OBJECT-PROCESS METHODOLOGY APPLIED TO AGENT DESIGN

Zoheir Ezziane
College of Information Technology, Dubai University College, P.O. Box 14143, Dubai, United Arab Emirates

Keywords: Object-process methodology, Agent design.

Abstract: As computer systems become ever more complex, we need more powerful abstractions and metaphors to explain their operations. System development shows that designing and building agent systems is a difficult task, which is associated with building traditional distributed, concurrent systems. Understanding natural, artificial, and social systems requires a well-founded, yet intuitive methodology that is capable of modeling these complexities in a coherent, straightforward manner. Object-Process Methodology (OPM) is a system development and specification approach that combines the major system aspects (function, structure, and behavior), into an integrated single model. This paper will provide a paradigm for designing agent systems using the object-process methodology. It aims to identify design concepts, and to indicate how they interact with each other.

1 INTRODUCTION

Systems and products are becoming increasingly complicated. Technology has been so pervasive that even commonly used products feature high computational power, embedded within increasingly miniature, precise, and involved hardware. Systems of an infrastructure nature, such as the Internet and digital economy, are orders of magnitude more complex than products individuals normally use.

Understanding natural, artificial, and social systems requires a well-founded, yet intuitive methodology that is capable of modeling these complexities in a coherent, straightforward manner.

Artificial systems require development and support efforts throughout their entire lifecycle. Systematic specification, analysis, design, and implementation of new systems and products are becoming even more challenging and demanding, as contradicting requirements of shorter time-to-market, rising quality, and lower cost, are on the rise. These trends call for a comprehensive methodology, capable of tackling the mounting challenges that the evolution of new systems poses.

OPM is a system development and specification approach that combines the major system aspects (function, structure, and behavior), into an integrated single model. This methodology is used to design multiagent systems.

Multiagent systems (MAS) are concerned with the behavior of a collection of possibly pre-existing autonomous agents aiming at solving a given problem. A MAS can be defined as a loosely coupled network of problem solvers that work together to solve problems that are beyond the individual capabilities or knowledge of each solver (Jennings et al., 1998). These problem solvers (agents) are autonomous and may be heterogeneous in nature.

This paper will provide a paradigm for designing agent systems using the object-process methodology. It aims to identify design concepts, and to indicate how they interact with each other.

2 BACKGROUND

2.1 Object-Process Methodology

OPM takes a fresh look at modeling complex systems that comprise humans, physical objects, and information (Dori, 2002). OPM is a formal paradigm to systems development, lifecycle support, and evolution. OPM was applied in such diverse areas as computer integrated manufacturing (Dori, 1996), image understanding (Dori, 1996), modeling research and development environments (Meyersdorf et al., 1997), document analysis and

Ezziane Z. (2004).
OBJECT-PROCESS METHODOLOGY APPLIED TO AGENT DESIGN.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 455-462
DOI: 10.5220/0002657604550462
Copyright © SciTePress

recognition (Wenyin, 1998), algorithm specification (Wenyin et al., 1999), software engineering (Tomlin et al., 1998), modeling electronic commerce transactions (Dori, 2001), and web applications (Reinhartz-Berger et al., 2002).

Natural and artificial systems alike exhibit three major aspects: function, structure, and behavior. In the case of artificial systems, OPM provides a framework for the entire system's lifecycle, from the early stages of requirement elicitation and analysis, through further development and deployment, all the way to termination and initiation of a new generation (Dori, 2002).

OPM combines formal yet simple graphics with natural language sentences to express the function, structure, and behavior of systems in an integrated, single model. Objects and processes are the two main building blocks that OPM requires to construct models. A third OPM entity is state, which is a situation at which an object can be and therefore a notch below object. Objects, processes, and states are the only bricks involved in building systems. The links connecting these three entities act as the mortar that holds them together (Dori, 2002).

OPM is an integrated approach to the study and the development of systems in general and information systems in particular. The basic premise of OPM is that objects and processes are two types of equally important classes of things, which together describe the structure, function, and behavior of systems in a single framework. OPM unifies the system specification, design, and implementation within one frame of reference, using a diagramming tool (Object-Process Diagram), and a corresponding, English-like language (Object-Process Language) (Reinhartz-Berger et al., 2002).

OPM has two scaling mechanisms: unfolding/folding and zooming-in/zooming-out. The unfolding/folding mechanism uses structural relations for detailing/abstracting the structural parts of a thing. For example, in figure 1, the process Traveling is unfolded to expose its parts, processes Destination Selecting and Budgeting Planning.

The zooming mechanism exposes/hides the inner details of a thing within its frame. In figure 2, the process Traveling is zoomed-in. This way OPM facilitates focusing on a particular subset of things (object and/or processes), elaborating on their details by drilling into them to any desired level of detail.

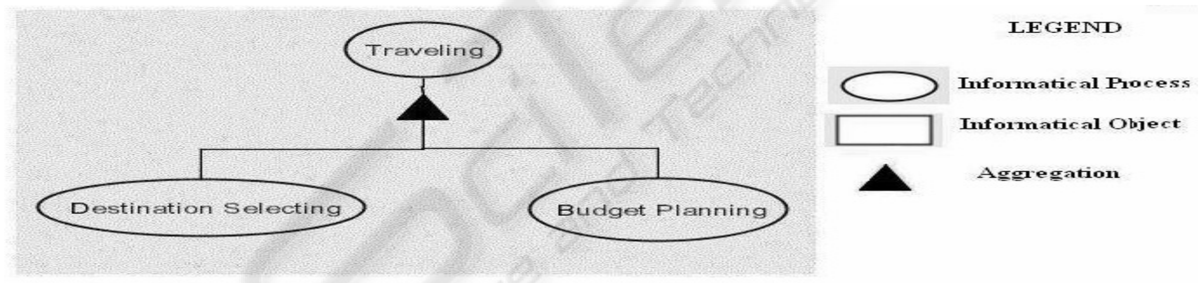

Figure 1 : Process traveling is unfolded

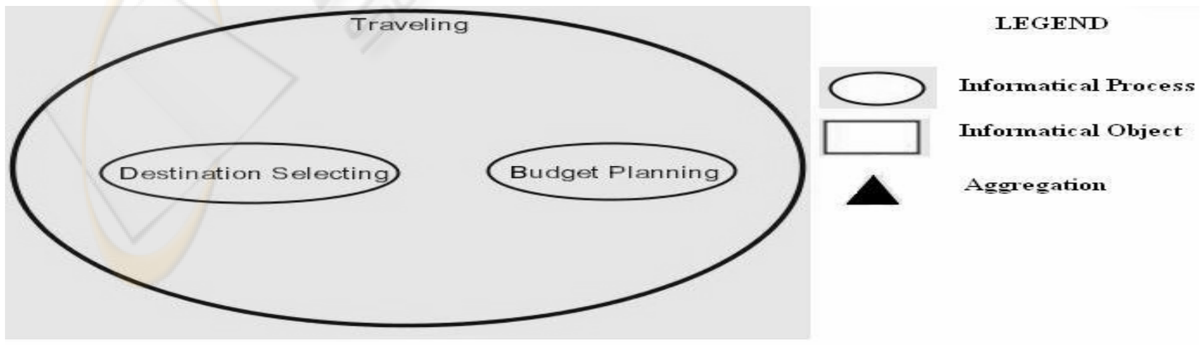

Figure 2 : Process traveling is zoomed-in

2.2 Multiagent Systems

MAS provide a paradigm for analyzing, designing, and implementing software systems. The agent-based view offers a powerful repertoire of tools that have the potential to considerably improve the way in which people design and implement many types of applications. Agents are being used in an increasingly wide variety of applications, ranging from small applications such as personalized email filters (Abushar et al., 2003) to large, complex mission critical systems such as Internet trading (Chavez et al., 1996), (Stone et al., 2001), (Chan, 2001), (Das, 2003) and air-traffic control (Cammarata, 1983), and (Tomlin, 1998). Kasbah is an example of Internet trading, in which it establishes a virtual marketplace on the Web where users create autonomous agents to buy and sell goods on their behalf (Chavez et al., 1996). In addition to providing solutions to meet real-world needs, they demonstrate to be a useful technology.

An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives (see figure 3). An agent has ability to perceive, reason, act, and communicate. Furthermore, an agent has explicitly knowledge and methods for operating on or drawing inferences from its knowledge. The key problem facing an agent is that of deciding which action it should perform in order to best satisfy its design objectives (Wooldridge, 2002)

Agents operate and exist in some environment, which is both computational and physical. The environment might be open or closed, and it might or might not contain other agents. Although there are situations where an agent can operate usefully by it self, the increasing interconnection and networking of computers is making such situations rare, and therefore interaction among agents is the most common situation (Weiss, 2000).

Much of the traditional AI has been concerned with how an agent can be constructed to function intelligently, with an internal reasoning and control. However, intelligent systems do not function in isolation. They are part of the environment in which they operate, and the environment contains other intelligent systems. Consequently, those systems form a society of agents.

Information environments are too large, complex, and open to be managed centrally. Computational intelligence must be embedded in such environments to provide distributed control. A problem for a multiagent system is how it can maintain global coherence without explicit global control. Hence, agents must be adaptive (they should be able to explore and learn their environment), and social (they should interact and coordinate to achieve their own goals, and the goals of their society). One way of achieving coherence is through social commitments.

The nature of the environment is highly dependable on many properties. The environment properties are classified as follows: Accessible versus inaccessible; Deterministic versus non-deterministic; Static versus dynamic; Discrete versus continuous (Russel, 2003). The most complex class of environments (i.e., open) is those that are inaccessible, non-deterministic, dynamic, and continuous. Environmental properties have a role in determining the complexity of the agent design process, but a second property such as the nature of interaction between agent and environment is also important to consider.

Details about how agent communicate, cooperate, and negotiate are available in the literature (Wenyin et al., 1998), (Wenyin et al., 1999), (Weiss, 2000), (Russel, 2003), and (Jennings et al., 1998).

2.3 Current Design Methods

Object-oriented (OO) design is a general paradigm for developing systems that focuses on the objects that build the system. The strength of OO techniques is in modeling the structural aspects of a system.

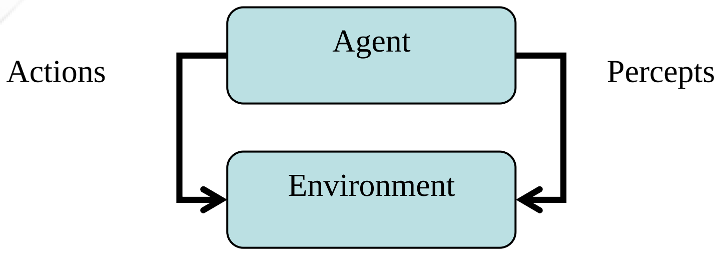

Figure 3: Agent interacts with its environment

However, they are far less suitable for representing the dynamic and functional aspects of a system.

Current OO techniques suffer from three major inter-related problems: the encapsulation, the complexity management, and the model multiplicity problem. They do not have a mechanism for specifying stand-alone processes, which are not owned by a certain object and counter the encapsulation principle. Moreover, the encapsulation principle eliminates the dynamic aspect of the system. Thus, while being a useful programming convention, this unnecessary encapsulation constraint has been a source of endless confusion.

The complexity management problem is related to the way OO methods deal with the complexity that emerged by splitting systems into various models such as structure (the object/class model) and dynamics (Statecharts). Therefore, when the complexity of the system increases, no tools will be available to describe the entire system.

The inadequacy of accommodating the functional and dynamics system aspects, which OO methods suffer from, has contributed to the model multiplicity problem (Peleg, 2000) and (Lovitz, 1998). The most common object-oriented modeling language is UML (OMG, 2003), (Agarwal et al., 2003), and (Medvidovic et al., 2002). The UML standard requires nine different models, including class diagrams, use case diagram, object message diagram, state diagram, module diagram, and platform diagram. The model multiplicity problem refers to the need to comprehend a variety of models of the same system and synchronize them. Hence, the need to specify a system with just one model will be definitely better than a multi-model one (Peleg, 2000).

3 AGENT SYSTEM DESIGN USING OBJECT-PROCESS METHODOLOGY

In this paper, we present an application of OPM to modeling the basic multiagent design as a case to demonstrate OPM's semantics. However, OPM is domain independent, and has been applied to many areas. The feature of OPM is that it depends on how objects and processes are defined. This characteristic of OPM makes it suitable for developing systems in a large variety of domains.

Viewing agents from an abstract level prepares the ground for a smooth analysis. However, it does not necessarily lead to their construction. This abstraction is a useful software engineering abstraction (e.g., abstract data types). Hence, OPM is used to refine the model of agents, by breaking it down into sub-systems in exactly the way that one does in standard software engineering. As the view of agents is being refined, design options related to the agent's subsystem would become transparent. Figure 4 describes the OPM design for a multiagent system, indicating the characteristics of an agent and the way it interacts with its environment.

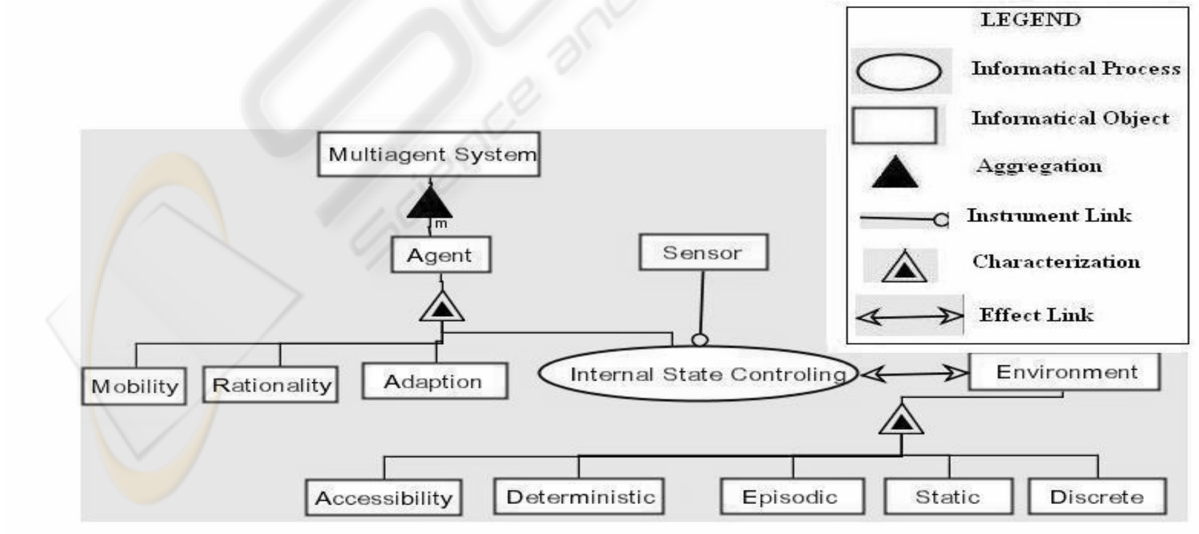

Figure 4 : OPM design for a multiagent system

Mobility indicates whether the agent has any ability to move from one site to another. One of the characteristics of social agents is that participating agents ought to have some sort of rationality, in order to build a coherent society. Adaptation means whether the agent should adapt itself to how the environment evolves

3.1 Simple Reflex Agents

One of the simplest classes of agents is the simple reflex agent, also called purely reactive agents. They simply respond directly to their environment. These agents select actions based on the current percept ignoring the rest of their percept history. They base their decision-making entirely on the present.

A lamp agent is an example of a simple reflex agent. Assume that the lamp's environment can be in one of the two states (either on, or off). Figure 5 depicts this kind of agents.

3.2 State-based Reflex Agents

The model for simple reflex agents helps designing agents with state. That is, the agent should keep track of an internal state that depends on the percept history. These agents must have some internal data structure, which is used to update the internal state information. Thus, the internal state of any agent needs to record how the world evolves independently of the agent, and how the agent's actions do affect the environment. Figure 6 depicts an OPM design for sate-based reflex agents.

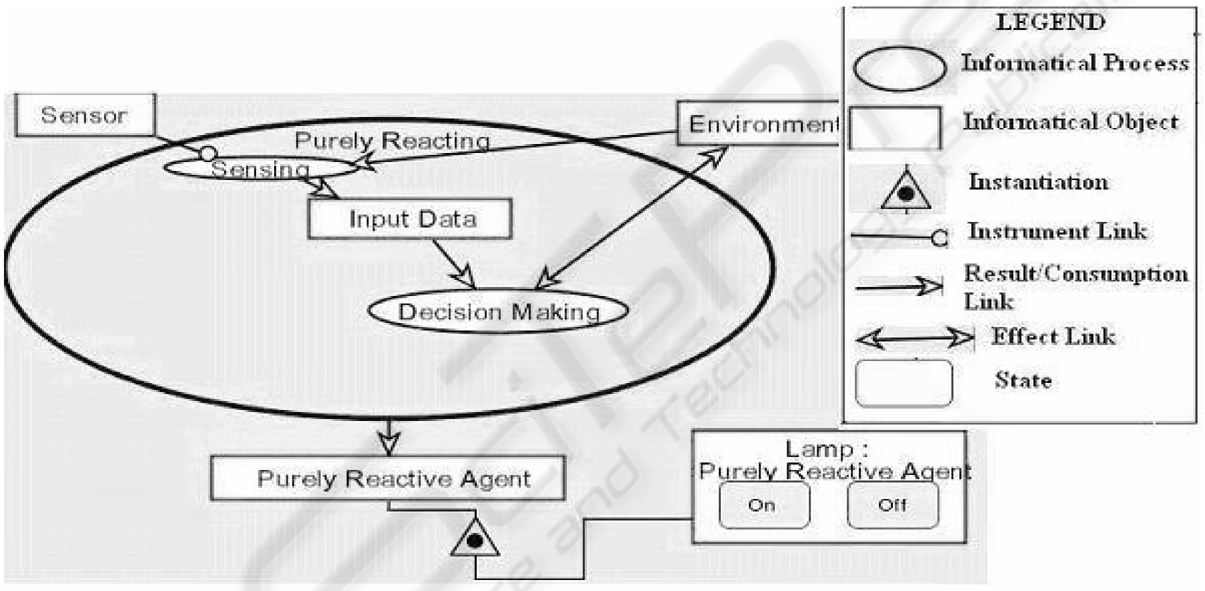

Figure 5: OPM design for a simple reflex agent

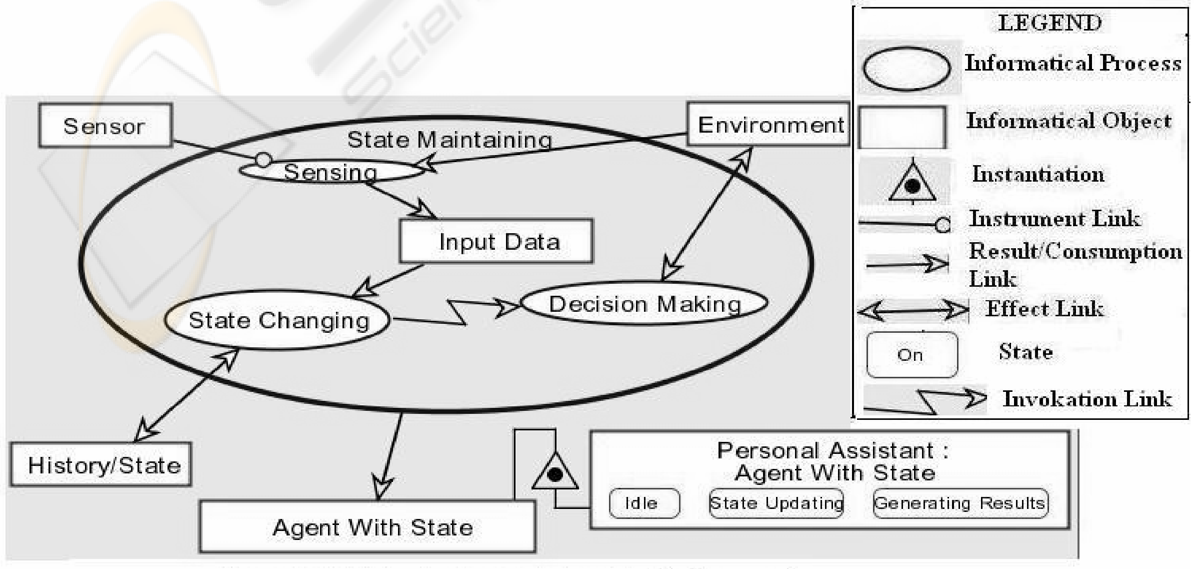

Figure 6 : OPM design for a state-based reflexive agents

A good example of purely reactive agents is an internet personal assistant, which can be used to help the user by storing the preferred links, advice the user to visit a certain site based on the user's profile, etc.

3.3 Utility-based Agents

A utility function is needed in order to map a state (or a sequence of states) onto a real number, which describes the associated degree of happiness. This utility function is used to specify the appropriate tradeoff that could result from certain action. Figure 7 depicts an OPM design for utility-based agents.

Game assistant is an example of a utility-based agent. It suggests to the user to make certain moves, using a certain strategy that either minimize penalties or increase points.

3.4 Negotiations and Reaching Agreements

Cooperation is a concept in the human world. Often when a group of people work together to solve a problem or to achieve an objective, not only they can increase productivity and efficiency, but also they can solve a problem that cannot be solved by an individual alone.

Communication can enable the agents to coordinate their actions and behavior, which results in a more coherent society. The desired property of coordination avoids resource contention, livelock, and deadlock. Typically, in order to reach a successful cooperation, each agent must maintain a kind of a model of how other agents evolve in the environment.

As electronic commerce (e-commerce) is rapidly becoming a reality, the need for negotiating methods that take into consideration the complexities of the real world environment, such as incomplete data and negotiation deadline.

A frequent form of interaction that occurs among agents with different goals is termed negotiation. A negotiation is a process by which a joint decision is reached by two or more agents, where each one is trying to reach an individual goal. In this case, the agents need to communicate their actual positions, which might conflict, and then try to make an agreement through concessions or searching for other options.

One of the most important issues in both conventional and electronic trading is for sellers and buyers to reach a consensus on pricing and other terms of transactions. The task of negotiation and reaching an agreement can often be difficult and time consuming. Thus, it is crucial to use a transparent design tool to design and engineer a society of trading agents that help users negotiate terms of business transactions.

3.4.1 Auctions

Recently, online auctions rapidly achieved enormous popularity in e-commerce. Numerous online auction houses have been already established on the web, such as Amazon, BargainFinder and eBay. Hence, the design and development of a multiagent system will provide some opportunities to improve auction performance.

Agents can be used to compare products of several producers and help to find the best bargain,

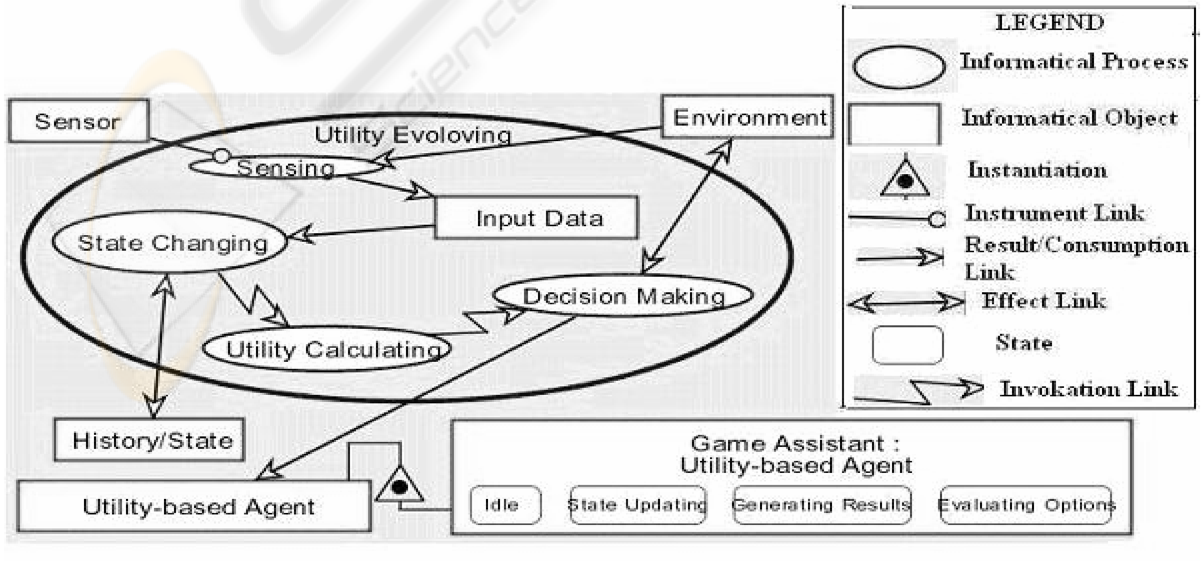

Figure 7 : OPM design for utility-based agents

but they can also be used to perform the actual transaction. The interest is focused on the latter case because it represents an actual multiagent e-commerce system. If a direct trading procedure is followed, agents of producers and consumers should negotiate the deal and inform their users. In case of a very simple negotiation, the transaction can be done without the use of an institution. For example, an agent can buy a book at Amazon.com, without requiring an institution because the transaction is simple.

However, the transaction can be performed in other cases where a certain institution is needed. The institution determines the way that the parties can conduct the transaction and provides an infrastructure to do it. A good example is Kasbah, a market place where agents can buy and sell goods. Kasbah gives a predefined structure for specifying the product and also facilitates the negotiation process. Kasbah most closely resembles an online classified ad system. When a user wants to sell something (e.g., used book) he/she registers the object for sale with the Kasbah server via a computer interface, buyers then go to Kasbah to look for something to purchase. But Kasbah is able to use software agents to negotiate a sale.

In the case of indirect trading, there exists a natural institution (e.g., auction) in the form of the broker that mediates in the transaction. The auction determines the rules of encounters, such as how products can be delivered at auction level as well as how products can be bought. Despite the auction's simplicity, they present a powerful tool that automated agents can use for allocating goods, tasks, and resources.

An auction takes place between an agent (auctioneer) and a set of agents (bidders). The objective of the auction is for the auctioneer to allocate the good to one of the bidders. Usually, the auctioneer would like to maximize the price at which the good is allocated, while the bidders want to minimize it. Achieving the auctioneer's goal is realized through some rules of encounter. Figure 8 shows an attempt to design an auction multiagent system.

Winner determination is an auction protocol. The first-price is used to allocate the good to the agent that bids the most, while the second-price is to allocate the good to the second highest bid. The other auction protocol specifies on how bids are announced. If the bids are known and made available to all agents in the system, then the auction is open-cry, otherwise it is sealed-bid.

4 CONCLUSION

This paper emphasizes on a different framework (OPM) in designing agents. OPM is able to represent all the important interaction in a system, and is widely used in various fields. This approach includes a clear and concise set of symbols that form a language enabling the expression of the system's building blocks and how they relate to each other. Various attempts in designing agent systems were approached; a comparison of OPM with UML was discussed. It was shown how OPM is more effective than the current UML. As an extension to this research, it would be interesting to investigate the compliance of OPM in designing agents with FIPA standards. Many major IT and telecommunications companies had become involved in the FIPA trend, and a set of prototypical standards had been developed. Hence a new methodology in designing agents such as the one discussed here in this paper will be more valuable when is compliant with FIPA specifications.

Future work could also emphasize on a detailed

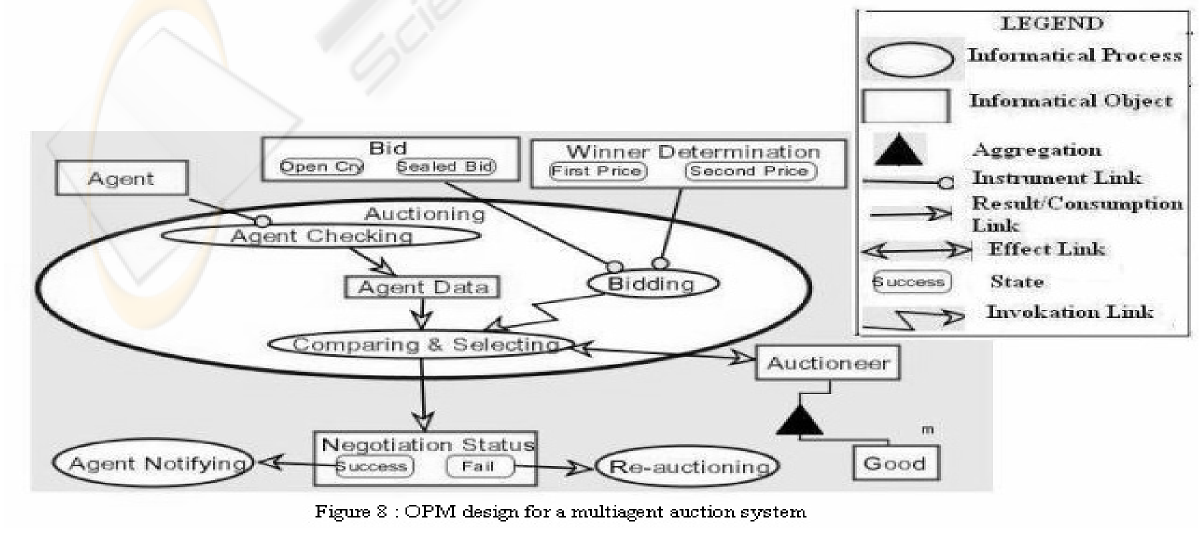

Figure 8 : OPM design for a multiagent auction system

description of the transactional process that occurs during trading. Another approach would be to investigate how bidding agents buy and sell multiple interacting goods in auctions of different types, which represent complex and rapidly advancing domains of e-commerce and e-marketing.

REFERENCES

Abushar, S. and Hirata, N. Filtering with Intelligent Software Agents. Available from: <http://www.engin.umd.umich.edu/CIS/course.des/cis479/projects/FISA.html.> [Accessed September 10, 2003]

Agarwal, R. and Sinha, A. P., 2003. Object-Oriented Modeling with UML: A Study of Developer's Perceptions. *Communication of the ACM*, 46(9), 248-256.

Cammarata, S., McArthur, D., and Steeb, R., 1983. Strategies of Cooperation in Distributed Problem Solving, *Proc. 8th Int'l Joint Conf. on AI (IJAI-83),* Elsevier, Karlsruhe, Germany, 767-770.

Chan, T., 2001. *Artificial Markets and Intelligent Agents*, Ph.D. dissertation, Dept. of Electrical Eng. and Computer Science, Massachusetts Institute of Technology, Cambridge, Mass., USA.

Chavez, A. and Maes, P., 1996. Kasbah: An agent marketplace for buying and selling goods. *Proc. 1st Int'l Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology*, Practical Application Company, London, UK, 75-90.

Das, S., 2003. *Intelligent Market-Making in Artificial Financial Markets*, MSc thesis, Dept. of Electrical Eng. and Computer Science, Massachusetts Institute of Technology, Cambridge, Mass., USA.

Dori, D., 1996. Object-Process Analysis of Computer Integrated Manufacturing Documentation and Inspection Functions. *International Journal of Computer Integrated Manufacturing*, 9(5), 339-353.

Dori, D., 1996. Analysis and Representation of the Image Understanding Environment Using the Object-Process Methodology, *Journal of Object-Oriented Programming*, 9(4), 30-38.

Dori, D., 2001. Object-Process Methodology Applied to Modeling Credit Card Transactions, *Journal of Database Management*, 12(1), 2-12.

Dori, D., 2002. *Object-Process Methodology – A Holistic Systems Paradigm*, New York: Springer.

Jennings, N.R. Sycara, K. and Wooldridge, M. (1998). A Roadmap of Agent Research and Development, *Autonomous Agents and Multi-Agent Systems*, 1, 275-306.

Kovitz, B.L., 1998. *Practical Software Requirements: A Manual of Content and Style*, Manning Publication Company.

Medvidovic, N., Rosenblum, D.S., Redmiles, D.F. and Robbins,J.E., 2002. Modeling software architectures in the Unified Modeling Language, ACM Transactions on Software Engineering and Methodology. (TOSEM), 11(1), 2-57.

Meyersdorf, D. and Dori, D., 1997. The R&D Universe and Its Feedback Cycles: An Object-Process Analysis, *R&D Management*, 27(4), 333-344.

OMG, Unified Modeling Language Specification. Available from: <http://www.omg.org/technology/documents/formal/uml.htm> [Accessed September 9, 2003]

Peleg, M. and Dori, D., 2000. The model multiplicity problem: Experimenting with real-time specification methods, *IEEE Transaction on Software Engineering*, 26(8), 742- 759

Reinhartz-Berger, I. and Dori, D., 2002. OPM/Web–Object-Process Methodology for Developing Web Applications, *Annals of Software Engineering*, 13, 141-161.

Russel, S. and Norvig, P., 2003. *Artificial Intelligence: A Modern Approach*, *2nd* ed., New Jersey: Prentice-Hall, USA.

Stone, P. and Greenwald, A., 2001. Autonomous bidding agents in the trading agent competition. *IEEE Internet Computing*, 5(2), 52-60.

Tomlin, C., Pappas, G. J. and Sastry, S., 1998. *Conflict resolution for air traffic management: A study in multi-agent hybrid systems*, IEEE Transaction on Automatic Control, 43, 509-521.

Weiss, G., 2000. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, Cambridge, Mass. USA.

Wenyin, L. and Dori, D., 1998. A Generic Integrated Line Detection Algorithm and its Object-Process Specification, *Computer Vision – Image Understanding*, 70(3), 420-437.

Wenyin, L. and Dori, D., 1999. Object-Process Diagrams as an Explicit Algorithm Specification Tool, *Journal of Object-Oriented Programming,* 12(2), 52-59.

Wooldridge, M., 2002. *An Introduction to MuliAgent Systems*, John Wiley & Sons, UK.