumption (right) of the grid..

with the registration (e.g. the registration message had missing information), a failure message will be sent from the latter agents describing the problem occurred (see Section 6.1). Figure 7.8 illustrates the GUI of the Station Recommender agent that visualizes information about the available charging station in the grid. On the top, it shows the locations of the charging stations, and on the bottom, the charging slot connector types of the registered charging stations. For this particular use case scenario, we have available among others, 18 *Type 2 at 7.0kW* and 11 *CHAdeMO at 50kW* charging slots.

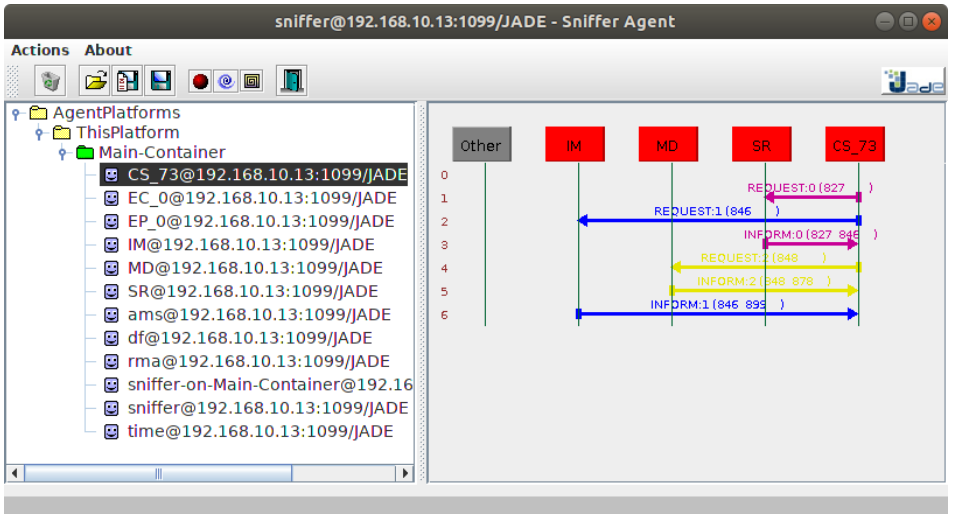

FIGURE 7.7: A screenshot of the JADE sniffer agent that illustrates the registration of a Charging Station agent with various other agents.

7.2.3 Electric Vehicle Agent: Station Recommendations and Reservation

The execution of this scenario can be seen in Figure 7.9. The agents time-lines shown in the figure are the Mechanism Design agent *(MD)*, the Electricity Imbalance agent *(IM)*, the Station Recommender agent *(SR)*, a Charging Station agent *(CS_1)*, and an Electric Vehicle agent *(EV_0)*. *EV_0* requests charging recommendations from *SR* using the *ChargingRecommendationProtocol* (line 1, REQUEST message) and waits for a response. *SR*, upon receiving such a request, requests the electricity imbalance from *IM* using the *ElectricityImbalanceRequestProtocol* (lines 2-3, REQUEST and INFORM message exchanges) and the electricity prices from *MD* using the *ElectricityPricesRequestProtocol* (lines 4,7). To calculate the electricity prices, *MD* requires the electricity imbalance as well, thus it uses the *ElectricityImbalanceRequestProtocol* (lines 5,6). Having the imbalance, the prices and EV's type, *SR* agent calculates and sends the recommendations and the *ChargingRecommendationProtocol* terminates (line 8, INFORM message).

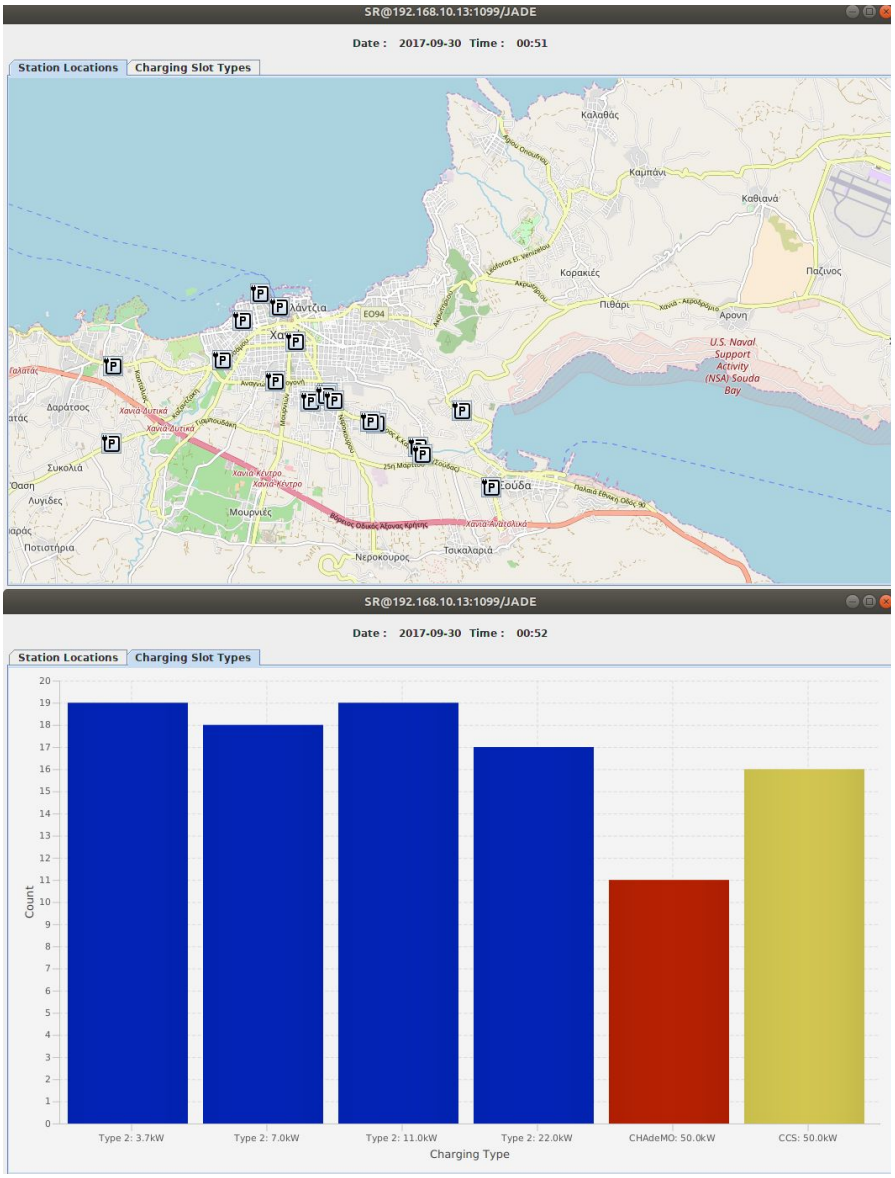

FIGURE 7.8: A screenshot of the GUI of the Station Recommender agent.

After receiving the charging recommendations, *EV_0* requests a reservation to the selected from the recommendations *CS_0* using the *ChargingStationReservationProtocol* (line 9, REQUEST message) and waits for a response. Note that in Figure 7.9 we have multiple charging stations available (e.g. *CS_1*, *CS_2*,) but *EV_0* selects *CS_0* due to its utility function. Before answering, *CS_0* invokes the *AuthenticateRecommendationProtocol* in order to verify that a recommendation is genuine (lines 10,11, QUERY-IF and CONFIRM messages), and then requests the electricity imbalance using the *ElectricityImbalanceRequestProtocol* (lines 12,13). Having this information, *CS_0* calculates the charging schedule of *EV_0* and makes a reservation. After that, it sends its new schedule and confidence using the *ChargingStationUpdateScheduleProtocol* and *ConfidenceUpdateProtocol* to the *IM* (lines 14-17) and *MD* (lines 18-21). Then, it sends its

updated availability to *SR* using the *AvailabilityUpdateProtocol* (lines 22,23). At the end, it replies with an *INFORM* performative to *EV_0* (line 24).

For the requirements of this use case scenario, the *SR* agent was configured to use the charging recommendation algorithm that was briefly presented in Section **??** and the *EV* agent was configured to use the utility function that selects the charging recommendation with the minimum price. Both algorithms were imported to the corresponding agents using the functionality design pattern presented in Section 6.3.

As we explained in the previous chapters, the aforementioned communication process is initiated from the Preference Elicitation module of the EV agent and can be either automatic or manual (the driver sets her preferences using a specific GUI). For the particular use case scenario we discuss in this section, the process for *EV_0* started using its GUI, as the autonomous decision making of the EV agent is out of the scope of this thesis. However, the recommendation selection is performed automatically and is based on the aforementioned utility function. The graphical panel that illustrates the selected charging preferences, the received recommendations and the selected recommendation used to make a charging reservation, is presented in Figure 7.10. The selected recommendation (the green-colored line) is the one with the minimum price per kWh, which was the expected outcome as the agent used a minimum price utility function. In the next subsection, we will compare three utility functions that are based on different criteria and we will discuss the results.

7.2.4 Electric Vehicle Agent: Recommendation Selection Utility Functions

For this use case scenario, we implemented three very simple utility functions to evaluate the functionality of our design which allows the easy incorporation of alternative functionality using the functionality design pattern presented in Section 6.3. The implemented utility functions are the following:

- **Maximum Energy**: Selects the recommendation with the maximum energy
- **Minimum Distance**: Selects the recommendation with the minimum distance
- **Minimum Price**: Selects the recommendation with the minimum price

Note that an EV agent can select the recommendation selection algorithm even on runtime through the GUI of the EV agent (see Figure B.3).

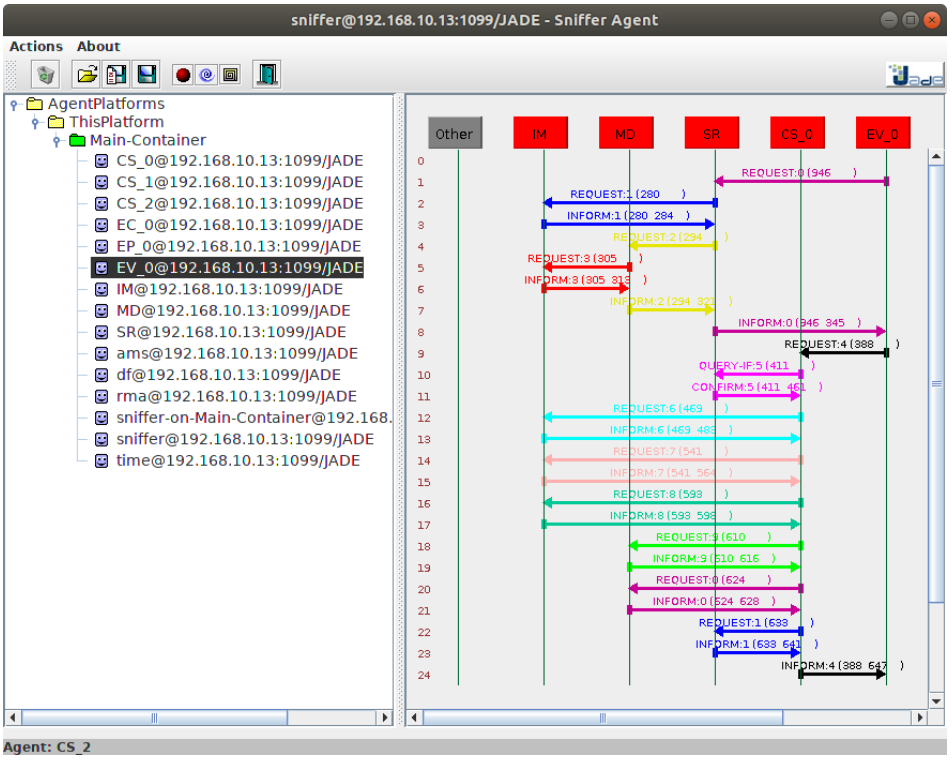

FIGURE 7.9: A screenshot of the JADE sniffer agent that illustrates the communication process of an EV agent that requests charging recommendations and makes a reservation at the selected charging station.

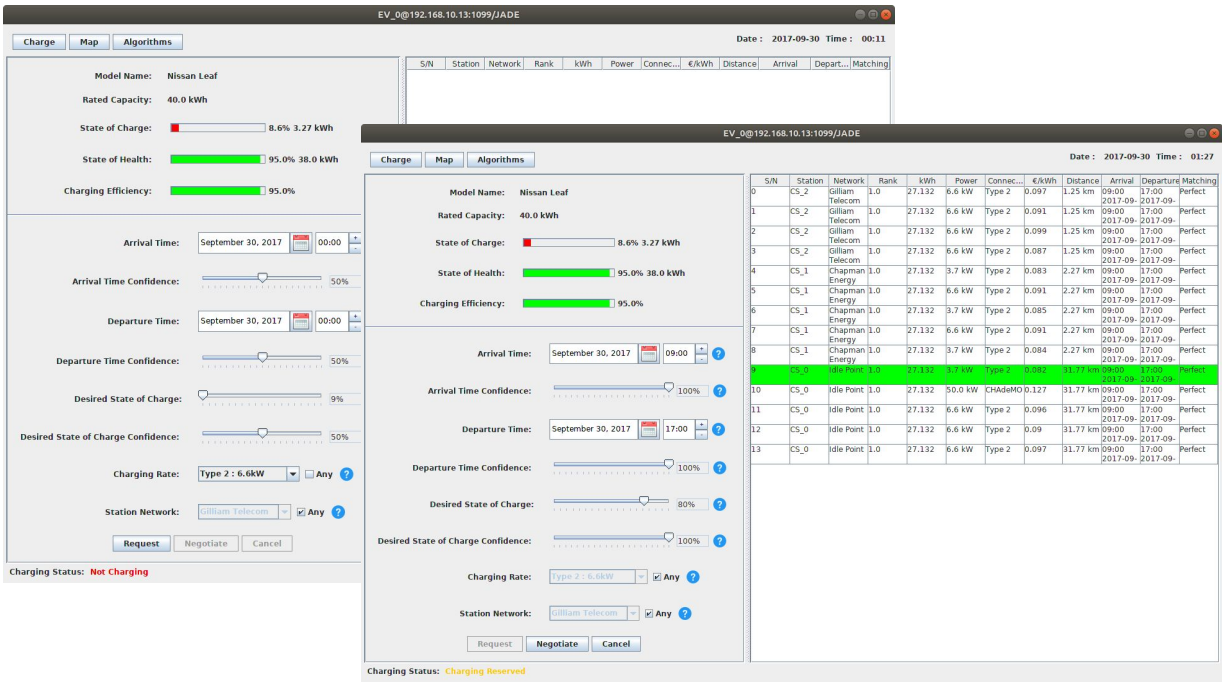

FIGURE 7.10: A screenshot of the GUI of the EV agent that illustrates the charging preferences as set by the driver, the received recommendations based on these preferences, as well as the selected recommendation which was used for the reservation to the Charging Station agent (green-marked recommendation).

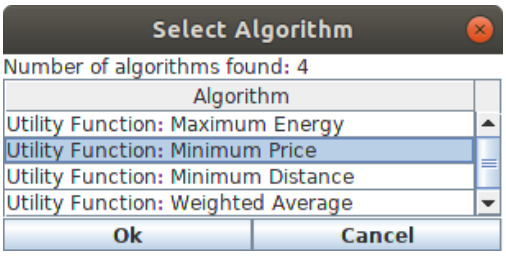

FIGURE 7.11: Different algorithms for selecting the best station. The user selects the desired one at runtime, or the agent selects it after assessing user preferences.

TABLE 7.4: Average results for the three different utility functions used from Electric Vehicle agents to select charging recommendation .

Measure	Maximum Energy		Minimum Price		Minimum Distance	
	Mean	Std	Mean	Std	Mean	Std
Price (€)	0.10	0.009	0.089	0.0026	0.098	0.0087
Energy (kWh)	13.11	5.67	11.98	4.84	10.60	4.61
Distance (km)	5.99	10.50	3.90	5.99	0.44	0.26

Furthermore, for the Station Recommender agent, we implemented a very simple recommendations algorithm which was briefly presented in Section 7.1.1. We aggregated the results of three independent executions of the system for the three different utility functions, and for each execution, we launched 100 Electric Vehicle agents and 60 Charging Station agents. Their initialization is described in Section 7.1. The results of this use case scenario are summarized on Table 7.4. When the agents use the utility function that selects the recommendation with the maximum energy, we observe that the average energy of the selected recommendations for all agents is the highest in comparison with the other two utility functions. Similarly, when the utility function selects the recommendation with the minimum price, the average price of the selected recommendations is the lowest. Finally, when the utility function selects the recommendation with the minimum distance, the average distance of the selected recommendations is the lowest. These results show what we expected to happen, that is, depending on the utility function, we have alternative results.

The purpose of this scenario was mainly to present the convenience that our design patterns offer to a realistic simulation setting, where the aim is to evaluate alternative utility functions and not the development of a fully functional multi-agent system from scratch. The developers can reuse the protocol parts and logic defined in our framework, and focus only on the implementation of custom key functionalities or capabilities of the agents according to their needs/goals.

7.2.5 Electric Vehicle - Charging Station: Negotiation

The final use case scenario of our system demonstrates a negotiation between an EV agent and a Charging Station agent, and it is the continuation of the scenario presented in Section 7.2.3. The negotiation is initiated by the EV agent and the initiation process is illustrated in Figure 7.12. As the EV agent has an active reservation to a Charging Station agent, it can start a negotiation, thus the GUI enables the negotiation option (see the inta-agent control of the EV agent in Section 6.2). By pressing the *Negotiation* button (indicated by A), a dialogue window shows up (indicated by B), in which the EV driver can insert the details of a proposal to the Charging Station agent. The details of a proposal (*NegotiationObject*) are defined in our V2G/G2V ontology (see Section 5.4). As we can see, initially, the driver requested 80% desired SOC on the departure (indicated by C) and the selected recommendation which then was translated to a charging reservation to *CS_0*, has price 0.082 €/kWh. The driver now requires 100% desired SOC on the departure, for the same period, with the same price, thus he inserts his updated preferences to the interface (indicated by D) and presses the *Propose* button. This sends the command to the EV agent to initiate an automated negotiation with the Charging Station agent. The proposal contains the updated preferences of the driver as inserted to the GUI.

The negotiation decision-making algorithms used in this use case scenario, are very simple and only for demonstration purposes. The Charging Station agent, when receives a proposal, checks if the requested energy is more from this of the active reservation. If this is the case, it makes a proposal which contains the requested energy, for the same period, but with an increased price per kWh. In any other case, it rejects the proposal. The EV agent always accepts the proposals of the Charging Station agent and it does not answer anything if one of its proposals is rejected. The algorithms were inserted to the agent using the functionality design pattern presented in Section 6.3. Thus, its very easy to replace our demonstration algorithms with more sophisticated implementations.

The negotiation process is illustrated in Figure 7.13. As we can see, the messages that precede the negotiation (lines 25-27), are the ones which were exchanged from the various agents during previous use case scenario. By pressing the '*Propose* button, *EV_0* initiates the negotiation (line 25, PROPOSE message) forming the contents of her proposal using the input from the GUI. Then *CS_0* answers with a counter-proposal (line 26) which in our case, accepts the updated energy requirements of *EV_0*, but requires a higher price per kWh (see Figure 7.14 for the details of the messages), which the *EV_0* accepts (line 27, ACCEPT-PROPOSAL message). The negotiation now has successfully completed and the *CS_0*, according to the new agreement, has to send updates about its new charging schedule (if updated) and its new availability (if updated). In our scenario,

only the charging schedule is updated as the *EV_0* requires additional energy at the same period, thus the *CS_0* updates the *MD* and *IM* agents about its new charging schedule and confidence.

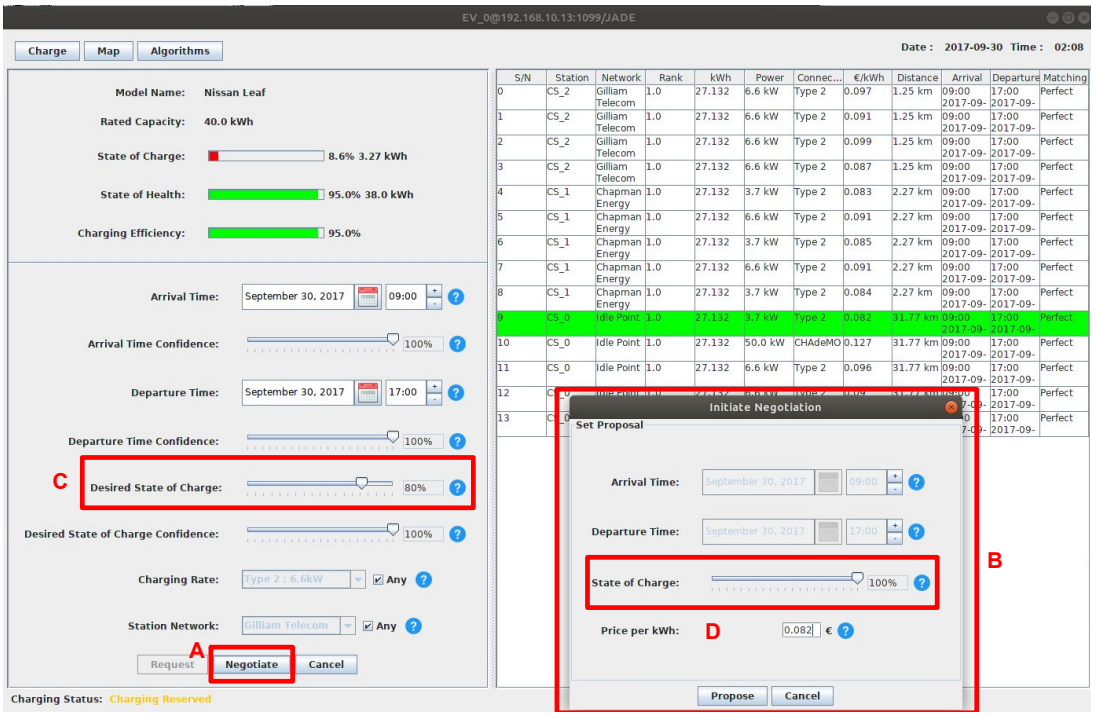

FIGURE 7.12: A screenshot of the GUI of the EV agent that illustrates where the EV driver inserts its updated preferences and how a negotiation is initiated.

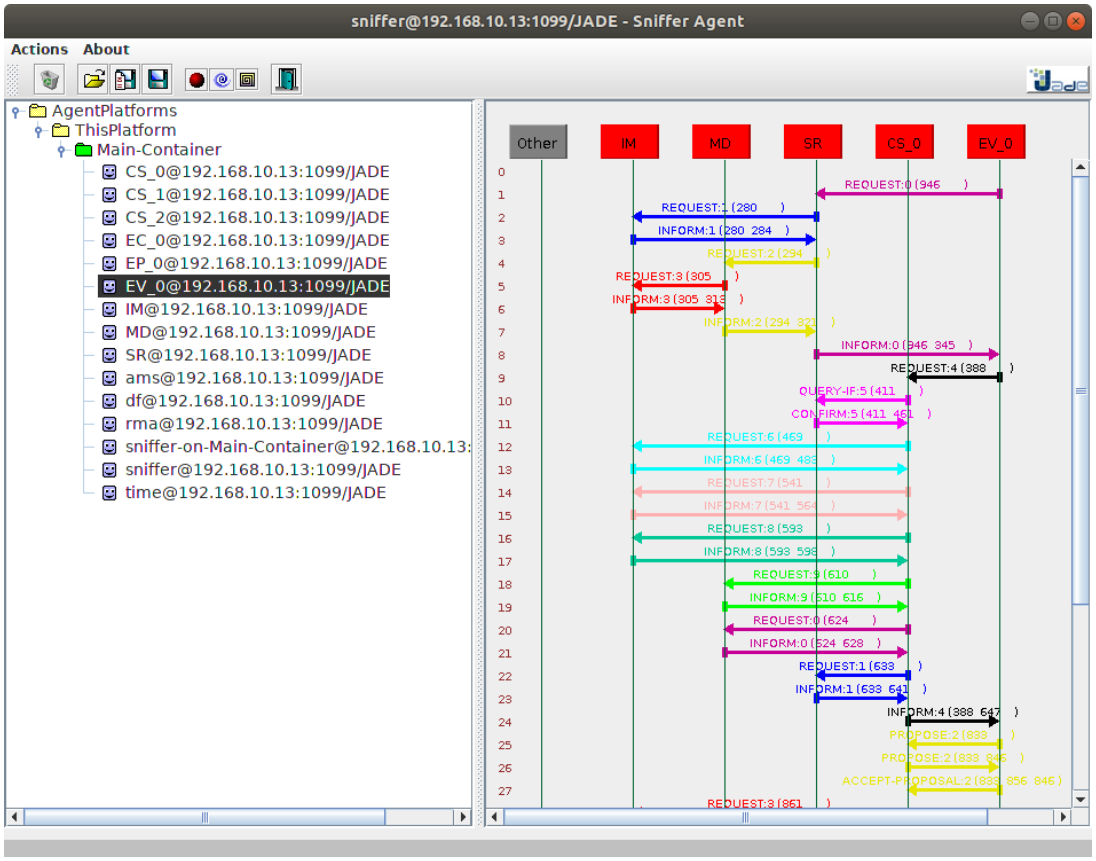

FIGURE 7.13: A screenshot of the JADE sniffer agent that illustrates a negotiation between an EV agent and a Charging Station agent.

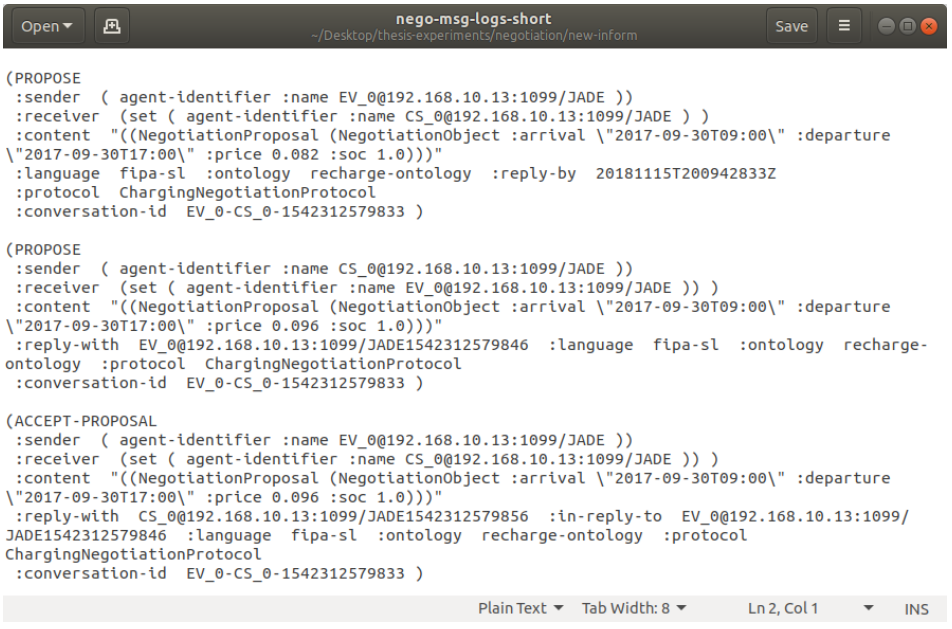

FIGURE 7.14: The contents of the negotiation messages.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In this thesis, we reviewed a considerable part of the literature that is related to the smart grid and in particular with the V2G/G2V problem with a focus to agent-based approaches. Based on a novel architecture, we implemented an integrated prototype MAS for the V2G/G2V problem. The system addresses the needs for openness, and the coverage of diverse business models via the definition of a number of key agent types, and the development of open protocols. These are freely provided along with the ontology, thus any interested parties can build their agents, implementing their capabilities given their expertise and business case.

We identified two design patterns, allowing for participating agents *(a)* to dynamically select functionalities and *(b)* to define their own implementations of abstract behaviours. The various agents we implemented in the context of this thesis show how a developer can compose the proposed protocols, the V2G/G2V ontology and the design patterns to create novel agents that form a fully operational MAS for the challenging V2G/G2V problem. For agents such as the EV agent, we implemented a GUI prototype that enables the human-agent interaction and helps the better visualization of the outputs of our system. We demonstrate the functionality and effectiveness of our system prototype on a variety of realistic use case scenarios, executed using both real-world and synthetic datasets.

8.2 Future Work

There are several directions to extend the work presented in this thesis. In this section, we will highlight four of them.

The first direction is concerned with the composition of more realistic testing scenarios and evaluation of specific state of the art algorithms related to the V2G/G2V problem. Such algorithms are for charging recommendations, EV charging scheduling, payments using mechanism design approaches and recommendation selection (utility functions). The literature that we considered and studied to identify the requirements and details of the proposed system, provides an excellent place to start. Note that the management of the various payments in our system requires the implementation of the corresponding Mechanism Design and Financial Management modules that were presented in Chapter 5.

The second direction focuses on the improvement and extension of the communication protocols that we implemented in order to capture the communication difficulties introduced in the real world. For example, if EV agents are installed in mobile devices and these devices suffer from signal loss and intermittent Internet connection, the sent messages could arrive at their receiver with severe delays, leading to efficiency losses, reduced satisfaction, and frustration. Real-time systems related with smart grid, are extremely time sensitive, thus, the extensions of the proposed protocols to predict such communication difficulties and handle them appropriately, may improve the stability of the of the grid and the satisfaction of EV drivers. Furthermore, new communication protocols have to be implemented in order to capture new requirements. For this part, we can utilize ASEME, in order to speed up the development process and for the better documentation the various architectures and protocols.

The third direction for future work includes the extension of the open-source ASEME IDE so to include the design patterns we proposed to its automatic code generation making the generated code more robust and easy to extend. As the inter-agent control model is a statechart, the ASEME IDE allows to generate code and store it in its own package. However, this feature is not connected with the intra-agent control model generation and even though code generation is automated it only generates the control code, not the action methods of the agents. Thus, in practice, all behaviours action codes must be rewritten for each agent. This, however, poses certain risks. Protocols intended flow may be disrupted by the code of programmers, or the same code must be rewritten in several packages.

As final future work direction, we consider the conversion of the Java (*Swing* and *JavaFX* based) desktop GUI prototypes developed in the context of this thesis, to Android and

web-based GUIs that are going to utilize the agents we proposed. As first priority, we consider the development of the Android application for the EV agent to facilitate the interaction between the EV driver and the EV agent. This application can be installed into the mobile of the driver or can be part of control-board of her EV or both, and will provide the EV driver with a friendly and easy to use interface to set her charging preferences and also to display the current status of her EV. A number of add-ons to support the aforementioned applications have been developed from the open-source community of JADE, and are freely available.

Appendix A

A Simple Recommendation Algorithm

Despite the fact that the development of an algorithm for charging recommendations is out of the scope of this work, we provide a very simple recommendation approach for the requirements of our use case scenarios. The source code of algorithm, which is implemented in Java, is the following:

```
public List<ChargingRecommendation> calculateStationRecommendations(ElectricVehicle
    vehicle) {

      // the charging preferences of the EV
      EVPreferences evPrefs = vehicle.getPreferences();

      // the charging types of the vehicle
      ChargingType[] types = vehicle.getBattery().getChargingTypes();

      // if charging type selected, then get only the relevant station and slots
      if (evPrefs.getChargingType() != null) {
         types = new ChargingType[] { types[evPrefs.getChargingType()] };
      }

      // all the charging stations
      HashMap<String, ChargingStation> stations0 =
    getRepoStations().getChargingStations();

      // the set of slots and stations that are available during the arrival and
      // departure period
      List<Object[]> slotNStations =
    getChargingStationSlotsReservations().searchAvailableSlots(evPrefs.getArrival(),
            evPrefs.getDeparture());
```

```
    slotNStations = filterByVehicle(stations0, slotNStations, types);

    // en empty recommendations list
    List<ChargingRecommendation> recommendations = new ArrayList<>();

    for (Object[] ss : slotNStations) {

       int slotID = (int) ss[0];
       String stationID = (String) ss[1];

       ChargingStation cs = stations0.get(stationID);
       ChargingSlot slot = cs.getChargingSlots().get(slotID);

       for (ChargingType type : types) {

          if (type.getConnectorType().equals(slot.getConnectorType())) {

             ChargingRecommendation rec = new ChargingRecommendation();
             rec.setArrival(evPrefs.getArrival());
             rec.setDeparture(evPrefs.getDeparture());
             rec.setLocation(cs.getLocation());

             double rank = calculateRank();

             rec.setRank(rank);

             double chargingRate = Math.min(type.getPower(), slot.getPower());
             double requiredKwhs = AgentUtils.requestedkWhs(vehicle);
             double kwhs = calculateTotalKWhs(chargingRate,
vehicle.getBattery().getChargingEfficiency(),
                   evPrefs.getArrival(), evPrefs.getDeparture());

             kwhs = Math.min(kwhs, requiredKwhs);

             rec.setTotalKWHs(kwhs);

             double price = calculatePrice(kwhs, slot.getNetProfitPerKwh());

             rec.setPricePerKWH(price);
             rec.setSlotID(slot.getSlotID());
             rec.setStationID(cs.getId());
             rec.setChargingPower(chargingRate);
             rec.setConnectorType(slot.getConnectorType());
             rec.setStationNetwork(cs.getStationNetwork());
             rec.setId(AgentUtils.generateUUID());

             recommendations.add(rec);
          }
```

```
        }

           // remove the station that do not match with vehicle's preferences
        if (evPrefs.getStationNetwork() != null) {
           for (Iterator<ChargingRecommendation> iter = recommendations.iterator();
    iter.hasNext();) {
              ChargingRecommendation rec = iter.next();
              if (!rec.getStationNetwork().equals(evPrefs.getStationNetwork())) {
                 iter.remove();
              }
           }
        }
      }
      return recommendations;
   }
}
```

Appendix B

Views of the EV Agent GUI

Each of the following views, represents one the top left buttons of the EV agent GUI. By pressing each button, we have the corresponding view.

B.1 Charge View

The panel of Figure B.1 is the main view of the EV agent GUI. Here the details of the state of the EV are displayed, such as the current state of charge, state of health of the battery and the name of the model. In addition, it contains an interface where the EV driver can insert her EV charging preferences, as well as a table, which illustrates the recommendations received and the recommendation selected from the EV agent.

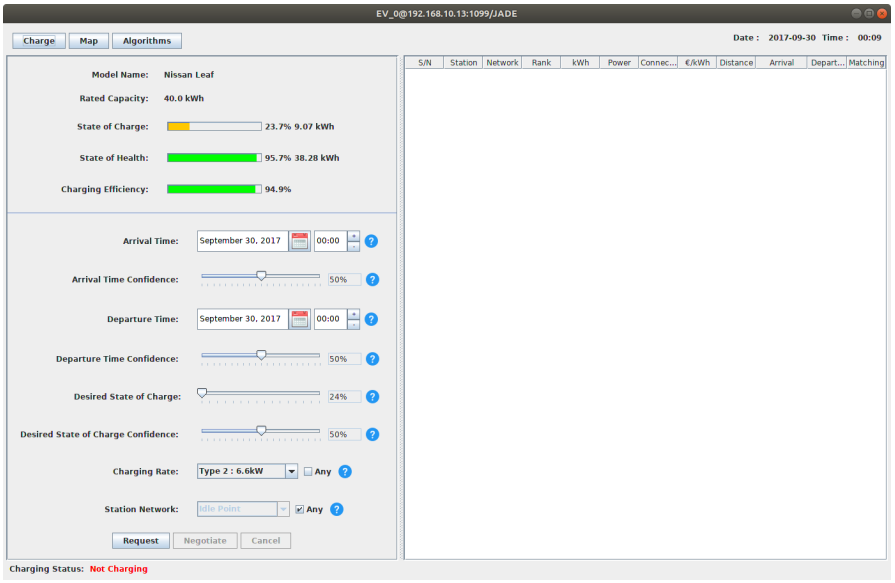

FIGURE B.1: The charge view (main) of the EV agent GUI.

B.2 Map View

The panel of Figure B.2 opens when we press the *Map* button of the EV agent GUI (top-left). It illustrates the locations of the recommended stations on the map (in our case for the city of Chania, Crete), as well as the location of our EV. Each point on the map is a special button that when pressed, shows the details of the particular charging station or vehicle.

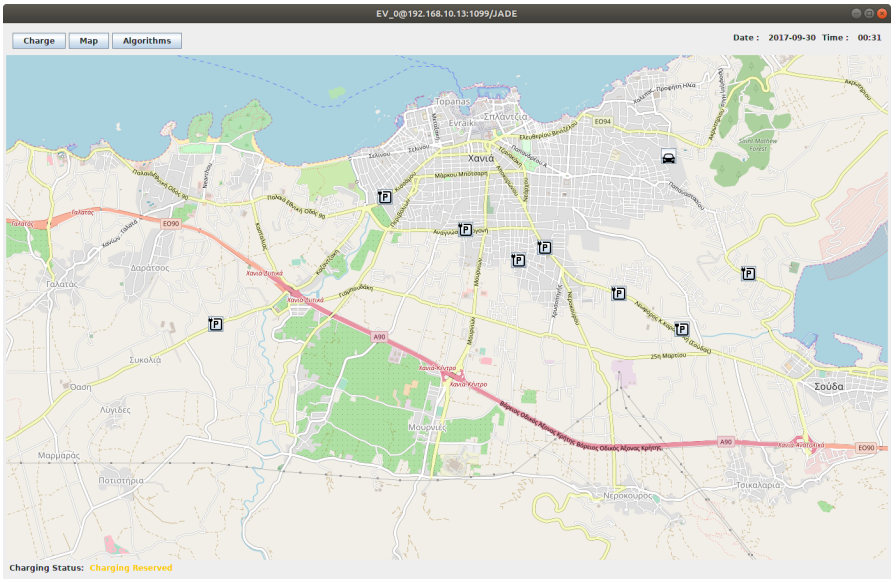

FIGURE B.2: The map view of the EV agent GUI.

B.3 Algorithms View

The panel of Figure B.3 opens when we press the *Algorithms* button of the EV agent GUI. The top view shows the algorithm selection dialogue, which opens when we press the GUI button with the three dots next to the algorithm name we want to select. The bottom view shows the dialogue that open when we select the *Utility Function: Weighted Average*, where we can insert the preferred weights of the parameters of this particular utility function.

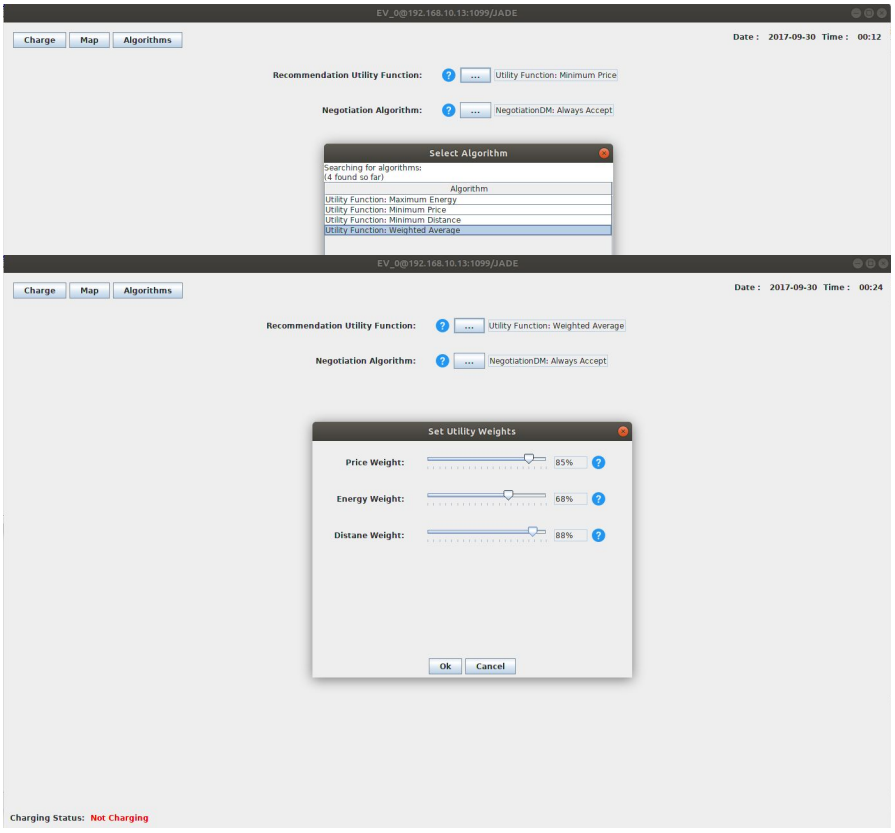

FIGURE B.3: The algorithms view of the EV agent (top), and the weights insertion panel (bottom).

Bibliography

[1] Akasiadis, C. (2017). *Multiagent Demand-Side Management for Real-World Energy Cooperatives*. PhD thesis, Technical University of Crete.

[2] Akasiadis, C. and Chalkiadakis, G. (2017a). Cooperative electricity consumption shifting. *Sustainable Energy, Grids and Networks*, 9:38 – 58.

[3] Akasiadis, C. and Chalkiadakis, G. (2017b). Mechanism design for demand-side management. *IEEE Intelligent Systems*, 32(1):24–31.

[4] Asmus, P. (2010). Microgrids, virtual power plants and our distributed energy future. *The Electricity Journal*, 23(10):72 – 82.

[5] Austin, J. L. (1975). *How to do things with words*, volume 88. Oxford university press.

[6] Babas, K., Chalkiadakis, G., and Tripolitakis, E. (2013). You are what you consume: a bayesian method for personalized recommendations. In *Seventh ACM Conference on Recommender Systems, RecSys '13, Hong Kong, China, October 12-16, 2013*, pages 221–228.

[7] Bellifemine, F. L., Caire, G., and Greenwood, D. (2007). *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. John Wiley and Sons Ltd.

[8] Borenstein, S. (2002). The trouble with electricity markets: Understanding california's restructuring disaster. 16:191–211.

[9] Caire, G. (2003). Jade tutorial: Jade programming for beginners. *http://jade. tilab. com/doc/JADEProgrammingTutorial-for-beginners. pdf*.

[10] Caragliu, A., Del Bo, C., and Nijkamp, P. (2011). Smart cities in europe. *Journal of urban technology*, 18(2):65–82.

[11] Chalkiadakis, G. (2007). *A Bayesian Approach to Multiagent Reinforcement Learning and Coalition Formation under Uncertainty*. PhD thesis, University of Toronto.

[12] Chalkiadakis, G., Elkind, E., and Wooldridge, M. (2011a). *Computational Aspects of Cooperative Game Theory*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

[13] Chalkiadakis, G., Elkind, E., and Wooldridge, M. (2012). Cooperative game theory: Basic concepts and computational challenges. *IEEE Intelligent Systems*, 27(3):86–90.

[14] Chalkiadakis, G., Robu, V., Kota, R., Rogers, A., and Jennings, N. R. (2011b). Cooperatives of distributed energy resources for efficient virtual power plants. In *10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), Taipei, Taiwan, May 2-6, 2011, Volume 1-3*, pages 787–794.

[15] Christianos, F. and Chalkiadakis, G. (2016a). Efficient multi-criteria coalition formation using hypergraphs (with application to the V2G problem). In *Multi-Agent Systems and Agreement Technologies - 14th European Conference, EUMAS 2016, and 4th International Conference, AT 2016, Valencia, Spain, December 15-16, 2016*, pages 92–108.

[16] Christianos, F. and Chalkiadakis, G. (2016b). Employing hypergraphs for efficient coalition formation with application to the V2G problem. In *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, pages 1604–1605.

[17] Cossentino, M. and Seidita, V. (2014). Passi: Process for agent societies specification and implementation. In *Handbook on Agent-Oriented Design Processes*, pages 287–329. Springer.

[18] Davito, B., Tai, H., and Uhlaner, R. (2010). The smart grid and the promise of demand-side management. 3:8–44.

[19] DeLoach, S. A. and Garcia-Ojeda, J. C. (2010). O-MaSE: a customisable approach to designing and building complex, adaptive multi-agent systems. *International Journal of Agent-Oriented Software Engineering*, 4(3):244–280.

[20] Efftinge, S. and Völter, M. (2006). oaw xtext: A framework for textual dsls. In *Workshop on Modeling Symposium at Eclipse Summit*, volume 32, page 118.

[21] European Commission (2005). Smart grids, european technology platform for the electricity networks of the future.

[22] Falvo, M. C., Sbordone, D., Bayram, I. S., and Devetsikiotis, M. (2014). Ev charging stations and modes: International standards. In *2014 International Symposium on Power Electronics, Electrical Drives, Automation and Motion*, pages 1134–1139.

[23] Ferreira, J., Pereira, P., Filipe, P., and Afonso, J. (2011). Recommender system for drivers of electric vehicles. In *2011 3rd International Conference on Electronics Computer Technology*, volume 5, pages 244–248.

[24] Franklin, S. and Graesser, A. (1996). Is it an agent, or just a program?: A taxonomy for autonomous agents. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 21–35. Springer.

[25] Gamma, E. (1995). *Design patterns : elements of reusable object-oriented software.* Addison-Wesley, Reading, Mass.

[26] Geisler, K. (2013). The relationship between smart grids and smart cities. *IEEE Smart Grid Newsletter.*

[27] Harel, D. and Naamad, A. (1996). The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(4):293.

[28] International Energy Agency (2018). Global ev outlook 2018: Towards cross-modal electrification.

[29] Jennings, N. R. (2000). On agent-based software engineering. *Artif. Intell.*, 117(2):277–296.

[30] Jennings, N. R., Faratin, P., Lomuscio, A. R., Parsons, S., Sierra, C., and Wooldridge, M. (2001). Automated negotiation: Prospects, methods and challenges. *International Journal of Group Decision and Negotiation*, 10(2):199–215.

[31] Jordán, J., Palanca, J., del Val, E., Julian, V., and Botti, V. (2018). Masev: A mas for the analysis of electric vehicle charging stations location. In Demazeau, Y., An, B., Bajo, J., and Fernández-Caballero, A., editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Complexity: The PAAMS Collection*, pages 326–330, Cham. Springer International Publishing.

[32] Kamboj, S., Kempton, W., and Decker, K. S. (2011a). Deploying power grid-integrated electric vehicles as a multi-agent system. In *10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), Taipei, Taiwan, May 2-6, 2011, Volume 1-3*, pages 13–20.

[33] Kamboj, S., Kempton, W., and Decker, K. S. (2011b). Deploying power grid-integrated electric vehicles as a multi-agent system. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '11, pages 13–20, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[34] Karapostolakis, S., Rigas, E. S., Bassiliades, N., and Ramchurn, S. D. (2016). Evlib: A library for the management of the electric vehicles in the smart grid. In Bassiliades, N., Bikakis, A., Vrakas, D., Vlahavas, I. P., and Vouros, G. A., editors, *Proceedings of the 9th Hellenic Conference on Artificial Intelligence, SETN 2016, Thessaloniki, Greece, May 18-20, 2016*, pages 13:1–13:4. ACM.

[35] Karfopoulos, E. L. and Hatziargyriou, N. D. (2013). A multi-agent system for controlled charging of a large population of electric vehicles. *IEEE Transactions on Power Systems*, 28(2):1196–1204.

[36] Kempton, W., Tomic, J., Letendre, S., Brooks, A., and Lipman, T. (2001). Vehicle-to-Grid Power: Battery, Hybrid, and Fuel Cell Vehicles as Resources for Distributed Electric Power in California. Institute of Transportation Studies, Working Paper Series qt0qp6s4mb, Institute of Transportation Studies, UC Davis.

[37] Kempton, W. and Tomić, J. (2005). Vehicle-to-grid power fundamentals: Calculating capacity and net revenue. *Journal of Power Sources*, 144(1):268 – 279.

[38] Kirschen, D. S. and Strbac, G. (2018). *Fundamentals of power system economics*. John Wiley & Sons.

[39] Labrou, Y., Finin, T., and Peng, Y. (1999). Agent communication languages: the current landscape. *IEEE Intelligent Systems and their Applications*, 14(2):45–52.

[40] Moraitis, P. and Spanoudakis, N. I. (2006). The gaia2jade process for multi-agent systems development. *Applied Artificial Intelligence*, 20(2-4):251–273.

[41] Myerson, R. (1991). *Game Theory: Analysis of Conflict*. Harvard University Press.

[42] Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V. (2007). *Algorithmic game theory*. Cambridge University Press.

[43] Noy, N. F., McGuinness, D. L., et al. (2001). Ontology development 101: A guide to creating your first ontology.

[44] Papadopoulos, P., Jenkins, N., Cipcigan, L. M., Grau, I., and Zabala, E. (2013). Coordination of the charging of electric vehicles using a multi-agent system. *IEEE Transactions on Smart Grid*, 4(4):1802–1809.

[45] Paris Agreement (2015). United nations framework convention on climate change. *Paris, France*.

[46] Rahwan, I., Ramchurn, S. D., Jennings, N. R., McBurney, P., Parsons, S., and Sonenberg, L. (2003). Argumentation-based negotiation. *Knowledge Eng. Review*, 18(4):343–375.

[47] Ramchurn, S. D., Huynh, D., and Jennings, N. R. (2004). Trust in multi-agent systems. *The Knowledge Engineering Review*, 19(1):1–25.

[48] Ramchurn, S. D., Polukarov, M., Farinelli, A., Truong, N. C., and Jennings, N. R. (2010). Coalition formation with spatial and temporal constraints. In *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, May 10-14, 2010, Volume 1-3*, pages 1181–1188.

[49] Ramchurn, S. D., Vytelingum, P., Rogers, A., and Jennings, N. R. (2012). Putting the 'smarts' into the smart grid: a grand challenge for artificial intelligence. *Commun. ACM*, 55(4):86–97.

[50] Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors (2011). *Recommender Systems Handbook*. Springer.

[51] Richardson, D. B. (2013). Encouraging vehicle-to-grid (v2g) participation through premium tariff rates. *Journal of Power Sources*, 243:219 – 224.

[52] Rigas, E. S., Karapostolakis, S., Bassiliades, N., and Ramchurn, S. D. (2018). Evlibsim: A tool for the simulation of electric vehicles' charging stations using the evlib library. *Simulation Modelling Practice and Theory*, 87:99–119.

[53] Rigas, E. S., Ramchurn, S. D., and Bassiliades, N. (2015). Managing electric vehicles in the smart grid using artificial intelligence: A survey. *IEEE Trans. Intelligent Transportation Systems*, 16(4):1619–1635.

[54] Robu, V., Gerding, E. H., Stein, S., Parkes, D. C., Rogers, A., and Jennings, N. R. (2013). An online mechanism for multi-unit demand and its application to plug-in hybrid electric vehicle charging. *J. Artif. Intell. Res.*, 48:175–230.

[55] Robu, V., Kota, R., Chalkiadakis, G., Rogers, A., and Jennings, N. R. (2012). Cooperative virtual power plant formation using scoring rules. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*

[56] Robu, V., Vinyals, M., Rogers, A., and Jennings, N. (2017). Efficient buyer groups with prediction-of-use electricity tariffs. *IEEE Transactions on Smart Grid*, pages 1–1.

[57] Roche, R., Lauri, F., Blunier, B., Miraoui, A., and Koukam, A. (2013). *Multi-Agent Technology for Power System Control*, pages 567–609. Springer London, London.

[58] Seitaridis, A., Rigas, E. S., Bassiliades, N., and Ramchurn, S. D. (2015). Towards an agent-based negotiation scheme for scheduling electric vehicles charging. In *Multi-Agent Systems and Agreement Technologies - 13th European Conference, EUMAS*

2015, and Third International Conference, Athens, Greece, December 17-18, 2015, pages 157–171.

[59] Spanoudakis, N. (2009). *The Agent Systems Engineering Methodology (ASEME)*. PhD thesis, Paris Descartes University.

[60] Spanoudakis, N. and Moraitis, P. (2008). The agent modeling language (amola). In Dochev, D., Pistore, M., and Traverso, P., editors, *Artificial Intelligence: Methodology, Systems, and Applications*, pages 32–44, Berlin, Heidelberg. Springer Berlin Heidelberg.

[61] Spanoudakis, N. I. and Moraitis, P. (2010). Modular JADE agents design and implementation using ASEME. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2010, Toronto, Canada, August 31 - September 3, 2010*, pages 221–228.

[62] Spanoudakis, N. I. and Moraitis, P. (2015). Engineering ambient intelligence systems using agent technology. *IEEE Intelligent Systems*, 30(3):60–67.

[63] Sundström, O. and Binding, C. (2012). Flexible charging optimization for electric vehicles considering distribution grid constraints. *IEEE Trans. Smart Grid*, 3(1):26–37.

[64] Trillo, R., Ilarri, S., and Mena, E. (2007). Comparison and performance evaluation of mobile agent platforms. In *Autonomic and Autonomous Systems, 2007. ICAS07. Third International Conference on*, pages 41–41.

[65] Uschold, M. and Gruninger, M. (1996). Ontologies: Principles, methods and applications. *The knowledge engineering review*, 11(2):93–136.

[66] US Department of Energy (2003). "Grid 2030" — a national vision for electricity's second 100 years.

[67] Valogianni, K., Ketter, W., Collins, J., and Zhdanov, D. (2014). Effective management of electric vehicle storage using smart charging. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 472–478.

[68] Van Aart, C., Pels, R., Caire, G., and Bergenti, F. (2002). Creating and using ontologies in agent communication. In *Proceedings of the Workshop on Ontologies in Agent Systems*.

[69] Vayá, M. G. and Andersson, G. (2012). Centralized and decentralized approaches to smart charging of plug-in vehicles. In *2012 IEEE Power and Energy Society General Meeting*, pages 1–8.

[70] Vinyals, M., Bistaffa, F., Farinelli, A., and Rogers, A. (2012a). Coalitional energy purchasing in the smart grid. In *2012 IEEE International Energy Conference and Exhibition (ENERGYCON)*, pages 848–853.

[71] Vinyals, M., Bistaffa, F., Farinelli, A., and Rogers, A. (2012b). Stable coalition formation among energy consumers in the smart grid. In *Proceedings of the 3rd International Workshop on Agent Technologies for Energy Systems (ATES 2012)*. Citeseer.

[72] Vinyals, M., Robu, V., Rogers, A., and Jennings, N. R. (2014). Prediction-of-use games: a cooperative game theoryapproach to sustainable energy tariffs. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 829–836. International Foundation for Autonomous Agents and Multiagent Systems.

[73] Wooldridge, M. (1997). Agent-based software engineering. *IEE Proceedings-software*, 144(1):26–37.

[74] Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152.

[75] Wooldridge, M. J. (2002). *Introduction to multiagent systems*. John Wiley and Sons Ltd.

[76] Zanella, A., Bui, N., Castellani, A., Vangelista, L., and Zorzi, M. (2014). Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1):22–32.