An Ontology Driven approach for modeling a Multi Agent Based Electricity Market

Geovanny Poveda and René Schumann

University of Applied Sciences Western Switzerland (HES-SO VS),
Rue de Technopole 3, 3960 Sierre, Switzerland
{geovanny.poveda,rene.schumann}@hevs.ch
http://silab.hevs.ch

Abstract. Model Driven Development has prompted domain experts to use high-level languages for specifying, testing, verifying and code final application. While Model Driven Development has had a significant contribution to the development of Multi-Agent-Based Simulation, it becomes more and more important to develop appropriate means for representing the multi-disciplinary domain expertise dimension of complex adaptive systems such as deregulated electricity markets. This paper proposes an approach for modeling and simulating agent-based models on the basis of an Ontology-Driven Conceptual Modeling approach in a quite generic scenario for modeling electricity markets. The contribution of this paper is focused on the creation of set of methods, mechanisms and tools to support the design, implementation and experimentation of agent-based models from the instantiation of an ontology-based conceptual modeling.

Keywords: Model driven, ontology driven, simulation, conceptual model, multi agent based simulation

1 Introduction

Over the last years, several lines of research have allowed deregulated electricity markets to be studied. Among others, Multi-Agent Based Simulation (MABS) has been one of the most prominent approach for simulating decentralized electricity markets. While MABS has had a significant contribution to the deeper understanding of complexity in the electricity market and guide the market practices in the real world, it becomes more important to develop appropriate means for capturing and representing the cross-disciplinary domain expertise dimension of complex social systems. Model-Driven Development (MDD) has become an important approach in software discipline to link design and code. It is a software engineering paradigm which uses models as a mean for specifying, testing, verifying and *generating* code for a final application [13].

While MDD has enabled modelers to support the creation of agent-based models from a conceptual design, it becomes more and more important to develop appropriate means for improving the short-term design of conceptual models in terms of how much they can be extended from a cross-disciplinary domain expertise dimension. Ontology-Driven Conceptual Modeling (ODCM) uses ontologies to drive the creation of simulation models and in doing so makes use of an agreed upon set of terms and relationships that are shared by domain experts, modelers, and model development tools. Most importantly, ontologies provide foundations for enabling the extension of structural changes on models, as well as, their automated extension.

In this article we introduce an integrated solution for modeling and simulating agent-based models on the basis of an ODCM approach. The contribution of this paper is focused on the creation of a model to support the automatic implementation of agent-based models from the instantiation of an ontology-based conceptual modeling. The structure of this paper is as follows: Section 2 provides an overview into MDD approach for Multi-Agent System technologies. Section 3 introduces our proposed ODCM approach by describing the conceptual model and simulation design phases. Section 4 describes the organizational architecture of the MABS model. Next, Section 5 illustrates the advantages of the contribution presented by a proof of concept. Finally, Section 6 concludes and gives possible future research directions.

2 Background

Several studies for modeling Multi-Agent Systems by using Model Driven Development have been already proposed [5,14,6,7]. A comprehensive comparison of these studies is out of the scope of this paper. However, a short overview on most representative related works is given in the remainder of this section. In [7] Garro and Russo have presented easyABMS, a full-fledged methodology for the agent-based modeling and simulation of complex systems. The approach of such study relies on both Agent-Oriented Software Engineering (AOSE) modeling techniques and simulation tools for ABMS. easyABMS is focused on system modeling and simulation analysis rather than details related to programming and implementation as it exploits the model-driven paradigm, making it possible the automatic code generation from a set of (visual) models of the system. In [6], the authors propose the jointly exploitation of both Platform-Independent Meta-models and Model-Driven approaches to define a Model-Driven process named MDA4ABMS. The stated process is conforms to the OMG Model-Driven Architecture (MDA) and enables the definition of Platform-Independent simulation models from which simulation models and their code can be automatically obtained.

In [14] Candelaria and Pavón propose a high-level conceptual modeling abstraction for simulation development, it includes transformation tools that facilitate the implementation of simulations on different simulation platforms. The framework presented is based on the MDA pattern by means of a platform-independent modeling language, a code generation process and templates, and specific simulation platforms. In [5] the authors have present a model to bridge the gap between the design and the application of Agent oriented systems. They demonstrated how the MDA could be used to derive practical applications of

Agents from Agent oriented design. Their major contributions were in the definition of a common, agent-neutral model that applies to all the concepts required by the FIPA-compliant agent platform.

Another line of research has highlighted the importance of using ontologies for supporting the implementation of agent-based simulation through the designing of conceptual model. In [16] Ying et al. describe MOMA, a methodology for ontology-based multi-agent application development which focuses on the development of an ontology as driving force of the development of Multi Agent Systems for experimentation in the financial domain. MOMA consist of two main development phases: Ontology Development and Agent Development. In the first stage concepts and relations in the domain are identified and they are modeled for a specific application, then code that can be used in the agent development phase is generated.

3 An Ontology-Driven modelling approach for Agent-Based Simulations

Following the approach mentioned in Section 1, an ontology-based model driven has been designed. The proposed model employs a single and integrated formalism to define the MABS dimensions, in particular, the design of conceptual modeling and the generation of agent-based models from instances of a conceptual model. The basis of the stated model includes two major phases: (i) conceptual modeling; and (ii) simulation design.

3.1 Conceptual Modelling Phase

In this phase, a set of high level abstraction of concepts and relations have to be created for describing the domain in which MABS models will be defined in. The basis of the proposed conceptual modeling includes an ontology which describes the agent organizational structure of simulated electricity market models (capabilities, behaviors, beliefs, actions), as well as, the vocabulary for describing the set of fundamental concepts around the electricity market domain. The Electricity Market Ontology (EMO) has been created for describing the negotiation mechanisms, actors, profiles and market dynamics.

3.2 Simulation Design Phase

The simulation design phase enables mechanisms for the automated construction of agent-based models from instances of the ontology-based conceptual modeling. In this phase, a set of transformation rules, directives and design patterns have been created to transform the axioms from the ontology-based conceptual modeling (classes, object properties, properties, properties restrictions) to elements of a MABS model (classes, attributes, associations, roles, plans, etc.), and deploy it as an agent model. Figure 1 shows the components of the stated phase. To derive agent models from the ontology-based conceptual modeling, two models have to be performed:

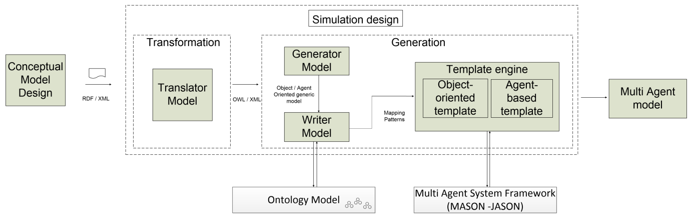


Fig. 1: Simulation design phase overview

– Transformation model: In this model, axioms from the conceptual model are classified into classes, object properties, and data properties and then, the OWL API [8] is used for translating Resource Description Framework (RDF) subjects into OWL classes that relate predicates and objects as property restrictions. The set of transformation rules used for keeping the dependence between subjects and their properties are: (i) objects are considered as part of a `<owl:Restriction>` property through the use of the `OWLObjectSomeValuesFrom` interface; and (ii) predicates are considered as a restriction of subjects through the use of the `getOWLSubClassOfAxiom` interface. Listing 1.1 shows a section of the algorithm used for transforming the model.

Listing 1.1 A section of the algorithm used for defining the transformation model.

```
String subLabel = subject.toString();
OWLClass owlclass1 = owlfact.getOWLClass(IRI.create(subLabel));

createOWLDeclarationAxiom(owlclass1);
OWLClass owlclass2 = owlfact.getOWLClass(IRI.create(objLabel));
createOWLDeclarationAxiom(owlclass2);
OWLObjectProperty hasRelation = owlfact.getOWLObjectProperty(IRI.create(predicate.toString()));

//create OWL Pattern
OWLClassExpression hasRelationSome = owlfact.getOWLObjectSomeValuesFrom(hasRelation,owlclass2);
OWLSubClassOfAxiom axiomPattern = owlfact.getOWLSubClassOfAxiom(owlclass1,hasRelationSome);
owlmanager.addAxiom(ont,axiomPattern);

public void createOWLDeclarationAxiom(OWLClass owlclass){
OWLDeclarationAxiom declarationAxiom = owlfact.getOWLDeclarationAxiom(owlclass);
owlmanager.addAxiom(ont, declarationAxiom);
}
```

– Generation Model: In this model, a set of operations are used to write the program structure of the agent-based model in terms of their organizational structure. The basis of the stated model includes an object generator model and rule-based template engine.

Generator Model. This model provides mechanisms for defining the high level structure of artifacts (packages, classes, functions and relations) to be involved in the serialization of files. It uses the transformed OWL-based conceptual modeling and executes a set of mapping rules for representing the ontological axioms into an internal model named Jmodel. A Jmodel is an intermediate programmatic representation created to perform the expressiveness of OWL axioms into Java oriented models. Among others, multiple inheritance, properties without type and inverse properties are the most important. A Jmodel consist of a set of instanced Java classes and Java objects assigned to specific packages instances.

In order to build the instances of Java classes, resources from the Jena data structure [3] are located, and for all instances of that class, Java interfaces and their respective implementing Java class are generated. In order to express the multiple inheritance of OWL axioms, interfaces are embedded inside their corresponding Java classes and they are enabled to be included into an object-oriented hierarchy composed of sub and supper-class relations. Once classes are defined, both, object and data properties are retrieved from resources and they are translated into object-oriented artifacts depending on the type of property. In the case of object properties, that is, relation of a class to another class, instantiated object-oriented methods are created recursively. Given that OWL properties can be of multiple types, a list of multiple generic objects have to be created for ensuring that object properties can be used in the specified range. When annotation properties come from a `<rdf:datatype>` property, that is literals, they are transformed as global object-oriented attributes with accessors methods (get and set) on the related OWL class.

Afterwards, subclasses from the Jena data structure resource are identified and their cardinalities and values restrictions are checked. To be ensure constraints are satisfied, the range of the object properties acting as `<owl:OnProperty>` property are associated to a Java interface named `RestrictionManager` to be exposed to the classes acting as `<owl:OnClass>` property. Thus, classes acting as restrictions can be instanced though a synchronized `getByName()` method that use the Java reflection mechanism. Figure 2 shows the mechanisms involved in the generator model.

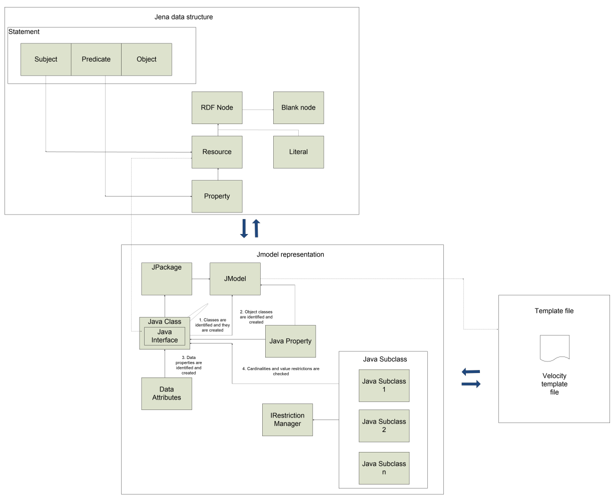


Fig. 2: Generator model overview

The object / agent template engine. To perform the process of writing the programmatic structure of the model (skeleton, constructor, methods, and headers) a rule-based template engine is involved. Template files use a set of high level programming structures that describe the syntactic and semantic structure to be used for serializing the model. Syntactic descriptions are used for describing the skeleton structure of classes and headers. Semantic structures are used for describing the methods and functions by using global identifiers from the stated `Jmodel`. The use of templates files help to keep the transformation rules out of the application code. Thus, new templates can be added and existing ones can be customized without modifying the application code.

Listing 1.2 shows how syntactics and semantics structures are used for describing the content of Java classes. Syntactic structures are created by using pre-defined language expressions for the Java identifiers and modifiers. Note that semantic structures are created by using global identifiers variables that use the Java reflection mechanism to create methods dynamically from values of the Jmodel. In the case of object properties, a `for-each` velocity [1] statement enable the creation of Java object iterators and lists by using the global `jmodel.listclasses` variable, which contains the set of transformed classes from the conceptual model.

Listing 1.2 A section of a template file used for the generic-object oriented model.

```
public class \$cName extends IndividualImpl implements \$iName {

    // syntactic structure
    public static void main(String[] args)
     {

     }
     // semantic / syntactic structure
#foreach (\$cls in \$jmodel.listJClasses())
                       #set(\$clsName = \$cls.getJavaClassName())
                         List<\${clsName}> \${clsName} = new ArrayList<\${clsName}>();
             \${clsName} temp\${clsName} = new \${clsName}();
             Iterator<\${clsName} > \${clsName}Itr;
```

As stated before, EMO Ontology provides a vocabulary that describes the set of organizational agent building blocks and their relations with the fundamental concepts around the electricity market. To provide the object-oriented model the ability to be described in terms of their capabilities, behaviors, plans, beliefs, interactions, protocols and communication mechanisms, a set of transformation rules based on the previously mentioned template engine have been involved. To describe how classes can be controlled and scheduled to access to the resources and services of the model, a velocity template file is used to include the set of fundamental functions and methods from the MASON API [11]. The stated file describes syntactic structures that include the `Stepabble` interface [11] as well as, its hook method named `Step` [11]. Thus, agents are able to reason at runtime the changes and updates on the environment.

[1] http://velocity.apache.org/

To provide a more understandable approach to enable the communication among agents, the `SimState` object [10] has been involved to represent the overall model. The `SimState` object is created to communicate agents in the model via `getter` methods of the stated `step` method. The model object serves as a communication device by holding information need by other agents. Finally, to describe the behavioral, belief and capabilities descriptions, classes and interfaces from the JASON API [2] are included. There is a template file which specify the structure of the agent model in terms of the agent speak annotations. Listing 1.3 shows how the agent organizational is included. Note that an interface has been created for providing classes the ability to extend the overall agent architecture of the JASON API.

Listing 1.3 A section of a template file used to involve agent capabilities

```
package \$testcasePkg;
  import java.util.Iterator;
  import com.hp.hpl.jena.ontology.OntModel;import com.hp.hpl.jena.ontology.Ontology;
  import com.hp.hpl.jena.rdf.model.ModelFactory;import java.util.ArrayList;
  import java.util.List;

  public class \$electricityMarketSimulation  extends \$SimState implements \$Steppable {
  private static String namePrefix = "ClassInstance";
  private static int nameCount = 0;

  public \$electricityMarketSimulation(final long seed) {
              super( seed );
        }
        @Override
        public void step(final SimState state) {

              OntModel ontModel = ModelFactory.createOntologyModel();
              Ontology ontology = ontModel.createOntology(base);

              #foreach (\$cls in \$jmodel.listJClasses())
                      #set(\$clsName = \$cls.getJavaClassName())
                       List<\${clsName}> \${clsName} = new ArrayList<\${clsName}>();
                      \${clsName} temp\${clsName};
                      Iterator<\${clsName} > \${clsName}Itr;
                      Iterator<\${clsName} > \${clsName}Itr;
        }

        public JasonAgent(String id, String aslFilePath, Logger logger)
            throws SimulationException {
        this.id = id;
        this.logger = logger;
        this.message = new ArrayList<Message>();
        this.actions = new HashMap<String, Class<? extends JasonAgentAction>>();
        this.configActions();
        try {
            agent = new Agent();
            agent.init();
            agent.load(aslFilePath);
        } catch (JasonException e) {
          ....................
        }
    }
```

4 Multi Agent System Organizational Structure

This section describes the organizational architecture of the MABS model to be generated. It describes a set of building blocks for defining the role, behaviors, capabilities and interactions of the agents on models. Such structure, has been defined on the basis of an agent-based interaction model, where an agent describes the capabilities that electricity market players have to perform for negotiation and the roles they play during the execution of bilateral contracts. In that structure, agents are autonomous entities able to update their negotiation knowledge by perceiving information from their market (environment) and performing actions from the organization (norm, mission, cooperation, goals). Figure 3 shows the concepts used for defining the agent organizational structure of our model.

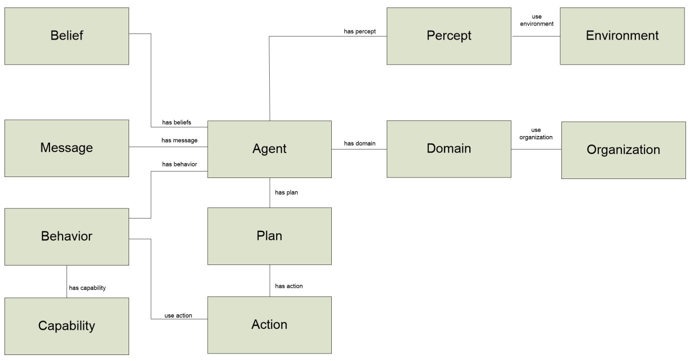


Fig. 3: Agent organizational structure of model

4.1 Agent Model

The agent concept describes the capabilities agents have to perform negotiation and the roles they play during the execution of bilateral contracts. In our model, agents are autonomous entities able to update their negotiation knowledge by performing actions from the organization (cooperation, organization) and perceiving information from their market (resource). Agents update their plans by performing roles and behaviors that depends the context the agent is acting and the task to be achieved.

4.2 Role Model

The role model provides a definition of the roles and domains to be employed for modeling the electricity market. The role model involves the actor definition, a generic concept used for representing the parties that take places in the bilateral transactions. It also involves the domain definition, a concept used for delimiting the areas of negotiation for a particular purposes (consumption, distribution, retailing). Last but not least, a role describe itself the external negotiation interactions with other players / parties in relation to the goal to be achieved given a bilateral transaction.

4.3 Interaction Model

The interaction concept describes how the intercommunication among agents take place. To model agent interactions, a two-level system structure has been involved. In such structure, the highest level enables the interaction among agents each other and between agents and the environment through roles. Lowest level concerns the environment. An interaction takes place when an agent performs an action and such action is translated into an event that is notified to another agent that exhibits the specific behavior [12]. The meta model of the interaction concept is depicted in Figure 4. It is important to note that in our model, an interaction it is not a just a *message passing*, it also includes protocol, a message scope and actor definitions. In this aspect, the actors interact with the protocol through a set of messages that are exchanged by the concerned parties.

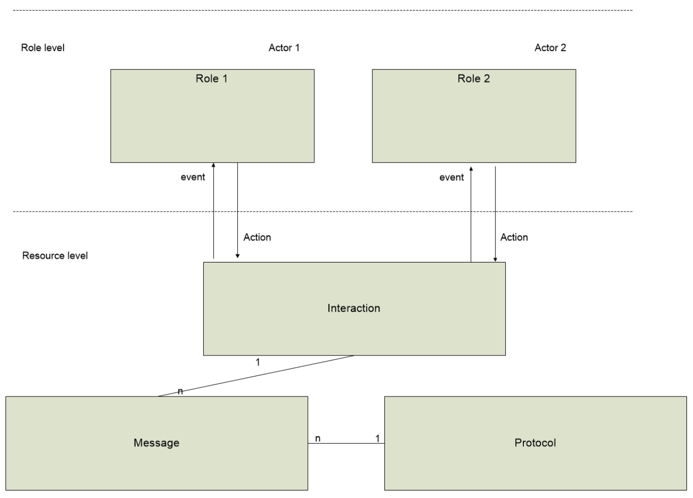


Fig. 4: Meta model of the concepts used in the interaction aspect specification.

4.4 Organization Model

The organization model describes how the system organizations of electricity retailer market is modeled and their relationships with another market system. It describes how players from the electricity retailer market can cooperate and participate in the market by considering their roles and capabilities. Organizational model also defines how the negotiation mechanisms of bilateral contracts can be defined in terms of their interactions (communication). It involves protocols, a concept used to establish a standard to interchange message during the negotiations.

4.5 Behaviour Model

The behavior concept describes how the plans are performed by players of the electricity retailer market and how the information flows among such actors. In this aspect, a behavior describes an abstract representation that connects the players with their behavioral features. Behaviors provide the basis of specifying the internal activities of players through plans. A plan is a specialization of the behaviors which defined a set of flows and activities. In this aspect, the activities represent the actions to be performed in the plans and they are linked throw flows [15].

4.6 Environment Model

The environment concept describes the resources that can be dynamically created or shared by the agents in the electricity retailer market. In this aspect, features for modeling how offers and contracts can be accessed have been considered. Environment concept is responsible of enabling and controlling the access to the market and some negotiation services by using concrete means for agents. It provides a set of inherited elements: observable and state to enable agents to have access to resources and services. Thus, agents are able to reason at run-time a new environment they are discovering [4].

5 Proof of Concept

This section presents how our proposed approach can be applied. We provide here an example for modeling a strictly simplified negotiation in the electricity retail market. In particular, a scenario for a simulated deregulated electricity market focusing on bilateral negotiations [9] between a single retailer and multiple customers. The example to be presented in that section is focused on show how the model proposed in Section 3 can be used for simulating agent-based models from an ontology-based conceptual model. In this scenario, customers negotiate a time of use tariff [1] under a set of specified terms and conditions, including energy price, energy quantity and duration. Note that the proposed proof of concept provides the foundations for showing how generic agent-based models explained in section 3 can be generated rather than fully agent-based models. In the following subsections, the processes involved in each phase are presented.

Proof of Concept - Conceptual Modelling Phase In this step, we specify the concepts in the target of the scenario previously detailed and describe it in the in the ontology-based conceptual model. In a first stage, the conceptual model is explored and high hierarchy classes are identified to perform the correspondence between the conceptual model definitions and the concepts distinguished by domain experts during the analysis stage. By using the Protégé tool [2], domain experts describe the classes, properties, data properties and individuals to be involved in the agent-based electricity model. Such description includes participants, market features and strategic descriptions. The following types of participants have been considered:

- Retailer: represents the business units that sell energy to a retail market.
- Customer: represents the players buying energy in the retail market. It includes households, commercial, industrial and other electricity consumers.
- Market: represents the resources created and shared by agents on the model. It includes aspects such as contracts, trades, orders, bids, others.

Next, we describe some fundamental concepts around the bilateral negotiations. Concepts such as market operation, methods of negotiation, as well as strategic behavior of participants have been included:

- Bilateral negotiation methods: represents the set of negotiation methods to be used. It includes (i) request for bid, (ii) offer method and (iii) announce reward table.
- Protocol: represents the set of message flow that specify how the exchange of messages is processed. It includes stacked alternating offer protocol and monotonic conseccion protocol.
- Strategy: represents the set of strategies to be used. It includes logrolling and compensation.

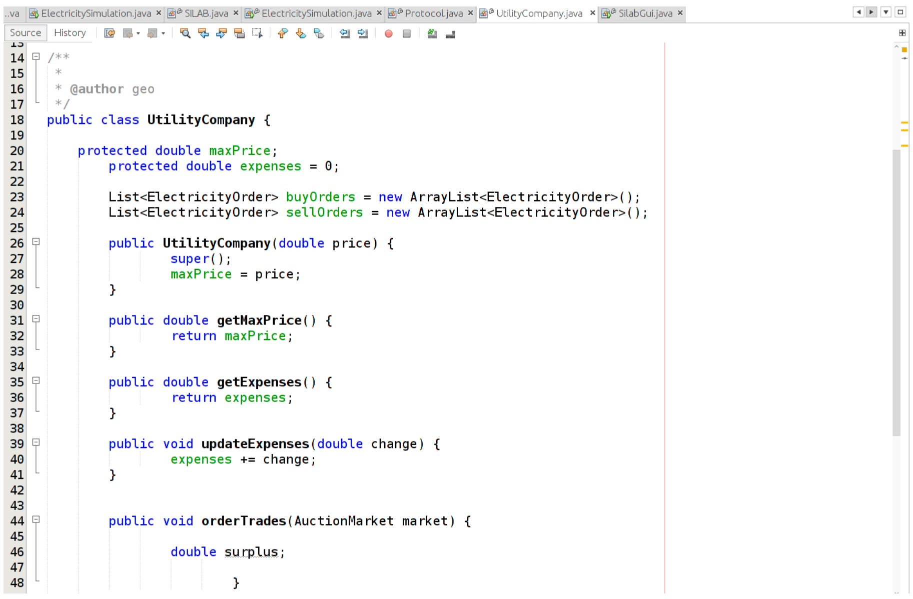
Proof of Concept - Simulation Design Phase In this step, the agent-based electricity model is generated from instances of the previously designed conceptual model. To transform the conceptual model into an agent-oriented generic model, we created a software utility to use the functions and methods proposed in the simulation design phase (see Section 3.2). Listing 1.4 shows the main two activities executed during the transformation model. In a first stage, the RDF model is transformed into OWL axioms and rewritten to RDF/XML syntax. Next, methods from the generator model are instanced and the RDF/XML model is transformed into a object-oriented generic model. Finally, the generated classes should be placed in the corresponding package and directory.

Figure 5 shows how the generic object-oriented model is generated automatically after the execution of the stated class. Note that concepts and objects properties from the conceptual modeling haven been transformed into JAVA classes. Note that created classes do not involve the organizational structure of agents since the writing activity of the model has been based on the object-oriented template. Next section shows how the agent-oriented model is created.

[2] http://protegewiki.stanford.edu/wiki/

Listing 1.4 A class to create the generic object-oriented model.

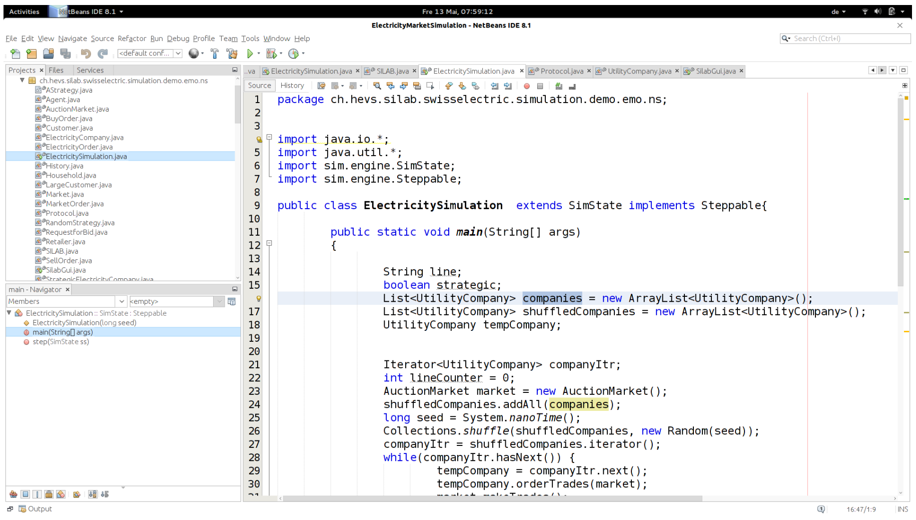
```
 public  class  ElectricityMarketSimulation  {

     public  static  void  main(String[]  args)  {
                 try  {
                             File  file  =  new  File("emordfinstances.rdf");

                             String  fileName  =  file.getName();
                             System.out.println(fileName  );

                             ConvertRDFtoOWL  convert=   new   ConvertRDFtoOWL();
                             convert.transformRDFtoOWL(fileName  );

                             File  source  =  new  File("emordfinstances.owl/");
                             File  dest  =  new  File("emotransformed.owl/");
                             try  {
                                         FileUtils.copyFile(source,  dest);
                                 }  catch  (IOException  e)  {
                             e.printStackTrace();
                             }

                             ConvertOWLSyntax  converter  =  new  ConvertOWLSyntax();
                             converter.SyntaxRewrite("http://silab.hevs.ch/projects/simulation/testing/emotransformed.owl&format=RDF/XML");

                             OntModel  ontModel  =  OntologyUtils.loadOntology("file:emo.owl");
                             JenaArtefacts  artefacts  =  new  JenaArtefacts();
                             artefacts.generate(ontModel,  "src",  "ch.hevs.silab.swisselectric.simulation.demo.emo.ns");
                 }
         }
}
```



```
/**
 *
 * @author geo
 */
public class UtilityCompany {

    protected double maxPrice;
        protected double expenses = 0;

        List<ElectricityOrder> buyOrders = new ArrayList<ElectricityOrder>();
        List<ElectricityOrder> sellOrders = new ArrayList<ElectricityOrder>();

        public UtilityCompany(double price) {
                super();
                maxPrice = price;
        }

        public double getMaxPrice() {
                return maxPrice;
        }

        public double getExpenses() {
                return expenses;
        }

        public void updateExpenses(double change) {
                expenses += change;
        }


        public void orderTrades(AuctionMarket market) {

                double surplus;

                        }
```

Fig. 5: A class of the generated generic object-oriented model.

Proof of Concept - Agent System Organizational Structure In this step the generic agent-oriented model is created. The stated model is created from the set of transformation rules previously explained in the agent template section. Agent-based model is generated in terms of the application programming interface of the MASON framework. Figure 6 shows how the main class of the package generated is expressed in terms of the MASON API. The picture also shows how the concepts and relations previously defined in the conceptual model, they have been created in the agent model.



```
package ch.hevs.silab.swisselectric.simulation.demo.emo.ns;


import java.io.*;
import java.util.*;
import sim.engine.SimState;
import sim.engine.Steppable;

public class ElectricitySimulation  extends SimState implements Steppable{

        public static void main(String[] args)
        {

                String line;
                boolean strategic;
                List<UtilityCompany> companies = new ArrayList<UtilityCompany>();
                List<UtilityCompany> shuffledCompanies = new ArrayList<UtilityCompany>();
                UtilityCompany tempCompany;


                Iterator<UtilityCompany> companyItr;
                int lineCounter = 0;
                AuctionMarket market = new AuctionMarket();
                shuffledCompanies.addAll(companies);
                long seed = System.nanoTime();
                Collections.shuffle(shuffledCompanies, new Random(seed));
                companyItr = shuffledCompanies.iterator();
                while(companyItr.hasNext()) {
                        tempCompany = companyItr.next();
                        tempCompany.orderTrades(market);
```

Fig. 6: The generated agent-based model.

6 Summary

In this article we have introduced an integration solution for modeling and simulating agent-based models on the basis of an ODCM approach. The proposed solution has been envisaged to facilitate the creation of agent-based electricity market simulations by using ontologies as the driving force of a MDD approach. We have proposed a model which merge MDD and ODCM approaches to offer a set of methods able to generate automatically agent-based models from the design of a conceptual model. We have presented and explained the phases of the model by distinguishing three main stages: (a) a conceptual model transformation; (b) an organizational agent architecture adaption; and (c) agent-based generic code generation. The model is still under development, it is currently focused to generate generic agent-oriented structures for the agent system. Future research efforts will be devoted to: (i) improving the algorithms used for translating the object-oriented generic model into agent-based simulation models. We plan to include mapping strategies that involves the automatically generation of object oriented methods from the annotation properties defined in the conceptual model. (ii) our research will also provide techniques for supporting the design and experimentation of the agent-based models by involving semantic-based rules for adapting changes in the behavior of the agent model, in such a way that conditional structures can be included on the generic agent-based model and it can generate executable agent-based models.

References

1. Hugo Algarvio, Fernando Lopes, and Joao Santana. Multi-agent retail energy markets: Bilateral contracting and coalitions of end-use customers. In *European Energy Market (EEM), 2015 12th International Conference on the*, pages 1–5. IEEE, 2015.
2. Rafael H Bordini and Jomi F Hübner. A java-based interpreter for an extended version of agentspeak. *University of Durham, Universidade Regional de Blumenau*, 2007.
3. Jeremy J Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 74–83. ACM, 2004.
4. Oguz Dikenelli, Marie-Pierre Gleizes, and Alessandro Ricci. *Engineering Societies in the Agents World VI: 6th International Workshop, ESAW 2005, Kusadasi, Turkey, October 26-28, 2005, Revised Selected and Invited Papers*, volume 3963. Springer Science & Business Media, 2006.
5. Mohamed Elammari and Zeinab Issa. Using model driven architecture to develop multi-agent systems. *Int. Arab J. Inf. Technol*, 10(4), 2013.
6. Alfredo Garro, Francesco Parisi, and Wilma Russo. A process based on the model-driven architecture to enable the definition of platform-independent simulation models. In *Simulation and Modeling Methodologies, Technologies and Applications*, pages 113–129. Springer, 2013.
7. Alfredo Garro and Wilma Russo. easyabms: A domain-expert oriented methodology for agent-based modeling and simulation. *Simulation Modelling Practice and Theory*, 18(10):1453 – 1467, 2010. Simulation-based Design and Evaluation of Multi-Agent Systems.
8. Matthew Horridge and Sean Bechhofer. The owl api: A java api for owl ontologies. *Semantic Web*, 2(1):11–21, 2011.
9. Filipa Lopes, Cristina Ilco, and Jorge Sousa. Bilateral negotiation in energy markets: Strategies for promoting demand response. In *European Energy Market (EEM), 2013 10th International Conference on the*, pages 1–6. IEEE, 2013.
10. Sean Luke. Multiagent simulation and the mason library. *George Mason University,*, 2011.
11. Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527, 2005.
12. Robert Meersman, Zahir Tari, et al. *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*. Springer, 2002.
13. Juan Pavón, Jorge Gómez-Sanz, and Rubén Fuentes. Model driven development of multi-agent systems. In *Model Driven Architecture–Foundations and Applications*, pages 284–298. Springer, 2006.
14. Candelaria Sansores and Juan Pavón. Agent-based simulation replication: A model driven architecture approach. In *MICAI 2005: Advances in Artificial Intelligence*, pages 244–253. Springer, 2005.
15. Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

16. Wier Ying, Pradeep Ray, and Lundy Lewis. A methodology for creating ontology-based multi-agent systems with an experiment in financial application development. In *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, pages 3397–3406. IEEE, 2013.