COMPONENT BASED DEVELOPMENT IN MULTI AGENT SYSTEMS

Koen Bertels[1] and Magnus Boman[2]

[1] Delft University of Technology - Computer Engineering Lab
P.O. Box 5031, 2600 GA Delft,The Netherlands
koen@dutepp0.et.tudelft.nl
[2] Swedish Institute of Computer Science
Electrum 230 SE-164 40 Kista, Sweden
mab@sics.se

Abstract. While tools for agent based social simulation have come a long way towards making life easier for the social scientist depending on computerized methodologies, we find that some of the requirements for efficient scientific progress have not been met. Inspired by the success of simulation environments and tools in the last decade, by the work on the alignment of models and by proven technology in business information systems, we propose a common methodology in the form of a common object model architecture for agents(COMA).

1 Introduction

The focus of this paper is on artificial agent based societies as created by both social scientists and real world application developers. Even though the final objectives may be different, the underlying architecture and technology are the same. As far as social scientists are concerned, there is considerable excitement with respect to the possibilities of agent based computer simulations of social phenomena. Access to data is crucial to the social scientist for testing hypotheses, and computer simulations bear the promise of unlimited data generation, provided by the computer model. It is certainly true that the use of Multi-Agent Systems (MAS) opens up new scientific perspectives, but we must keep in mind the fundamental limitations and pitfalls of computer based modeling. Our goal is to address these issues, to make some of the tacit assumptions in MAS explicit, and to present a proposal allowing for systematic and co-coordinated progress. Application developers are increasingly using agent based techniques, especially for web based applications. It is considered that especially for ecommerce and ebusiness this technology will play a key role as it will allow the delegation of real world decisions to agents that will act on behalf of the human owner. The paper therefore addresses issues relevant for both groups even though certain issues may at first sight seem more oriented towards social scientists, the methodological issues are common to both application domains. The most important of these issues is to arrive at an agent behaviour which is understandable by its owner or by the scientist who specified the model. This almost automatically implies

that the agent should behave in a realistic and even rational (explainable) way. Axelrod points out that MAS and more specifically agent based social simulations(ABSS) as a scientific method can be problematic [1]. He lists three goals of programming a simulation model, namely internal validity, usability, and extendibility (op.cit., p.26). These pertain respectively to correctly implementing the model, executing and documenting the program, and making the program adaptable for new uses. Axelrod not only stresses the need to exchange models and to replicate results, but also to verify that the obtained results are not unwanted side effects of a flaw in the model or even a programming error. The main argument of the present paper is that this latter objective is very difficult to realize unless some coordinated research action is undertaken.

One of the widely proclaimed advantages of ABM is its ability to exhibit so-called emergent properties: novel, coherent, global, dynamical, and ostensive properties [2]. This concept refers to the observed behavior of the model, which was not explicitly programmed and is therefore unanticipated in the sense that the theory did not predict it. Not only properties and behavior can be emergent. The process by which organizations emerge, autopoieses [3], can be described as striving towards organizations not built on stable traditional structures. The prime example is the emergence of a global friction-free marketplace on which artificial agents can act as electronic intermediaries [4]. It is important to make sure that emergent behavior, or other forms of macro level self-organization in multi-agent systems (MAS), is not a result of certain structural properties of the model, or even programming bugs. Whereas Axelrod pleads for more replication efforts, we go one step further in that we suggest a common object model architecture (COMA) that includes a methodology for studying emergence. The concept itself has achieved more attention in physics and in organization theory than in computer science, but its relevance to scientific explanations is transdisciplinary [2].

In the remainder of the paper, we discuss the reasons in favour of a middleware approach and discuss what the expected benefits are as well as what is required to introduce succesfully such an approach.

2 Towards a Middleware Approach

There are currently some programming and modeling environments around to assist the creation of MAS-models, but they do not solve the fundamental programming and parameterization problems. SDML, for instance, provides a nice temporal framework and certain visualizations of model behavior but the actual coding and programming of even the simplest agent wanting, e.g., to buy, sell, or consume still needs to be programmed from the bottom up.

Related research on agents for electronic commerce area is so far reporting more on middle-agents in the form of brokers than on middleware use as such [5]. However, there do exist a number of platforms such as Tryllian (www.tryllian.com), Jade (jade.cselt.it). Such environments provide an environment in which the agents behave and operate. They provide a wide range of

necessary mechanisms such as message based communication, reasoning capabilities etc. necessary to give the agent interesting behaviour. However, these mechanisms have less semantic value than what we are looking for. Where such systems enable the developer to easily set up a MAS, realistic functional (and therefore semantically interesting) behaviour is still lacking and remains the responsibility of the developer even though a lot of the desired behaviour may have been developed by others. The environment which comes closest to what we envisage is Zeus (193.113.209.147/projects/agents/zeus/) where the user can specify his own ontology. The argument we raise in the remainder of the paper is a coordinated approach to develop ontologies for various domains and turn each these ontologies in a technical implementation based on components. We therefore plead for an approach similar to the one adopted for business information systems. There, the current technological trend is towards component based modeling, resulting in several accepted standards: Component Object Model (COM), Common Object Request Broker Architecture (CORBA), Enterprise Java Beans and SOAP.

The main idea of component based development is to take the benefits of object oriented design and programming one step further. This can be achieved by re-using units, which perform certain functions and which can be integrated to form large scale distributed information systems. The middleware approach grew out of the need to interconnect different information systems and to be able to link to different kinds of databases. Its use is greatly stimulated by the recent changes in the technology landscape of which Internet, e-commerce, and private virtual networks are the most important examples. In [6], components are defined as "widely deployable products that are specifically designed and packaged to be assembled with other such products". These components will most likely become the new standard for modular software design and information system development will then potentially consist of assembling different components, as has happened in the area of business information systems [7]. It is important to realize not only that these components can and will be developed by different vendors who use standard interfaces, but that each of these components can be developed in whatever programming language and be running on any platform. Standards for components have been proposed by the Object Management Group resulting in CORBA and Microsoft, Sun, and IBM are promoting similar activities for their own middleware architectures.

The goal is to realize user-level assembly of plug and play applications using CORBA components, which can run on any kind of distributed computing environment. To this purpose, a scripting language is used and CORBA-services are provided for distributed OO-applications. They specify in a standardized way the services any object or component requires. Examples are a naming and trading service that will allow access to objects across the network and an event service that will notify important events or changes in object states. When CORBA is adopted, it allows multiple interchangeable implementations of the basic functionality, provided by different commercial vendors. On the vendors' side, the main benefit is that they are able to focus primarily on the application domain

realising seamless integration with other components. In the next section, we address the issue of what is required in order to realize these potential benefits.

3 Where are the Benefits ?

What lessons can be learned from this COM and CORBA story ? In multi agent systems, it is an evident necessity to be able to easily interact with other agents or with the environment. When restricting our view to simulations, components, which have been developed and thoroughly tested by other researchers, could become accessible and usable by other researchers. It will speed up the development of models, make them more robust and richer. In order to clarify these advantages, we give two examples.

3.1 Constructing a Buying Agent

Let us assume that we want to develop a Buying Agent. This agent should be able to compare different offers of a particular product, and therefore, to have some kind of reasoning capability for assessing the usefulness of different products for his owner[3]. The agent should also be able to interact with his owner in order to improve his understanding of his preferences. Without detailing the complete structure of the agent, it seems reasonable to assume that our agent needs knowledge on different sets of consumer products. He should also be able to learn from past experiences and interact with his owner. Rather than program everything from scratch, it is much more interesting to re-use modules which have been developed, tested and optimized by others. In Figure 3.1, we depict the situation in which there is somewhere a Consumer Behavior Component. We can access this component and use the methods of the different classes in this component. In that way, we can give our agent learning capabilities and experiment with both neural networks (NNLearning) and genetic algorithms (GALearning) to see which of both techniques works best for our application. We can also use the class UtilityFunction in which, again, different kinds of utility functions are available. A researcher would then be able to simply 'plug in', for instance, a different utility function. He can thus evaluate how changing a concave, strictly monotonous utility function into a convex, strictly monotonous utility function would impact the observed behavior in a model. Another element of consumption behavior could be to search for appropriate goods and to compare the relevant characteristics of found goods.

The agent should also have flexible negotiation strategies enabling him to deal with for instance fixed price markets where only a discount can be negotiated but also auction market situations where a bidding strategy iis required. The Iclass Factory is present in any component and is responsible for creating a new instance of the component whenever a request has been received.

[3] We refer to the human consumer on whose behalf the agent searches the Internet as the owner.

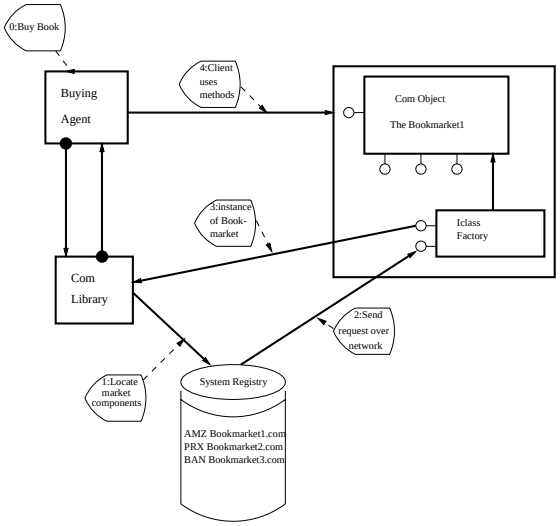


Fig. 1. A Consumer Behavior Component and some of its Methods

3.2 Using a Buying Agent

Once we have developed the agent, we can use it either in a simulation or in the 'real' world. We will assume that another research group has created an artificial (book) market using the COMA approach and that our agent goes to their market to look for a book. We can trace a normal trajectory of the agent when going out on the Internet to look for a particular bargain. [4] As shown in Figure 3.2, the Buying Agent is instructed to look for a particular book. He needs to visit a number of bookstores and access their information. The Buying Agent needs to have some kind of knowledge of how and where to find the information. He therefore requests access to the component of bookstore TheBookMarket1 implementing these methods. The COM library looks up where the components of the different bookstores are located and sends the request over the network to, for instance, TheBookMarket1. The Iclass Factory of TheBookMarket1 instantiates the component and the COM library receives a pointer to that component's interface. This interface is passed to the Buying Agent who can then use the methods supported by the TheBookMarket1 component. For each method, the parameters are given and its data type is defined.

4 Towards a Component Object Model for Agents (COMA)

It is obvious that the Agent community can only exploit the full potential when sufficient synergy and well co-coordinated reflection can be established. The

[4] We assume that containment has been chosen as inheritance mechanism that implies that whenever some functionality is needed, the required component is contacted.[8]

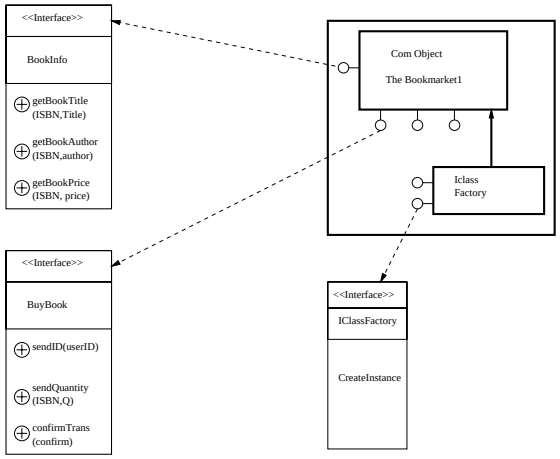


Fig. 2. A Buying Agent Accessing TheBookMarket1-logic

following appear to be essential elements of such a strategy: the creation of a commonly accepted ontology and the specification of different technical realizations. The latter implies the formulation of standardized interfaces and the use of COM and CORBA technology.

4.1 Creating an Ontology

The first requirement is to create a widely accepted ontology of the different phenomena that are mostly used in social simulation. Such an exercise is necessary as it clarifies the structure of knowledge and the vocabulary used in a given domain. In [9], the following advantages of shared ontologies are listed : a common ontology allows knowledge sharing and will increase the coherence of its conceptual structure. Others having similar needs can then use this knowledge representation and don't have to go through the phase of knowledge creation. We can distinguish between a general-purpose upper ontology in which common sense knowledge is represented and domain specific ontologies. A well-known example of the creation of such an upper level ontology is the CYC project by D. Lenat [10]. An example of a domain specific ontology is [11] where a library of logical data models and data warehouse designs is described. More specifically, defining an ontology for a particular domain, implies describing the following [9]:

- Describe the objects that exist in the world. Examples are humans, organizations, goods,...
- Describe these objects' properties or attributes. These attributes can then have values.
- Relations can exist between objects.
- Objects can have parts or components.

- The attributes and the relations between objects can change over time.
- Certain things can happen to the objects. These events happen at a particular moment in time.
- Objects can participate in processes that have a temporal dimension also.
- The whole of the objects or each object separately can be in different states.
- There exist some kind of (causal) interaction between events causing changes in other events or the states of objects.

4.2 Technical Implementation

Given this ontology, we can then specify in a coherent way the appropriate technical implementations. Special attention needs to be given to the interface which should be as standardized as possible. Then the optimal, least biased, simplest and richest way to technically implement them can be specified. An example taken from economics would be to distinguish between different kinds of markets, in terms of certain criteria such as the kinds of goods traded on the market, whether we are dealing with a fixed price market or an auction market. We could also specify what different kinds of auction mechanisms are around. A similar exercise is required for describing producers, sellers, buyers, consumers, etc. When describing consumers' preferences, we can experiment with different kinds of utility functions, budget constraints, risk aversion, etc. We could also have perfectly rational economic agents or ones with bounded rationality. We can also distinguish between different kinds of products : perishable goods, standard products, luxurious products, utility products (electricity, water,...). When appropriate, it should be specified what is required as input and what comes out as a result. This information is needed to technically detail the interfaces. Does the creation of an ontology imply that there would be only one component called for instance market, client, organization, etc. ? Not at all. The researcher would rather have the choice between different kinds of markets, market clearing mechanisms, consumers, etc. It would be easy to change each component and thus study its effect on the overall behavior.

5 Advantages of a COMA-approach

What sense would it make to adopt a COMA-strategy for MAS and what can be learned from the Business Information Systems world ? The following benefits would become clear: The following technical advantages can be realized:

- Well validated and verified components: validation means answering the question whether we are building the right model and verification checks whether we are building the model right. Only after extensive testing and use in different situations can we be sure that the behavior of a certain model component actually performs the way we want and has no serious flaws. Allowing other people to use and test our model, or parts of it, will improve the quality of our solution. It is clear that an architecture based on components is a prerequisite for extensive (external) validation and verification.

– Highly robust components: when components are used in a wide variety of contexts and situations, and given feedback on problems encountered, we can be sure to end up with robust and reliable software that will perform well in different environments.

Scientific advantages could be :

– Richer models: when building MAS, developers can immediately focus on what they want to study, without the need to re-develop parts of the model which have been programmed and tested previously by others.
– Increased Reliability: when using components whose behavior is well known, we can be more certain of the obtained results or observed behaviour obtained. Unwanted or unknown side effects can be then be reduced to a minimum.
– Increased Peer Review: using standard components, models could be more easily replicated and the obtained results could be verified and extended.
– The main advantage for social scientists would be that model building then can be done in terms of concepts that are theirs and which they understand.

6 Conclusion

In this paper, we have pointed our future directions for building multi-agent systems. We took ideas which are common in the field of business information systems and applied them to MAS. The main idea is to adopt a component based architecture allowing developers to simply "plug in" new functionality. We suggest to go a step further than existing platforms by adding fully functional components. This will allow researchers to build richer but also more reliable and robust models. At the same time, the internal structure and behaviour will be much better documented and lead to a more open environment in which agents can easily enter and interact with each other. It is clear that such an advantage is also beneficial for the application developer as now the possibility of virtual worlds will exist for which the necessary navigational components will be available. In doing so, the necessary foundations for true advancements in the field of MAS are laid. Issues such as how the exact standardisation processes should look like or what ontology description language needs to be used, need to be adressed by established agent communites such as FIPA and Agentlink.

References

1. Robert Axelrod. Advancing the art of simulation in the social sciences. In R. Conte, R. Hegselmann, and P. Terna, editors, *Simulating Social Phenomena*, pages 21–40. Springer-Verlag, 1997.
2. Jeffrey Goldstein. Emergence as a construct: History and issues. *Emergence*, 1(1):49–72, 1999.
3. Duane P. Truex, Richard Baskerville, and Heinz Klein. Growing systems in emergent organizations. *Communications of the ACM*, 42(8):117–123, 1999.

4. Yannis Bakos. The emerging role of electronic marketplaces on the Internet. *Communications of the ACM*, 41(8):35–42, 1998.
5. K. Decker, K. Sycara, and M. Williamson. Middle-agents for the Internet. In *Proc IJCAI'97*, pages 578–583, 1997.
6. M. Guttman and G. Diego Lo. From objects to components. *SIGS Application Development*, 2(3):50–52, 1999.
7. Magnus Boman, Janis A. Bubenko, jr., Paul Johannesson, and Benkt Wangler. *Conceptual Modelling*. Prentice Hall, 1997.
8. P. Harmon and B. Sawyer. *UML for Visual Basic 6.0 developers*. Morgan Kaufmann Publishers, 1999.
9. V.R. Benjamin B. Chandrasekaran, J.R. Josephson. What are ontologies, and why do we need them. *IEEE Intelligent Systems & their Applications*, 14, nr 1:20–27, 1999.
10. R.V. Guha D. Lenat. *Building Large Knowledge-based Systems : Representation and Inference in the CYC Project*. Addison-Wesly, Reading Mass., 1990.
11. K. Graziano L. Silverston, W.H. Inmon. *The Data Model Resource Book*. John Wiley & Sons, New York, 1997.