THE POTENTIAL OF OBJECT-ORIENTED ANALYSIS AND DESIGN FOR AGENT-BASED COMPUTATIONAL ECONOMICS

Tuong Manh Vu, Peer-Olaf Siebers, Christian Wagner
School of Computer Science
University of Nottingham
Nottingham, UK
E-mail: {psxtmvu, Peer-Olaf.Siebers, Christian.Wagner}@nottingham.ac.uk

KEYWORDS
Object-Oriented Analysis and Design, Agent-based Modelling and Simulation, Agent-based Computational Economics

ABSTRACT

Agent-based Simulation is used for different purposes in Computer Science and Economics. While computer scientists design agents for controlling complex systems or for instilling objects with autonomy and intelligence, economists use agents for gaining a better understanding of the dynamics within economic systems. For the modelling task both communities use their specialised modelling approaches which are very different from each other. We believe that the tools used for the purpose of designing software agents have great potential to be useful for designing agent-based models to study Economics problems. In this paper, we provide some general guidance for building Agent-based Computational Economics (ACE) models using Software Engineering (SE) modelling methods. The applicability of these methods is demonstrated by studying a real-life multi-player dilemma using an ACE and SE approach.

I. INTRODUCTION

The research on autonomous "distributed computational units" has helped Multi Agent Systems (MAS) to become a well-established and applied branch of Artificial Intelligence. Recently, other disciplines such as social science and economics have applied the concept of agents in fine-grained models with the attention to dynamic, called Agent-based Modelling and Simulation (ABMS) (Tesfatsion, 2006). Economists, recognizing that the model of classical economics does not always match with human behaviour in real life situations, have started to use agent-based models for economic systems. With the decentralized processes provided by agent-based models, economists aim to gain a better understanding of economic systems by developing models that better represent the systems in real life. This new field, which uses agent-based models in economics, is generally referred to as Agent-based Computational Economics (ACE).

One research approach in ACE is that the economists and computer scientists collaborate. The economists design and develop the conceptual model of an economic system and its dynamics via the interaction among the economic agents. Then the conceptual model is sent to programmers to develop the computer software for this system simulation. However, most economists do not have formal training in software engineering methodology and design their model using the knowledge of classical economic modelling. If the design is improved, the collaboration can be more effective and the final agent-based model will reflect the real world better.

We believe that the economists can benefit from software engineering methodology used for the purpose of designing software agents in computer science. Computer scientists have improved the software engineering methodology for many years, and have extensive experience in agent design and implementation. One of the most important software engineering methodologies is Object-Oriented Analysis and Design (OOAD). In this paper, we briefly examine the differences between agents of economists and computer scientists, and then suggest how OOAD can be used for modelling economic agents.

II. BACKGROUND

A. From MAS to ACE via ABMS

Historically, research of systems of multiple agents was first investigated in the field of Distributed Artificial Intelligence. Jennings (1998) stated that the term "multi agent systems" refers to systems composed of many autonomous agents (or, more generally, components). Software agents are often viewed as complex objects with attitude. An agent can observe the environment, communicate with other agents, and make a decision without direct external intervention. Research in MAS involves multiple autonomous agents aiming to solve a given problem that is beyond the capability of any individual. MAS provides robustness and efficiency, allows inter-operation of existing systems, and solves problems in which data, expertise or control is distributed. One example of early application is Cammarata's (1983) study of cooperation strategy for resolving conflicts among plans of a group of agents in air-traffic control. MAS also attracted early attention with robotics, specifically the "robotic soccer" domain (Kitano et al., 1997). The interaction between MAS and Software Engineering has created a new paradigm: Agent-Oriented Software Engineering (AOSE) (Wooldridge, 2001). AOSE, in which agent-based computing is applied to software engineering, will not be discussed in detail because we focus on explaining the object-oriented thinking for agent-based social simulation.

With the growing capabilities of computers, the agents have been developed and many tools were created to support building a multi-agent system. Social scientists have seen the potential of agents and started to use the agent concept in social simulation. Social simulation is an imitation of a social system, which involves designing the model and performing experiments to gain understanding of the system and the humans in it (Gilbert, 1995). These fine-grained models with attention to dynamics have created a research field, Agent Based Modelling and Simulation (ABMS). It is an individual-centric, decentralized approach. In ABMS, a system is modelled as a collection of agents, which are autonomous decision-making units with diverse characteristics. The interaction of the individual behaviours of the agents results in the global behaviour of the system. Recently, ABMS has become a powerful technique with many applications in Sociology, Biology, Economics, etc.

Economics is one of the first disciplines using ABMS as a new research approach. In economics, economies are complex dynamic systems, which are composed of many interacting units and exhibit emergent properties. Economists have worked on modelling economic systems for many years. Recently, economists have a new agent-based methodology for modelling economic systems from the bottom up, considering the global behaviours rooted in the local interactions (Tesfatsion, 2006), called ACE. In ACE, agents can be any economic entities such as individuals (consumers, workers), social groupings (families, firms), institutions (markets), etc.

Gotts (2002) and Conte (1998) mentioned some differences between MAS (by computer scientists) and Agent-Based Social Simulation, which can be related to ACE (by economists). One of the differences is the focus. Computer scientists design the agent for complex systems or solving a particular engineering problem such as robotic control or prediction. Economists focus on modelling an economic system in order to have better understanding of the system and agents' behaviours. Given the different foci in research and different backgrounds, both communities use their specialized modelling approaches which are very different from each other. Although there are many differences, economists can still benefit from computer scientists, such as applying AI techniques (like genetic algorithms) in ACE models. We believe that OOAD, which is used for the purpose of designing software agents, has great potential to be useful for designing agent-based models to study economics problems.

C. Object-Oriented Analysis and Design

Object-Oriented Analysis and Design (OOAD) is applying the object-oriented paradigm and the visual modelling to the software development life cycle. The fundamental idea of OOAD is to break a complex system into its various objects (Booch, 2006). An object has attributes which define the state of the object and methods which describe its behaviours. Objects can communicate by sending messages to each other, and cooperate to solve a task. Objects which have the same characteristics are organized into a class. One of the most important principles in software engineering is separation of concerns, which is the idea of separating computer software into sections such that each section addresses a separate concern and overlaps in functionality are reduced to a minimum.

The Unified Modelling Language (UML) is the most universally recognized graphic representation scheme for object-oriented systems. It is a standard language for visualizing, specifying, constructing, and documenting the artifacts of software systems as well as business modelling and other non-software systems. UML is flexible, extensible, and independent of any particular design process. With UML as one standard set of graphical notations, modellers, and developers can express their designs and communicate. There are 3 types of UML diagrams (out of total of 26) that will be used in this paper: use case diagrams, class diagrams, and state diagrams.

1. UML Use Case Diagram

Use Case Diagrams are a type of UML diagram that can be used to describe the interaction of actors with the system. This diagram is used in analysis of the system. A use case diagram consists of four elements: the *system*; *actors* which can be classed into people, organizations, devices, softwares, etc.; *use cases* which represent actions that are performed by actors; *relationships* between and among the actors and the use cases. Figure 1 is an example of a use case diagram of an automated teller machine (ATM). The system is a Bank ATM. There are three actors (customer, technician, bank) and four use cases (check balance, deposit, withdraw, maintenance). A relationship between use cases and actors is represented by a straight line.

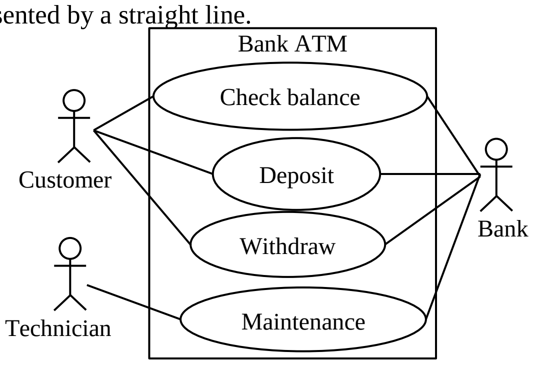

Figure 1: Use Case Diagram of a Bank ATM

2. UML Class Diagram

Class diagrams specify an object's internal data and representation, and defines the operations the object can perform. This diagram is used for structional design of the system. In class diagrams, a *class* is represented with a box, which consists of three parts. The first part contains the *name* of the class; the second part the *attributes* of the class; and the third part the *methods* or *operations* that the class can perform. Figure 2 is an example of Customer class of the Bank ATM example. The Customer class has three attributes (id, name, balance) and can perform three operations (checkBalance, deposit, withdraw).

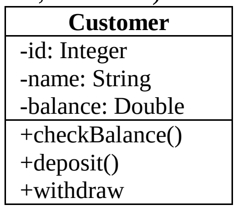

Figure 2: Class Diagram of Customer class

3. UML State Diagram

Statechart diagrams describe different states of a system and the transitions between them, and can be used to visualize and model the reaction of objects by internal/external factors. This diagram is used for behavioural design of the system. Figure 3 shows the state machine of an example Customer class, which has two main states: “Idle”, and “Using ATM”. In the “Using ATM” state, there are four sub-states (“Authentication”, “Checking balance”, “Deposit”, “Withdraw”). At the top is the statechart entry, which initializes the initial state of this statechart to be “Idle”. When in “Using ATM”, the initial sub-state is “Authentication”. Then, depending on the action the customer is going to take, the state can be either “Checking balance”, “Deposit”, “Withdraw”, before going to final sub-state in “Using ATM”.

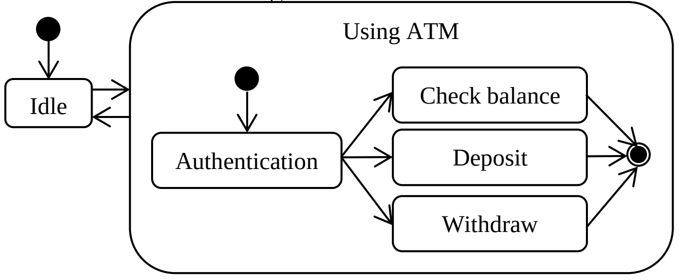

Figure 3: State machine diagram of Customer class

III. OBJECT-ORIENTED ANALYSIS AND DESIGN FOR ECONOMIC AGENTS

When models are getting more complex, it might be helpful to use the object-oriented approach. However, the difficulty for experienced economists is the transition from a functional or data centred problem-solving strategy to an object centred problem-solving strategy. In order to demonstrate the object-centred thinking, the next sections will describe how to use OOAD to model economic agents in a case study of a shared house.

A. A case study of a shared house

We will model a real-life multi-player dilemma: washing dishes in a shared house. An individual can choose to wash the dishes or not. One can contribute time to wash the dishes. If one chooses not to wash the dishes, one gains by saving his time. However, if everyone does so, there will be no clean dishes for anyone. With the limited amount of dishes, this situation is called Common Pool Resources (CPR) game in economics. A CPR is a good that individuals cannot be excluded from (non-excludable) and use by an individual reduces its availability to others (rivalrous) (Ostrom, 1990).

The next section will describe the modelling process of this case study. All models are implemented in AnyLogic 6 (XJ Technologies, 2014), a multi-method simulation modelling tool developed by XJ Technologies.

B. Model 1: Typical Economic Model (without OOAD)

We choose a generic CPR simulation (Pahl-Wostl, 2004), which was later utilized as a basis for developing the econnomics simulation of our shared house case study. The structure of our simulation is kept as similar to Pahl-Wostl’s original simulation as possible.

This model use *synchronous* scheduling, in which agents will update their characteristic at the end of the time step. At every round, all agents have 10 minutes of free time and make a decision how much time will be spent on washing dishes. To define profit, it can be said that the more invested time doing the cleaning resulted in more free time (as profit) for the agents. So the invested minutes of all agents will be increased by 60% and divided evenly among them as profit. The outcome of every agent after each round can be calculated using Equation (1).

$$\begin{aligned} outcome = freeTime - tempDecision \\ \times maxWashingTime \\ + (tempDecision \\ + expectedCooperativeness \times 3) \\ \times maxWashingTime \times 0.6 \div 4 \end{aligned} \tag{1}$$

Each agent also has a variable expectedCooperativeness (from 0 to 1) which is the level of cooperativeness it expects the other three agents to adopt (default value is 0.5). Then after every round, the expected cooperativeness is re-calculated using Equation (2) to improve the agent’s general belief about others (with default learningRate = 0.5).

$$\begin{aligned} expectedCooperativeness \\ = (1 - learningRate) \\ \times expectedCooperativeness \\ + leaningRate \\ \times meanDecisionOthers \end{aligned} \tag{2}$$

The *decision* is modelled as a variable ranged from 0 to 1, representing the cooperativeness of the agent. When the *decision* is 1, the agent spends all 10 minutes washing dishes. There are three types of agents with three different strategies:

- *Cooperative strategy*: the agent always cooperates. In this model, the cooperative agent will cooperate by spending most time washing dishes but it is also based on expected cooperativeness. If the expected cooperativeness is greater than 0.4, *decision* = 1. Otherwise, *decision* = 0.75.
- *Reciprocal strategy*: the agent changes their contribution according to recently experiences. In the case of reciprocal strategy, the agent will adjust the contribution to be the same as the mean expected contribution of the previous round (*decision* = expectedCooperativeness).
- *Maximizing strategy*: the agent always tries to deflect by maximizing its own profit. The *decision* is tried out from 0 to 1 with a predefined step of 0.1 (*temporaryDecision* = 0, 0.1, 0.2, …, 1). With every temporary decision, the outcome is calculated using Equation (1). Then the decision with highest expected outcome is chosen.

An agent of the Model 1 is implemented in AnyLogic as an active object with five variables.

- *strategy* has values of 1, 2, and 3 corresponding to three strategies of the agent.
- *expectedCooperation* (ranged from 0.0 to 1.0) represents the mean expected contribution by other agents.

- *learningRate* is used to represent rate of learning in the Equation (2) and has the default value of 0.5.
- *decision* (ranged from 0.0 to 1.0) represents the cooperativiness of the agent.
- *freeTime* (ranged from 0.0 to 10.0) represents the free time that agent has.

C. Model 2: A Model built using OOAD

In this version, OOAD is used during the analysis and design process. Similar to Model 1, the agents in the shared house can use one of three strategies: cooperative strategy, maximizing strategy, and reciprocal strategy. The cooperativeness and decision making of agents are modelled in the same as way as in the previous model. But in this version, the model uses *asynchronous* scheduling, in which agents' charateristics are updated and made available throughout the interations within a time step. Using statechart, agent only act when an event is triggered (a condition is met or a message is received). This makes asynchronous scheduling more computationally efficient and reflects the real world more accurately. The modelling process will utilize three types of UML diagrams with object-oriented thinking in mind. To demonstrate the use of OOAD, the following workflow is used:

Phase	
Analysis	1. Identify interaction of agents with the system using Use Case Diagram
Design	2. Modelling structure of agents using Class Diagram
	3. Modelling behaviour of agents using Statechart

Next sections describe each step in the workflow of how the agents are designed and implemented.

1. Identify agent interaction with Use Case Diagram

When using Use Case Diagram in our case study, the system is the simulated environment, and an actor is an agent interacting with the system. In this case, Person is an actor of the shared house. A person can perform two actions "use the dishes" or "wash them", which are two use cases. Figure 4 shows the use case diagram of this system.

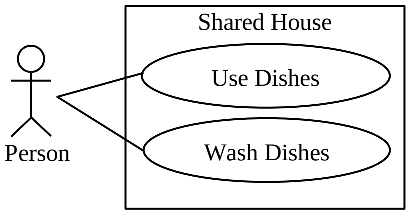

Figure 4: Use Case Diagram of the case study

2. Modelling structure of agents using Class Diagram

Based on the model description, the Person class (Figure 5) has five variables and two functions. The structure of the agent is similar to Model 1 but with two extra functions for two actions of the agents.

- makeDecision function makes a decision on the cooperativeness based on strategy of the agent.
- updateBelief function calculates the expected coopertiveness using Equation (1).

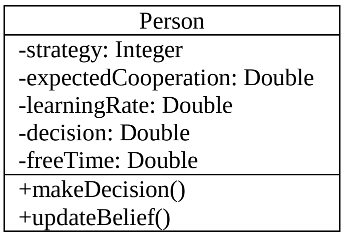

Figure 5: Class Diagram of Person agent

3. Modelling behaviour of agents using Statechart

Agent behaviour can be defined through its internal states and transitions; an agent alters its behaviour when its internal state changes. Statecharts can be implemented by adopting the state design pattern. Fortunately, modellers of all levels of expertise can now take advantage of recent visual support on statechart in simulation software such as AnyLogic (Wilensky, 2014), or Repast Simphony (Argonne National Laboratory, 2014).

In this scenario, the Person agent has three states: two states to represent two use cases (Wash and Use) and one state for when the agent is doing nothing (Idle). The statechart is shown in the Figure 6. In the beginning, the agent is in state Idle. When it is the meal time, the transition is triggered from Idle to Use. Each agent has three meals a day which are randomized in three different periods at the beginning of each day. After the agent has been in the Use state for a period, the outgoing transition will be triggered. When the agent leaves the Use state, the makeDecision function will be called to update the decision variable. The next state will be decided based on the decision variable. If the variable is 0, the agent will be in Idle state. Otherwise, it will go to Wash state. The greater the decision value is, the more time the agent stays in Wash state. After making the decision to use or wash dishes, the agent updates its belief by recalculating the expect cooperativeness using Equation (1).

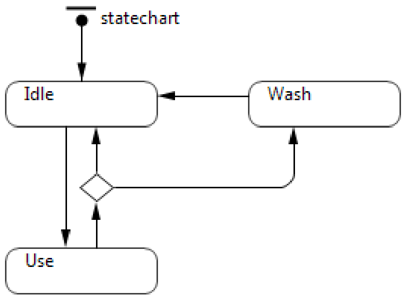

Figure 6: Statechart of Person agent

D. Add punishment options

In order to test the extensibility of two models, Model 1 and 2 will be extended by adding punishment options for agents. When the punishment option is available, economists can study how cooperation will be affected by punishment. In this case, Fehr (2002) stated that cooperation flourishes if altruistic punishment is possible. Altruistic punishment means that individuals punish although the punishment is costly for them and yields no material gain. An agent can decide to spend maximum 5 minutes for punishment. Every minute invested to punish, the punished player has to lose 3 minutes of free time.

To extend Model 1, we add a code section for punishment after the decision making and belief update. Each agent will examine the decision of each of other three agents and decide the punishment. The punishment heuristic is given below in pseudo code (p is the punisher; o is the agent in question for punishment).

```
   deflection=othersMeanDecision-o.decision;
   if (deflection < 0)
       deflection = 0;
   if (deflection > 0.1)
       angerLevel = p.inclinationBeAnnoyed
+deflection;
   else
       angerLevel = p.inclinationBeAnnoyed
+deflection - 0.8;
   if (uniform() < (2*angerLevel))
       p.punishDecision = (p. willingToPunish
+deflection) / 2;
   else
       p.punishDecision = 0;
```

Since punishment is allowed, the player has to take punishment into account when making decision. The outcome equation needs to be updated: after calculated with Equation (1), the outcome value will deduce the expected punishment from other agents:

```
   if (expectedCooperativeness>tempDecision)
        outcome = outcome - expectBePunished * (3
people) * 3 * maxPunishTime;
```

The implemenatation of agent in Model 1 is extended by adding 4 variables: inclinationBeAnnoyed, willingToPunish, punishDecision, and expectBePunished.

For Model 2, we follow the workflow. First another use case “Punish” is added to Use Case Diagram. Then the 4 punishment-related variables (similar to Model 1) and a punish method is added in Class Diagram. Lastly, the punishment behaviour is added into the statechart with two states: Punish and BePunished (Figure 7). In the Punish state, the agent will consider each of other three agents for punishment. The punishment heuristic is the same. When a punishment is decided, the punisher sends a message to the punished agent and this agent’s state changes to BePunished for the period of punished time. After all three agents are considered, the punishing agent is back to Idle state.

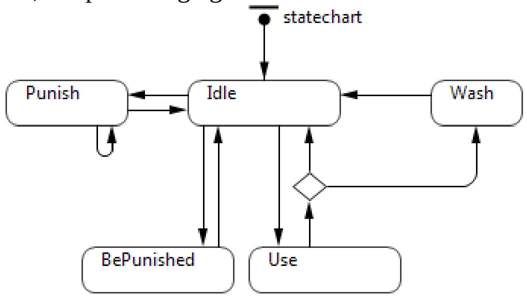

Figure 7: Statechart of Person agent (with punishment)

IV. EXPERIMENTATION

Due to the space restrictions, we only present one experiment. In this experiement, the simulation runs with four agents: one cooperator, two maximizers, and one reciprocator. Model 1 runs for 15 rounds; while Model 2 runs for 15 meals (5 days with 3 meals a day). The mean cooperativeness of all agents over time is collected in four different scenarios: Model 1 and 2 with or without punishment. In each scenario, the models run 100 times and the average cooperativeness of each round is calculated.

Figure 8 illustrates the mean cooperativeness of four agents in the house over time. In the senarios where punishment is allowed, the mean cooperativeness is escalating over time and becomes stable at 0.8. In the scenarios without punishment, the cooperativeness in the shared house decreases over time then remains stable at 0.33. The reason for the decrease is the two maximizers reduce the mean cooperativeness of the house lower than the expectation of the reciprocator. Therefore, the reciprocator keeps lowering the decision variable until its expectation is equal to the mean cooperativeness of the house. The trend of cooperativeness is expected and similar to Pahl-Wostl’s simulation.

Without punishment, the mean cooperativeness of Model 1 is similar to Model 2. But when punishment is allowed, Model 2 has a sudden drop in the second round before raises back to 0.8. The difference between Model 1 and 2 occurs because of the dissimilarity in order within decision making and the belief update between two models. In Model 1, in every round, all agents make the decision; and when every agent is finished with that, they will update the expected cooperativeness. In Model 2, all agents make a decision during a period of a meal time, and update the expected cooperativeness before going back to Idle state.

In Model 2, when the reciprocator makes a decision *after* the contributor, the reciprocator knows there are already contributions and raises his expected cooperativeness. But when the reciprocator makes a decision *before* the contributor, the reciprocator does not see any contribution. So his expected cooperativeness of other agents is lower than the previous case. This causes a significant drop in the reciprocator’s cooperativeness in the next meal which leads to the sudden drop in Model 2.

Model 2 has discovered some dynamics of how the order in agent making decision affects the cooperativeness, which is not presented in Model 1. This is a result of making the agent behaviour more realistic with asynchonous updates in Model 2.

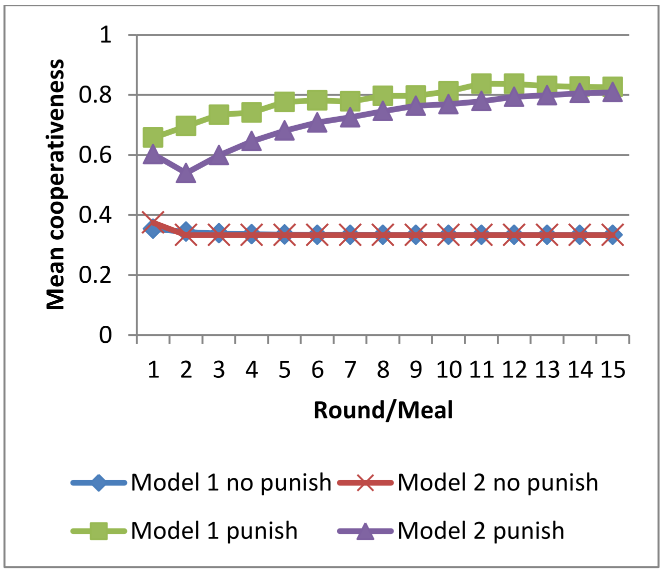

Figure 8: Mean cooperativeness over time

Assuming similar findings are possible with complex agents (in terms of cognitive capability and behaviours), it still can be difficult for ACE to design and build such an agent. Therefore, OOAD along with UML diagrams are recommended to use as the formal specification for ACE to ease the modelling process at all stages, from analysis to design and implementation.

V. DISCUSSION

A. UML: a formal visual language for agent specification

Even though only three out of 26 types of UML digrams are used in this workflow, when systems are getting more complex, modellers could find it useful to utilize other UML diagrams, as for example the Sequence Diagram, which describes how agents, objects, and the system interact and exchange messages over time.

Although ABMS research has increasingly agreed to use adopt OOAD to design and implement their models, UML is not mentioned much in their publications (Bersini, 2012). Bersini (2012) has made an effort to introduce some UML diagrams (class, sequence, state and activity diagrams) to the ABMS community. This paper's goal is not to compete but to extend this effort by demonstrating the application of OOAD thinking and UML. Not only does this help the communication between a wide range research disciplines, it can also facilitate a good development methodology for better agent-related researches.

B. OOAD as a complementary approach

In the above case study, the Person agent is not too complicated; and OOAD can still be applied. The more complex the system is, the more useful OOAD will be. In fact, OOAD is a good tool to use in KIDS (Keep It Descriptive Stupid) approach proposed by Edmonds (2005). KIDS approach suggests to start with a descriptive model as evidence and resources allow (which can be a complex model) and then only simplifies it when this turns out to be justified (as evidence and understanding of the model support this). In contrast, the KISS approach, termed by Axelrod (1997), starts with the simplest model and allows for more complex models if the simpler ones turn out to be adequate.

Neither KISS nor KIDS is the best one in every situation. But in the areas dominated by complex phenomena (typically in social science), the balance is shifted towards KIDS. Similarly, it is not always the best to use OOAD. When modellers want to build a descriptive model with complex agents, OOAD is a powerful tool to use. Otherwise, applying OOAD would be only optional.

V. CONCLUSION

This paper presents how OOAD can be used to design economic agents. In the suggestion, use case diagrams are used to identify the interation of agents with the system; class diagrams to describe the structure of agents; and statecharts to model agents' behaviour. This work is a general guidance that is sufficient for applying OOAD to modelling economic agents with complex decision-making process. It helps to increase the reusability and extensibility of agent-based models in the sense that existing models can be quickly adjusted by others researchers to perform different experiments. It is also easier to assemble different parts from multiple models to develop a new one.

Using OOAD as a complementary approach not only enhances the design of agents but also improves the communication between economists and computer scientists. This work is meant to be the first step to overcome the communication barrier between economists and computer scientists to achieve a standard agent technology with integrated interdisciplinary foundation.

For the future works, all features of Pahl-Wostl's simulation need to be implemented so that we can compare between two models for better validation. Furthermore, we can implement more social games and experiment to improve the approach and create a framework, in which OOAD is used for designing agents in social simulation.

REFERENCES

Axelrod, R. M. (1997). *The complexity of cooperation: Agent-based models of competition and collaboration*. Princeton University Press.

Bersini, H. (2012). Uml for abm. *Journal of Artificial Societies and Social Simulation*, *15*(1), 9.

Booch, G. (2006). *Object Oriented Analysis & Design with Application*. Pearson Education India.

Cammarata, S., McArthur, D., & Steeb, R. (1983). *Strategies of cooperation in distributed problem solving* (No. RAND/N-2031-ARPA). RAND CORP SANTA MONICA CA.

Conte, R., Gilbert, N., & Sichman, J. S. (1998, January). MAS and social simulation: A suitable commitment. In *Multi-agent systems and agent-based simulation* (pp. 1-9). Springer Berlin Heidelberg.

Edmonds, B., & Moss, S. (2005). *From KISS to KIDS–an 'anti-simplistic'modelling approach* (pp. 130-144). Springer Berlin Heidelberg.

Fehr, E., & Gächter, S. (2002). Altruistic punishment in humans. *Nature*,*415*(6868), 137-140.

Gilbert, N., & Conte, R. (1995). *Artificial Societies: the computer simulation of social life*. Taylor & Francis, Inc..

Gotts, N. M., Polhill, J. G., & Law, A. N. R. (2003). Agent-based simulation in the study of social dilemmas. *Artificial Intelligence Review*, *19*(1), 3-92.

Jennings, N. R., Sycara, K., & Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, *1*(1), 7-38.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., & Osawa, E. (1997, February). Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents* (pp. 340-347). ACM.

Ostrom, E. (1990). *Governing the commons: The evolution of institutions for collective action*. Cambridge university press.

Pahl-Wostl, C., & Ebenhöh, E. (2004). An adaptive toolbox model: a pluralistic modelling approach for human behaviour based on observation. *Journal of Artificial Societies and Social Simulation*, *7*(1).

Tesfatsion, L. (2006). Agent-based computational economics: A constructive approach to economic theory. *Handbook of computational economics*, *2*, 831-880.

Wooldridge, M., & Ciancarini, P. (2001, January). Agent-oriented software engineering: The state of the art. In *Agent-oriented software engineering* (pp. 1-28). Springer Berlin Heidelberg.

WEB REFERENCES

Wilensky, U. (2014). NetLogo. Available from http://ccl.northwestern.edu/netlogo. Center for Connected Learning and Computer-Based Modeling. Northwestern University. Evanston, IL.

Argonne National Laboratory (2014). The Repast Suite. Available from http://repast.sourceforge.net/

XJ Technologies (2014). AnyLogic 6. Available from http://www.anylogic.com/.