Models and Methodology for Agent-Oriented Analysis and Design

Birgit Burmeister
Daimler-Benz AG, Research Systems Technology, Alt-Moabit 96a, 10559 Berlin, Germany
bur@DBresearch-berlin.de

Abstract

Agent-oriented techniques are likely to be the next significant breakthrough in software development process. They provide a uniform approach throughout the analysis, design and implementation phases in the development life cycle.

Agent-oriented techniques are a natural extension to object-oriented techniques, but while there is a whole plethora of analysis and design methods in the object-oriented paradigm, very little work has been reported on design and analysis methods in the agent-oriented community.

After surveying and examining a number of well-known object-oriented design and analysis methods, we argue that none of these methods, provide the adequate model for the design and analysis of multi-agent systems. Therefore, we propose a new agent-specific methodology that is based on and builds upon object-oriented methods. We identify three major models that need to be build during the development of multi-agent applications and describe the process of building these models.

1 Introduction

Agent-oriented programming or in more general terms agent-oriented techniques (AOT) provide a new approach that aims at supporting the whole software development process. Analysis, design, and implementation are done in a simple and natural way, at a level of abstraction more adequate to the problem to be solved. The goal of AOT is to handle all phases with a single, uniform concept, namely that of agents.

During the analysis phase the acting entities of the problem domain are identified and modelled as agents. Agents and their actions (or behavior) are refined and specified in the design phase. Finally, at the implementation phase, agents are programmed with the aid of an agent-oriented programming language or using a multi-agent development environment.

AOT are a natural extension to object-oriented techniques (OOT), that are also aimed to support all phases of software development in a general and uniform way. Agents can be seen as active objects. The differences between objects and agents, as stated by [Sho90], are (i) the structuring of the internal state of an agent by mental notions like beliefs, goals, intentions, and the like, and (ii) characterization of messages by message types and the structuring of messages into protocols. As a result of these conceptual differences, agent-oriented systems need to be analyzed, designed and implemented differently.

While there is a plethora of analysis and design methods in the area of OOT, very little has been done for analysis and design in the agent-oriented community. This paper is aimed as a con-

tribution to this area. By looking at some well-known object-oriented analysis and design methods we highlight the inadequateness of these methods for modelling multi-agent systems. Nevertheless inspired by OOT we introduce a set of models and outline the process of building these models for the development of multi-agent systems (MAS).

In section 2 we give a short survey of the general concepts in OO analysis and investigate whether these concepts can be carried over to AO analysis. Based on this in section 3 we will present our models and a methodology that builds upon and uses some notations used in OO methods. Finally, we give an overview of the related work in section 4 and conclude with a summary and an outlook in section 5.

2 Object-Oriented Analysis

There are quite a number of approaches to object-oriented analysis (and design)[1] such as the Object Modelling Technique [RBP+91], Responsibility Driven Design [WWW90], Object-Oriented Software Engineering (Objectory) [JCJ+92], Object-Oriented Design [Boo91], and the Fusion Method [CAB+94]. Abstracting away from their finer grained differences, common to most of these approaches one could identify the following step-wise procedure:

1. The first step in the majority of methods is identifying the objects and classes in the system.
2. After the objects/classes are identified, the static relationships among them are specified. These relationships are inheritance, aggregation or more general association relationships.
3. Then the dynamic relationships, (i.e. events and messages) are specified. As for the dynamics, two aspects have to be considered: (i) ‘dynamics in the large’, that is, the messages exchanged between objects that build up the system behavior, and (ii) ‘dynamics in the small’, that is, the internal flow of states and events/messages within one object (sometimes called object life cycle).
4. Finally the internal structure of the objects is described, i.e. the attributes are defined and the operations (methods) of the object are described.

The result of these steps can be seen as three sub-models of a complete model of the system to be built. These models are named *basic*, *static* and *dynamic* models in [Bal94].

- The basic model contains the objects/classes and their attributes and operations.
- The static model contains the structure of the system as described by the relationships among objects/classes as inheritance, aggregation or more general associations and grouping into subsystems.
- The dynamic model contains messages and interaction diagrams as well as objects’ life cycles and the specification of object operations.

1. Since there is a fluent borderline between analysis and design in OOT, we will use the term ”OO analysis” instead of ”OO analysis and design” throughout this paper.

After a thorough investigation of OO analysis and design techniques we found that they are not directly applicable to the development of multi-agent systems. This is basically due to their conceptual differences between objects and agents.

1. Agents have a more complex behavior and structure than objects, and in this respect they are more comparable to subsystems in some OO methodologies, (e.g., [WWW90]). Their internal structure differs from objects in that agents have a more complex underlying functional architecture such as the belief-desire-intention (BDI) architecture [RG92]. In this respect they are on a higher level of abstraction than objects.

2. Unlike objects whose internal states are defined in terms of some arbitrary attributes, the internal states of agents are defined in terms of some mental notions, like beliefs, plans and goals, which distinctly characterize the agent.

3. In contrast to objects that are rather passive entities, agents are active. Objects immediately become active through messages and in this respect they are benevolent, whereas agents act on their own behalf by following their goals, and can decide whether they act and respond to events and the messages received from other agents.

4. Agents' behaviors are described as scripts or plans by some graphical means that resemble state transition diagrams used to describe object life cycles in some OO techniques, (e.g., [RBP+91]). Agents' plans can be directly implemented using an agent programming language or an appropriate tool or environment such as DASEDIS [Bur93] or dMARS [Kin93]. Specifically using such tools, implementing an agent is mainly specifying its plans.

5. The communication of objects only looks at single messages. In AO not only messages are characterised by message types, agents dialogues with respect to specific contexts are prestructured into cooperation protocols like the ones proposed in our earlier work [BHS93].

The Responsibility Driven Design (RDD) method [WWW90] for the design of object-oriented systems comes closest to the concepts in AOT. In this method identifying and specifying responsibilities are the dominant starting point. A class inheritance hierarchy is build up using the responsibilities identified. Collaborations among objects are defined by *contracts* and *protocols* (i.e., the formal specification of method calls). Finally the system is structured by subsystems consisting of closely collaborating objects. But as was stated earlier, due to the conceptual differences between agents and object, OO methods are not immediately applicable to the design and analysis of agent-oriented systems.

This result has led us into studying and proposing a more adequate approach for agent-oriented analysis and design. The approach draws upon OO methods and uses some of the notations commonly used in many OO techniques. The next section describes this approach.

3 Agent-Oriented Analysis: Models & Methodology

Similar to the three models to be specified and built during OO analysis (i.e., basic model, static model and dynamic model), we divide the results of an agent-oriented analysis into three submodels: the *agent model*, the *organizational model* and the *cooperation model*.

- The agent model contains agents and their internal structure, described in terms of mental notions such as goals, plans and beliefs or whatever structure deems to be appropriate to an agent architecture. This model resembles the basic model of OO methods.

- The organizational model specifies the relationships among agents and agent types. These are on one hand inheritance relations (among agents and agent types, and agent types and sub- or supertypes), and on the other hand relationships among agents based on their roles in organizations. These organizations can be means for structuring a complex system into subsystems (as done in some OO techniques) or can be used to model real organizations. This model is in some respect similar to the static model, but since roles can change over time it is not genuinely static model.

- The cooperation model describes the interaction or more specifically the cooperation among agents. This model only contains the 'dynamics in the large' part of the OO dynamic model. The 'dynamics in the small' part, (i.e. the description of agent behavior), is part of the agent model.

Although these models are not strictly disjoined they can be developed separately. In the remaining part of this section we describe each of these models and outline how they may be developed. In contrast to most OO techniques however, we do not prescribe which model has to be developed first. In our experience in building MAS for real world applications the appropriate choice is ruled by the nature of the application being developed.

As a general guideline, it seems useful to start with an informal description of the scenario to be modelled before actually developing the models. This is also the first step in some of the OO techniques or is even a pre-requisite to applying the method. After this very first description one of the three models is developed, starting with the model most appropriate in the application considered.

To illustrate the methodology and how one may develop these models we will sketch an example from one of the applications we are currently developing. As part of a project investigating the possible implications of data highways and high performance computing on road traffic, we are developing a simulation system for widely decentralized and self-organized allocation of parking places. This system should allow for the simulation of different scenarios to experiment with different strategies and organizational forms and will be implemented as a multi-agent system.

The scenario to be simulated is briefly described as follows: The scenario consists of a road network, where cars move from their start node to an end node. At their end node, it is always preferred to park as close to the destination as possible. Parking places must be allocated dynami-

cally by cars negotiating among themselves (for all the parking places in the vicinity of the end node), or by consulting a node manager. In this example we will only consider the latter. A node manager is in charge of the parking places at a specified node.

3.1 Agent Model

The agent model contains the agents and their internal structure. It is built following the steps described below:

1. Identify agents and their environment.
   In many cases identifying agents and their environment is rather intuitive. Agents are the live and "active entities" in the system in that they can change their own states and their action can affect their environment, whereas the environment consists of passive elements whose state only changes by agents actions. It is quite usual for the collection of agents to grow during the analysis process. For instance, at later stages one may include agents that realize the system internal purposes. As with the RDD method, it is a good practice to create a CRC-card[2] for each agent. In the RDD method, CRC-cards are used to document classes. To document agents the CRC-card is enhanced by predefined 'attributes' like beliefs, motivations and plans identified in the remaining steps below. Other 'attributes' such as those related to the cooperation partners, must be analyzed during building the other models and added to the CRC-card afterwards.

2. For each agent define its motivations.
   Motivations (e.g., interests, preferences, responsibilities, long-term goals and so on) play an important role in the way an agent makes its decisions, and in this respect, they characterize the agent. Their role is especially important when an agent has more than one way of acting in a given situation. For instance, long-term goals lead to the selection of more concrete goals, subgoals and plans. And motivations are used to control the decision making at branching points within plans. These issues are also added to each agent's CRC-card.

3. Define the behavior of each agent.
   The behavior of agents is defined in some scheme called plans. Plans describe the sequence of actions an agent may carry out to fulfil certain motivations or react to the occurrence of certain events. Therefore plans must be related to the specific motivation or events that make them applicable and added to the CRC-cards.
   Plans can be specified in a graphical notation similar to state transition diagrams. These specifications are refined successively during the process of analysis and design towards a complete specification. As mentioned earlier, the complete specification can then be implemented directly when using an appropriate tool.

4. Define knowledge and belief.
   Finally the knowledge and belief an agent uses to execute its plans is defined and added to the CRC-card.

2. CRC stands for 'classes–responsibilities–collaborations'.

At the end of these steps there is a collection of agents with the specification of their motivations and plans in some sort of operational scheme, and the type of knowledge and belief they require to execute the plans.

In the example sketched out earlier, cars and node managers are the active entities and therefore can be modelled as agents. We introduce two types of agents, namely *car-agent* and *manager-agent*. The road network itself is not active and therefore not modelled as an agent, but is simulated as the environment of the agents.
The long-term goal of a car-agent is to drive from its start to its end node. The car-agents' preference is to park as near to its destination as possible. Node managers administer parking places at their node. Their task is to optimally satisfy the parking wishes of car-agents and cooperate with other manager-agents.
The major behaviors (plans) that can be associated to a car-agent are driving on the road network, announcing a desired parking location, and negotiating about a parking place. To be able to carry out these plans, a car-agent must have knowledge about its start and end nodes and the route to take. In case of the fully decentralized scenario, a car-agent must also have information about the availability of parking places at its destination. The manager-agents handle the parking wishes that were announced by car-agents, and administer the parking places that they are in charge of. Therefore, they need knowledge about the capacity and allocation of their parking places. Due to space limitations the successive refinement of these plans down to the operational schemes is not presented here.

3.2 Organizational Model

In this model, (roles of) agents are classified and related to each other.

1. Identify roles in the scenario
   The roles and responsibilities in the scenario are identified. If a real organization is to be modelled in the multi-agent system the roles appearing in that organization should be stated. Roles are then mapped to agents and agent types, where they may influence the motivations and therefore behavior of the agents.

2. Build an inheritance hierarchy
   Different roles or agents can be classified according to their knowledge and belief, motivations and behavior. Agents having the same beliefs, the same goals and the same behavior are instances of an agent type. As in OO techniques the common 'attributes' can be defined in the agent type and inherited by the agents. Abstract super-types can be introduced that do not have instances. The inheritance hierarchy can be represented in a notation used by OO methods, (e.g., the object diagrams of the Object Modelling technique (OMT)).

3. Structure roles into organizations
   Organizations can be used to structure a complex system into smaller subsystems. Subsystems are then parts where roles that are interacting more closely are put together. (This is also the way subsystems, clusters etc. are used in some OO techniques.)
   Another way of using organizations within the system is to model real organizations, where

roles have certain relationships to each other. The organization can also be represented by notations from OO methods, (e.g., the object model in OMT).

The organizational model gives an overview of the connections among agents and agent types, the roles and organization of agents.

The organizational model for the example scenario is quite simple. There are only two roles/ agent types, namely cars and node managers. The cars are driving and have wishes to park, these wishes can be handled by node managers. No inheritance relationships are present.

3.3 Cooperation Model

The cooperation model describes the interaction among the agents. It is built using the following steps.

1. Identify cooperations and cooperation partners
   In this step, it is stated which goal of an agent must be fulfilled by cooperation with other agents. This can be necessary to share resources, to synchronize actions or to coordinate behavior. The cooperation partners of an agent and the reason for cooperation must be entered into the agent's CRC-card.

2. Identify message types
   To carry out a cooperation we assume that agents must be able to communicate. For a meaningful communication we consider using message types. A set of predefined standard message types should be specified as proposed by KQML [FWW+94] or in our previous work [BHS93].

The result of steps 1 and 2 can be noted in a sort of collaboration or interaction graph, as used in OO techniques.

3. Define cooperation protocols
   From the interaction graphs (and sometimes from an intuitive semantics of message types) cooperations protocols can be derived. A cooperation protocol defines the possible flow of messages among cooperating agents. Where it is convenient, cooperation protocols can be built from a few basic protocols. In earlier work [BHS93], [Had96] we have identified some basic protocols for *Informing*, *Querying*, *Proposing* etc. Application specific protocols should be noted. As with behaviors (i.e., scripts or plans), protocols can also be represented in a graphical notation that is operational.

Therefore, the cooperation model states the kinds of interaction/cooperation going on among the agents and the contents exchanged by messages.

There are two types of cooperations in the parking management scenario: a) cooperation between car-agents and node managers who respectively announce and handle parking wishes, and b) cooperation between neighboring node managers who negotiate about parking places when one of the managers runs short of parking places in the area of its control. For cooperation of type (a) the corresponding message types are: *announce* a parking wish (car-agent), *confirm* the

allocation of a parking place (manager), and *propose* a different parking place (manager). For cooperation of type (b) the corresponding message types are: *inform* about allocation of parking places, *request* for a parking place, *offer* a parking place, and *reject* a request. We have designed a set of generic protocols that can handle some of these cooperation forms. Examples of these protocols can be found in [BHS93].

The three models described above together constitute the complete model of the system to be realised. This model can now be implemented directly with an appropriate tool: agent types are defined by their knowledge and beliefs, their motivations (e.g., goals) and behaviors in the form of scripts/plans, and their cooperation protocols.

4 Related Work

So far very little work has been reported in the area on the systematic and methodological analysis and design approaches for building multi-agent systems. Some of these works are briefly discussed here.

- An approach that was inspired by the KADS-model is mainly concerned with knowledge acquisition aspects [OG92]. No hints are given on how to come to an agent-oriented model of the domain.
- Some general criteria for a modular system design from OO techniques as defined in [Mey88] were transferred to multi-agent system design by [OW92].
- The project CONSENSUS [CON92] proposed a very high-level two step approach. The first step being a ‘normal’ requirements analysis as done for any software system. The second step is the design of a multi-agent system. However, again no instructions were given as how this may be done.
- The approach described in [Dor93] discusses the decisions that have to be made when developing a multi-agent system, and in this respect is closer to our approach. The decisions concern the definition of the agents, the description of their beliefs and capabilities and finally the specification of generic interactions and communication contents. However, no guidelines are provided as how appropriate structures and models may be developed to facilitate making “good” decisions.
- Recent work of Kinny and his colleagues [KGR96] is very similar to the work presented in this paper. Starting from a specific OO technique, namely OMT, they propose a methodology and modelling approach for multi-agent systems. They distinguish between an external and an internal model. The external model consists of an agent model (our organizational model) and an interaction model (our cooperation model), and is independent of a chosen agent architecture. The internal model (our agent model) is specific to the BDI agent architecture and describes the agents by a plan model, a goal model and a belief model. The internal model is a specialization of our agent model and is quite elaborated. On the other hand their interaction model is not as worked out as our cooperation model.
- In the work of Rosenschein et al., [RZ94], the main focus is on the cooperation and interaction aspects of multi-agent systems. They describe the definition of a negotiation process (as a special form of cooperation), as a three step task. Namely defining the space of

possible deals, the negotiation process (as seen from the outside) and the negotiation strategy (for each agent). These instructions can be seen as a special form of building the cooperation model.

5 Summary and Outlook

We have presented an approach for a systematic development of multi-agent systems. We have introduced three distinct models and some model building instructions.

The motivation for our work was the need for a systematic approach for developing multi-agent systems for real world applications and the fact that very little effort has been made in the AO community on this subject.

Since AOT are a specialization of OOT and there exist many OO analysis and design techniques, we examined whether they could be used for the analysis and design of multi-agent systems. We found that none of the methods considered can be used immediately, without further modifications, mainly because there are important conceptual differences between objects and agents and consequently the resulting system models. However, OO techniques provided a good starting point for our work, especially the RDD method, which is conceptually closest to AO modelling.

Our approach consists of three distinct models that are built during the process of analysis and design: The agent model, consisting of the agents and their internal structure defined in terms of knowledge and belief, motivations and behavior; the organizational model, describing the structure of the system and modelling organizations (real or artificial) agents are working in; and the cooperation model, containing the interactions among the agents, i.e., describing who interacts with whom, for what purpose, using which message types and protocols.

For each model we gave some instructions as to how the model may be built. These three models together describe the multi-agent system to be realised.

Most of the reported work on this topic is related to the agent model, so this is the most detailed model. For BDI agents in particular, we found the methodology given in [KGR96] to be a recommended alternative.

Our previous work on cooperation protocols influenced the cooperation model. Defining appropriate cooperation protocols and the way scripts or plans are used to program the agent's behavior is an important issue in the development of multi-agent systems and needs to be elaborated further.

The least elaborated model is the organizational model. The impact of a role an agent takes within an organization on its behavior has not been investigated thoroughly. Furthermore so far there is no way of directly implementing organizations by corresponding language constructs. One way would be to look at organizations as special agents, that contain special joint plans the member of the organization can use (or inherit) as Shoham proposed in his first discussion of agent-oriented programming [Sho90].

Our approach is currently being tested in a number of real world applications. This will give us more insights into the development process for multi-agent systems and thus will help to refine and improve the models and methodology.

Acknowledgement

I would like to thank the members of the ‘agents’ team at Daimler–Benz Research for their input and fruitful discussions. Special thanks to Afsaneh Haddadi for helping to make this a readable paper.

References

[Bal94] H. Balzert: *Methoden der objekt-orientierten Systemanalyse,* Angewandte Informatik 14, BI-Wissenschaftsverlag, Mannheim, 1994. (in german)

[Boo91] G. Booch: *Object-Oriented Design with Applications*, Benjamin/Cummings, Redwood City, CA, etc.,1991.

[Bur93] B. Burmeister: “DASEDIS – Eine Entwicklungsumgebung zum Agenten-Orientierten Programmieren” in: H.J. Müller (ed.): *Verteilte Künstliche Intelligenz*, BI–Wissenschaftsverlag, Mannheim, 1993. (in german)

[BHS93] B. Burmeister, A. Haddadi, K. Sundermeyer,: ”Generic Configurable Cooperation Protocols for Multi-Agent Systems” in: C. Castelfranchi, J.-P. Müller (eds.): *From Reaction to Cognition. Proc. MAAMAW’93*, LNAI 957, Springer, Berlin, etc., published 1995.

[CON92] “Methodology Report (D13)”, British Aerospace, Durham University, Cambridge Consultants, 1992.

[CAB+ 94] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, P. Jeremaes: *Object-Oriented Development. The Fusion Method*, Prentice Hall, Englewood Cliffs, 1994.

[Dor93] J. Doran: ”Using DAI Software Testbeds” in: *Proceedings CKBS’93*, DAKE Centre, Keele, 1993.

[FWW+94] T. Finin, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzson, D. McKay, J. McGuire, R. Pelavin, S. Shapiro, C. Beck: “Specification of the KQML Agent-Communication Language” (Draft), The DARPA Knowledge Sharing Initiative, External Interfaces Working Group, 1994.

[Had96] A. Haddadi: *Communication and Cooperation in Agent Systems: A Pragmatic Theory*, LNAI 1056, Springer, Berlin, etc., 1996.

[JCJ+92] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard: *Object-Oriented Software Engineering. A Use Case Driven Approach*, ACM Press/Addison-Wesley, 1992.

[Kin93] D. Kinny: “The Distributed Multi-Agent Reasoning System Architecture and Language Specification”, Australian Artificial Intelligence Institute, Melbourne, 1993.

[KGR96] D. Kinny, M. Georgeff, A. Rao: “A Methodology and Modelling Technique for Systems of BDI-Agents” in: W. van der Velde, J. Perram (eds.): *Agents Breaking Away. Proc. MAAMAW’96*, LNAI 1038, Springer, Berlin, etc.,1996.

[Mey88] B. Meyer: *Object-oriented Software Construction*, Prentice Hall, Englewood Cliffs, 1988.

[OG92] A. Ovalle, C. Garbay: “Towards a Methodology for Multi-Agent System Design” in: *Proceedings Expert Systems 92*, Cambridge, 1992.

[OW92] G.M.P. O’Hare, M.J.Wooldridge: “A Software Engineering Perspective on Multi-Agent System Design: Experience in the Development of MADE” in: N. Avouris, L. Gasser (eds.): *Distributed Artificial Intelligence - Theory and Praxis*, Kluwer Academic, Dordrecht, etc., 1992.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorenson: *Object-Oriented Modelling and Design*, Prentice Hall, Englewood Cliffs, 1991.

[RG92] A. S. Rao, M. P. Georgeff: ”An Abstract Architecture for Rational Agents” in B. Nebel, C. Rich, W. Swartout (eds.): *Proc. International Conference on Principles of Knowledge Representation and Reasoning (KR-92)*, Morgan Kaufmann, San Mateo, 1992.

[RZ94] J. S. Rosenschein, G. Zlotkin: *Rules of Encounter. Designing Conventions for Automated Negotioation among Computers*, MIT Press, Cambridge MA, London, 1994.

[Sho90] Y. Shoham: “Agent Oriented Programming”, Stanford University Technical Report STAN-CS-90-1335, Stanford, 1990.

[WWW90] R. Wirfs-Brock, B. Wilkerson, L. Wiener: *Designing Object-Oriented Software*, Prentice-Hall, Englewood Cliffs, 1990.