



UNIVERSITY OF  
CALGARY

Agent Based Software Engineering  
(SENG 609.22)  
2002 Fall

Research Tutorial

*Methodologies for Agent-Based System Analysis and Design  
(KGR, GAIA and DESIRE)*

Instructor: [B. H. Far](#)  
Student Name: Yuhang Wang  
Email address: [yuhang@cpsc.ucalgary.ca](mailto:yuhang@cpsc.ucalgary.ca)

## Methodologies for agent-based system analysis and design (KGR, GAIA and DESIRE)

### Abstract

*The interest in agent-oriented modeling techniques and methodologies has been emerged recently. A simple survey, which classifies various techniques and methodologies, and reviews three of them (KGR, GAIA, and DESIRE) and compared them, in particular comparing GAIA with two others, is given in this tutorial. Since the methodologies in this tutorial adopt some concepts from Object-oriented techniques to develop themselves, a relationship between Object-oriented techniques and Agent-oriented techniques is discussed in this tutorial also. At the last, by combining the experience obtained from a real project we finished for this course, a suitable conditions where the different methodologies are being discussed.*

### 1. Introduction

In recent times there has been a surge of interest in agent-oriented modeling techniques and methodologies. The various approaches may be roughly grouped as follows:

- Those such as [4, 24] which take existing Object-Oriented (OO) modeling techniques or methodologies as their basis, seeking either to extend and adapt the models and define a methodology for their use, or to directly extend the applicability of OO methodologies and techniques, such as design patterns, to the design of agent systems,
- Those such as [3, 17] which build upon and extend methodologies and modeling techniques from knowledge engineering, providing formal, compositional modeling languages suitable for the verification of system structure and function,
- Those which take existing formal methods and languages, for example Z [31], and provide definitions within such a framework that support the specification of agents or agent systems [26], and
- Those that have essentially been developed *de novo* for particular kinds of agent systems. CASSIOPEIA [7], for example, supports the design of Contract Net [29] based systems and has been applied to Robot Soccer.

These design methodologies may also be divided into those that are essentially top-down approaches based on progressive decomposition of behavior, usually building on some notion of role, and those such as CASSIOPEIA that are bottom-up approaches which begin by identifying elementary agent behaviors.

It seems that we have already got various methodologies in analysis and design of agent-based system. However, whether we could directly use some Object-oriented techniques in Agent-Oriented system? How and when we should use the different methodologies? What is different between various methodologies even they shared similar concepts? Based on these questions, this tutorial provides a simple method to compare three different methodologies.

## 2. Object Oriented Techniques and Agent Oriented Techniques.

### 2.1. Shortcomings of object oriented techniques

The first problem concerns the modeling of individual agents or agent classes. While there are superficial similarities between agents and objects, representing an agent as an object, i.e., as a set of attributes and methods, is not very useful because the representation is too fine-grained, operating at an inappropriate level of abstraction. An agent so represented may appear quite strange, perhaps exhibiting only one public method whose function is to receive messages from other agents.

Thus an object model does not capture much useful information about an agent, and powerful OO concepts such as inheritance and aggregation become quite useless as a result of the poverty of the representation.

There are several reasons for this problem. One is that the agent paradigm is based on a significantly stronger notion of encapsulation than the object paradigm. An agent's internal state is usually quite opaque and, in some systems, the behaviors that an agent will perform upon request are not even made known until it advertises them within an active system. Related to this is the key characteristic of autonomy: agents cannot normally be created and destroyed in the liberal manner allowed within object systems and they have more freedom to determine how they may respond to messages, including, for example, by choosing to negotiate some agreement about how a task will be performed. As the underlying communication model is usually asynchronous there is no predefined notion of flow of control from one agent to another: an agent may autonomously initiate internal or external behaviors at any time, not just when it is sent a message. Finally, an agent's internal state, including its knowledge, may need to be represented in a manner that cannot easily be translated into a set of attributes; in any case to do so would constitute a premature implementation bias.

The second problem concerns the power of object models to adequately capture the relationships that hold between agents in a multi-agent system. While the secondary models in common use in OO methodologies such as use cases and interaction diagrams may usefully be adapted (with somewhat different semantics), the Object Model, which constitutes the primary specification of an OO system, captures associations between object classes that model largely static dependencies and paths of accessibility which are largely irrelevant in a multi-

agent system. Only the instantiation relationship between classes and instances can be directly adopted. Important aspects of relationships between agents such as their repertoire of interactions and their degree of control or influence upon each other are not easily captured. The essential problem here is the uniformity and static nature of the OO object model. An adequate agent model needs to capture these relationships between agents, their dynamic nature, and perhaps also relationships between agents and non-agent elements of the system, including passive or abstract ones such as those modeled here as resources. Both of these are problems concerning the suitability of OO modeling techniques for modeling a multi-agent system. Another issue is the applicability of OO methodologies to the process of analyzing and designing a multi-agent system. OO methodologies typically consist of an iterative refinement cycle of identifying classes, specifying their semantics and relationships, and elaborating their interfaces and implementation. At this level of abstraction, they appear similar to typical AO methodologies, which usually proceed by identifying roles and their responsibilities and goals, developing an organizational structure, and elaborating the knowledge and behaviors associated with a role or agent.

However, this similarity disappears at the level of detail required by the models, as the key abstractions involved are quite different. For example, the first step of object class identification typically considers tangible things, roles, organizations, events and even interactions as candidate objects, whereas these need to be clearly distinguished and treated differently in an agent-oriented approach. The uniformity and concreteness of the object model is the basis of the problem; OO methodologies provide guidance or inspiration rather than a directly useful approach to analysis and design.

## 2.2. AO techniques extending from OO techniques

Managing complexity is perhaps the most challenging task facing designers of large-scale embedded software systems. It is now widely accepted that the construction of reliable, maintainable, and extensible systems that conform to their specifications requires, inter alia, the use of design methodologies and modeling techniques that support abstraction, structuring, inheritance, modularity, and other mechanisms for managing their inherent complexity.

Perhaps foremost amongst the methodologies that have been developed for the design, specification, and programming of conventional software systems are various Object-Oriented. OO approaches, based upon the central notion of objects that encapsulate state information as a collection of data values and provide behaviours via well-defined interfaces for operations upon that information. OO methodologies guide the key steps of object identification, design, and refinement, permitting abstraction via object classes and inheritance within class hierarchies. OO methodologies provide a uniform paradigm, which is useful across a range of system scales and implementation languages. They have achieved a considerable

degree of maturity, and there is widespread acceptance of their advantages. A large community of software developers familiar with their use now exists, The OO design and development environment is well supported by diagram editors and visualization tools, There is even, perhaps, a convergence of viewpoints amongst the major proponents of different variations on the OO theme.

The agent paradigm in AI is based upon the notion of reactive, autonomous, internally, motivated entities embedded in changing, uncertain worlds which they perceive and in which they act, It supports a flourishing research community which has made substantial progress in recent years in providing a theoretical and practical understanding of many aspects of agents and multi-agent systems. There is a multitude of viewpoints on what exactly constitutes an agent, on how they should be structured, and how collections of agents interacting with each other and the environment can be used to implement complex systems. Despite this healthy lack of consensus, the benefits of implementing agent systems are little disputed, and several agent architectures have progressed to being useable technologies.

As yet, however, there are not many examples of the successful application of agent system technologies on a significant scale, In part this is due to the absence of mature languages and software tools, but our experience suggests that the absence of methodologies that allow system complexity to be electively managed is a greater obstacle. Even like what we said above, OO methodologies are not directly applicable to agent systems, agents are usually significantly more complex than typical objects, both in their internal structure and in the behaviors they exhibit. We believe that if multi-agent systems are to become widely accepted as a basis for large-scale commercial and industrial applications, adequate design methodologies and modeling techniques will be essential. This is not just to guarantee that such systems are sufficiently reliable, maintainable, and conformant, but to allow their design, implementation, and maintenance to be carried out by software analysts and engineers rather than AI researchers, Others have reached similar conclusions about the need for familiar, intuitive modeling techniques [Pan and Tenenbaum 1991],

The design, implementation and theoretical understanding of a particular Belief-Desire-Intention (BDI), agent architecture [George and Lansky,1986, Rao and George,1991] which has now achieved considerable maturity. And David Kinny, Michael George and Anand Rao developed suitable methodologies and models. Their approach, pragmatically motivated, has been to explore how existing OO modeling techniques can be extended to apply to BDI agent systems. By building upon and adapting existing well-understood techniques, they take advantage of the maturity of the OO approach and aim to develop models and a methodology that will be easily learnt and understood by those familiar with the OO paradigm.

A number of agent-oriented systems based on BDI architectures have been developed, including PRS [George, and Lansky, 1986], COSY [Burmeister and

Sundermeyer, 1992], GRATE [Jennings, 1993], ARCHON [Jennings et al, 1995], INTERRAP [Muller et al, 1995], and DMARS [Kinny, 1993], Likewise, agent-oriented languages, such as AGENT [Shoham, 1991, Shoham, 1993] and PLACA [Thomas, 1995] have been proposed as languages for programming agents.

However, there is very little work on how to analyze and design agent-oriented systems, Brazier et al, [Brazier et al, 1995] discuss a formal specification framework for multi-agent system design, called DESIRE (Design and Specification of Interacting Reasoning components), DESIRE emphasizes hierarchical task-based decomposition, It captures the task structure, sequences of subtasks, information exchange between tasks, knowledge decomposition, and role delegation. Gaia and KGR [23, 24] approach emphasize the importance of goals and services as opposed to tasks.

### 3. KGR, GAIA and DESIRE Techniques

#### 3.1. KGR Approach

The KGR approach was developed to fulfill the need for a principled approach to the specification of complex multi-agent systems based on the belief desire-intention (BDI) technology of the Procedural Reasoning System (PRS) and the Distributed Multi-Agent Reasoning System (DMRS) [8, 25]. A key motivation of the work was to provide useful, familiar mechanisms for structuring and managing the complexity of such systems. KGR approach is a specialized set of models which operate at two distinct levels of abstraction. Firstly, from the external viewpoints, the system is decomposed into agents, modeled as complex objects characterized by their purpose, their responsibilities, the services they perform, the information they require and maintain, and their external interactions. Secondly, from the internal viewpoint, the elements required by particular agent architecture must be modeled for each agent; these are an agent's beliefs, goals, and plans.

KGR methodology advocates the decomposition of a system based on the key roles in an application. The identification of roles and their relationships guides the specification of an agent class hierarchy; agents are particular instances of these classes. Analysis of the responsibilities of each agent class leads to the identification of the services provided and used by an agent, and hence its external interactions. Consideration of issues such as the creation and duration of roles and their interactions determines control relationships between agent classes. These details are captured in two models:

- An Agent Model describes the hierarchical relationship among different abstract and concrete agent classes and identifies the agent instances which may exist within the system, their multiplicity, and when they come into existence.

An agent class model is a directed, acyclic graph containing nodes denoting both abstract and concrete, instantiable, agent classes. Agent classes are represented by class icons, and abstract classes are distinguished by the presence of the adornment (A) in the upper section of the icon. Edges in the graph denoting inheritance are distinguished by a triangle with a vertex pointing towards the super class, and edges denoting aggregation by a diamond adjacent to the aggregate class. Other associations between agent classes are not allowed. Agent classes may have attributes, but not operations. Attributes may not be arbitrary user-named data items, rather they are restricted to a set of predefined reserved attributes. For example, each class may have associated belief, goal and plan models, specified by the attributes **beliefs**, **goals**, and **plans**.

For example, Figure 1 shows a simplified combined agent diagram from the Air Traffic Management application domain. Note that the attributes of agent classes do not appear in this diagram:

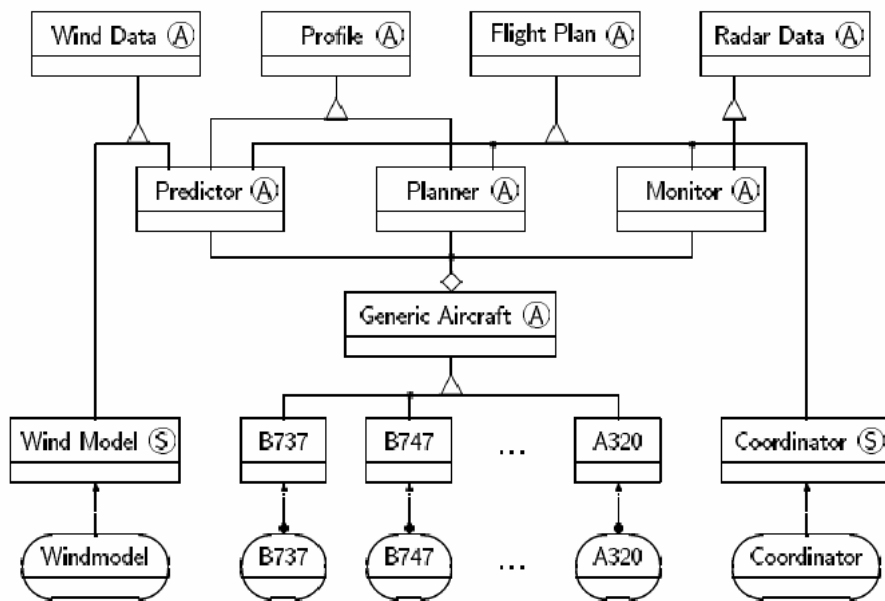


Figure 1: ATM Agent Class and Instance Diagram

- An Interaction Model describes the responsibilities of an agent class, the services it provides, associated interactions, and control relationships between agent classes, This includes the syntax and semantics of messages used for inter-agent communication and communication between agents and other system components, such as user interfaces.

The BDI paradigm provides a strong notion of agency, agents are viewed as having certain mental attitudes, Beliefs, Desires and Intentions, which represent respectively their informational, motivational and deliberative states. In BDI architecture an agent can be completely specified by the events that it can perceive, the actions it may perform, the beliefs it may hold, the goals it may

adopt, and the plans that give rise to its intentions. These are captured, for each agent class, by the following models:

- A Belief Model describes the information about the environment and internal state that an agent of that class may hold, and the actions it may perform. The possible beliefs of an agent and their properties, such as whether or not they may change over time, are described by a belief set. In addition, one or more belief states, particular instances of the belief set, may be defined and used to specify an agent's initial mental state.
- A Goal Model describes the goals that an agent may possibly adopt, and the events to which it can respond. It consists of a goal set which specifies the goal and event domain and one or more goal states, sets of ground goals, used to specify an agent's initial mental state.
- A Plan Model describes the plans that an agent may possibly employ to achieve its goals. A plan model consists of a set of plans, known as a plan set. Individual plans are specified as plan diagrams, which are denoted by a form of class icon. A generic plan diagram appears in Figure 2. The lower section, known as the plan graph, is a state transition diagram, similar to an OO dynamic model. Unlike OO approaches, however, plans are not just descriptions of system behavior developed during analysis. Rather, they are directly executable prescriptions of how an agent should behave to achieve a goal or respond to an event.

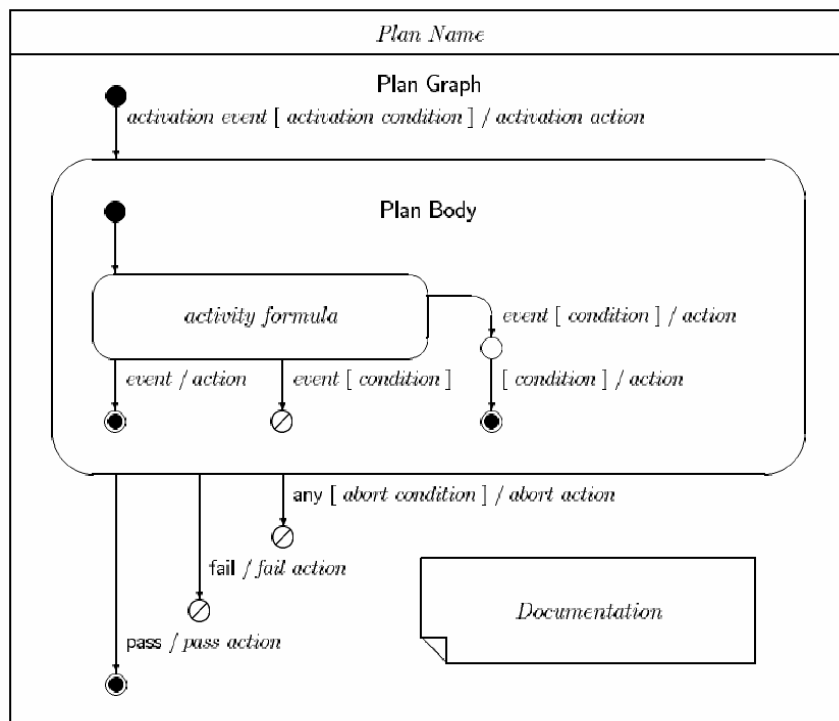


Figure 2: Generic Plan Diagram

The elements of the plan graph are three types of node, start states, end states and internal states, and one type of directed edge, transitions. Start states are denoted by a small filled circle (●). End states may be pass or fail states, denoted respectively by a small target (⊙), or a small no entry sign (⊘).

Internal states may be passive or active. Passive states have no substructure and are denoted by a small open circle (○). Active states have an associated activity and are denoted by instance icons. Activities may be sub goals, denoted by formulae from the agent's goal set, iteration constructs, including do and while loops, or in the case of a graph state, an embedded graph called a sub graph. Events, conditions, and actions may be attached to transitions between states. In general, transitions from a state occur when the associated event occurs, provided that the associated condition is true. When the transition occurs any associated action is performed, Conditions are predicates from the agent's belief set, Actions include those defined in the belief set, and built-in actions, The latter include assert and retract, which update the agent's belief state.

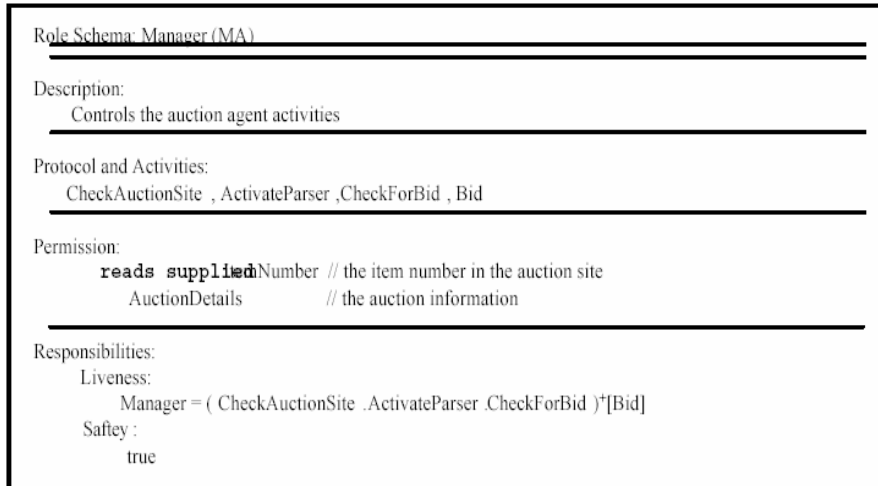
### 3.2. GAIA Approach

GAIA, which extends Agent-Oriented Methodology (AOM), is methodology that concentrates on the modeling aspect of agent-based systems. GAIA is based on a set of models, which are used in different stages of the agent development (both analysis and design). Following the GAIA guidelines, the analysis of an intended agent-based system results in the definition of roles, which are characterized by three attributes - permissions, responsibilities and protocols - and an interaction model which depicts interactions between roles:

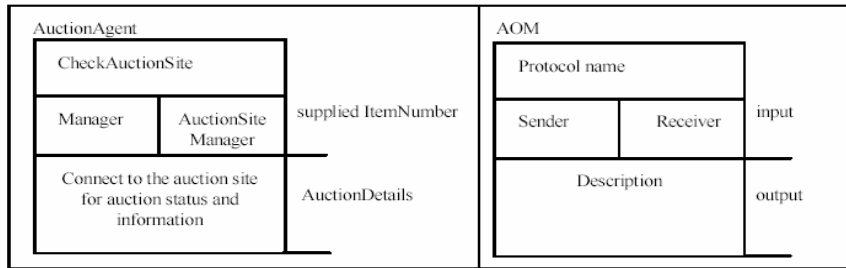
- The permissions attribute states what resources may be used to carry out the role and what resource constraints the role's executor is subject to.
- The responsibilities attribute determines the functionality of the role. This functionality is expressed in terms of safety and liveness properties. The safety properties indicate what conditions must be true through all of the states of the system. The safety properties are invariant. The liveness properties indicate what sequence of states the role's lifecycle consists of, and what alternative computational paths are available for the role.
- The protocols attribute states the interactions of the role with other roles. In addition it states the internal activities of the role. GAIA distinguishes between external activities (referred to as protocols) and internal ones. The internal activities will be elaborated upon as a part of the service model later in this section.

Each role has a schema, which describes the permissions, responsibilities and protocols attributes. An example of role schemata is depicted in Figure 3. In this Figure, the *Manager's* role schema (of the case study Auction Agent) consists of a description of the way in which the *Manager* should control the agent's activities. The *Manager's* protocols and activities consist of the *CheckAuctionSite* protocol, the *ActivateParser* activity, the *CheckForBid* activity and the *Bid* protocol. The *Manager's* permission is comprised of (a) read access to the *AuctionDetails* (which is a part of the internal structure of the agent); (b) supplying of the *ItemNumber*. The responsibilities are separated into two categories: safety and liveness. The *true* statement that appears at the safety properties entry indicates that there are no constraints on the role. As for the liveness properties, the statement indicates that the order of execution of the protocols and activities is as following: after the protocol of *CheckAuctionSite* is executed the *ActivateParser* activity is performed. Following, the *CheckForBid* activity is performed. This sequence of protocols and activities occurs one or more times. After that sequence, the *Bid* Protocol may occur. The interaction model is used for a protocol description.

Role Schema: AuctionSiteManager (ASM)	
Description: Manage the auction site	
Protocol and Activities: CheckAuctionSite , Bid, ValidateUser	
Permission:	
<b>reads supplied</b>	ItemNumber // the item number in the auction site
	AuctionDetails // the auction information
<b>generates supplied</b>	ItemNumber // the item number in the auction site
<b>supplied</b>	Price // the price for the Bid
<b>supplied</b>	User // the user make the Bid
	Bid // new Bid entity or nil
Responsibilities:	
Liveness:	AuctionSiteManager = ( CheckAuctionSite )+   ( ValidateUser )*   ( ValidateUser .Bid )*
Safety :	User=illegal => Bid=nil



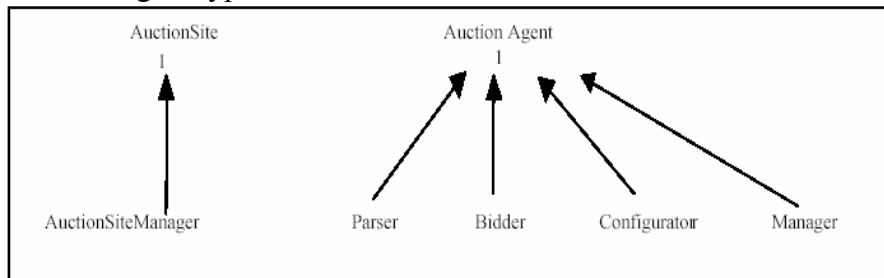
**Figure 3: the Manager and the AuctionSiteManager schemata**



**Figure 4: The Interaction Model of the CheckAuctionSite Protocol**

An example of the interaction model is depicted in Figure 4. The Figure shows the generic interaction model of GAIA and the case study model. In the case study model, there is a description of the *CheckAuctionSite* protocol, which consists of the roles of *Manager* and the *AuctionSiteManager*. The *Manager* supplies the *ItemNumber* and the *AuctionSiteManager* returns the *AuctionDetails*. The GAIA design phase consists of three models. These models are derived from the role and the interaction models built in the analysis stage:

- The agent model defines roles, which are carried out by each agent type. Each agent type has an indication of its instances number. The agent model is depicted in Figure 5. The Figure shows the roles assigned to the agent types and the agent types instances number.



**Figure 5: the Agent model represents the Auction Agent**

- The service model describes the functionality of the agent by extending the protocols and responsibilities models from the analysis stage. Each service includes the documentation of its input, output, pre-condition and post-condition. The service model is depicted in Figure 6. It presents a subset of the agent's services. These are derived from the *Manager*, the *AuctionSiteManager* and the interaction model.

Service	Input	Output	Pre condition	Post Condition
get auction details	ItemNumber	AuctionDetails	true	True
Validate user	User	Exist	true	(exist=true) $\vee$ (exist=false)
Bid	User, ItemNumber, Price	Success	user exist	(success=true) $\vee$ (success=false)

**Figure 6: The Service Model**

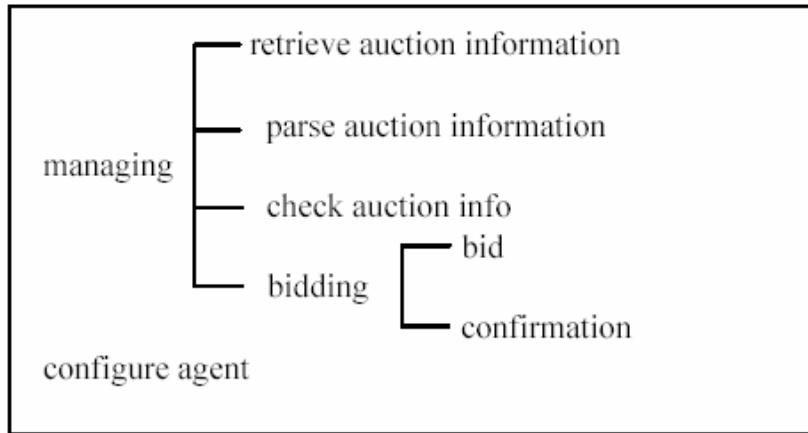
- The acquaintance model describes the communication path between the agent types. This is graphically expressed by a directed graph as depicted in Figure 7. The nodes in the graph represent agent types whereas the vertices represent the communication path. In the case study acquaintance model, there is a bi-directional communication path between the AuctionSite and the AuctionAgent.



**Figure 7: The Acquaintance Model**

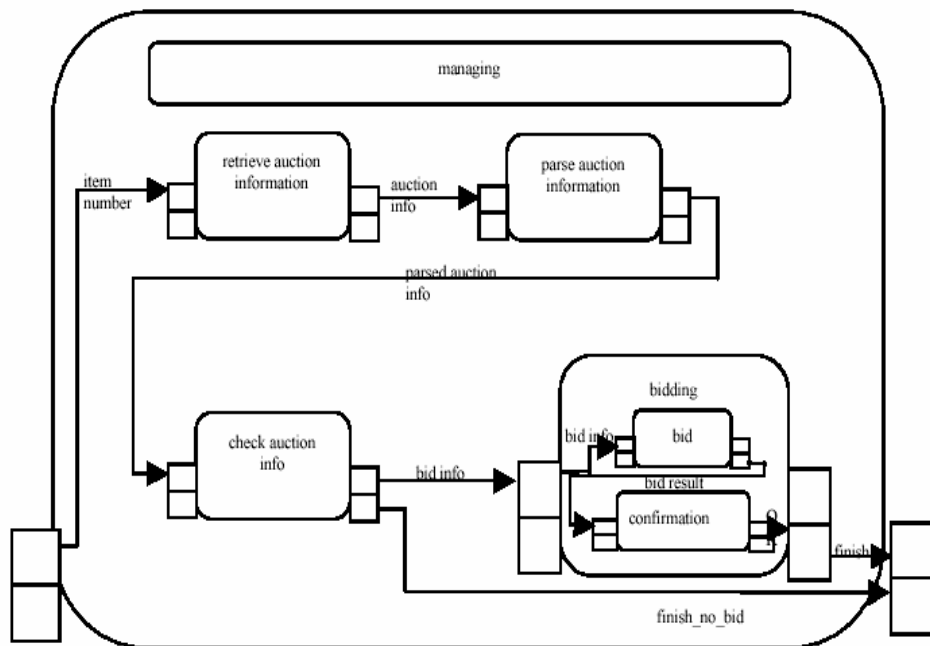
### 3.3. DESIRE Approach

The Design and Specification of Interacting Reasoning (DESIRE) framework is a complete environment for design and implementation of MAS. It allows the system designer to explicitly and precisely specify both the intra-agent and inter-agent functionality. In DESIRE, the following models are supported: (1) task (de-)composition, (2) information exchange, (3) sequencing of (sub-) tasks, (4) subtask delegation and (5) knowledge structure.



**Figure 8: Task Hierarchy of the Auction Agent**

The task (de-) composition model includes the information about the task hierarchy, the task input, the task output and the relationships between tasks. Each task in the hierarchy can be primitive as well as composed. Additional information regarding the task composition model is encapsulated within the information exchange model. The task hierarchy model of the Auction Agent is depicted in Figure 8. In this Figure, the *managing* task consists of the *retrieve auction information*, *parse auction information*, *check auction info* and *bidding* tasks. The *bidding* task consists of the *bid* and *confirmation* tasks. The tasks which do not consist of other tasks are primitive.



**Figure 9: the component managing of the Auction Agent**

```

if component-state(managing,start)
then next-component-state(retrieve_auction_info,active)
      and next-target-set(retrieve_auction_info,get-info)
      and next-link-state(item_number,awake)
      and next-link-state(auction_info,awake)

if evaluation(retrieve_auction_info,get-info,succeed)
then next-component-state(parse_auction_info)
      and next-target-set(parse_auction_info, get_info)
      and next-link-state(parsed_auction_info,awake)

if evaluation(parse_auction_info,get-info,succeed)
then next-component-state(check_auction_info)
      and next-target-set(check_auction_info, get_bid_info)
      and next-link-state(bid_info,awake)
      and next-link-state(finish_no_bid ,awake)

if evaluation(check_auction_info,get_bid_info,succeed)
then next-component-state(bid)
      and next-target-set(bid, place_bid)
      and next-link-state(finish,awake)

```

**Figure 30: The task control knowledge of managing**

In DESIRE, the information exchange between tasks is specified as information links between components. Each information link directs the output of one component to the input of another one. The information exchange model is depicted in Figure 30. This Figure shows the task hierarchy that was previously presented in Figure 9. The additional information is the information links that depict the data flow. For example, the link *ItemNumber* indicates that there is an input to the *retrieve auction information* task. Its output is fed as input, via the *auction info* link to the *check auction info* task. Note that the information exchange model has a formal syntax. Task sequencing is explicitly modeled within the component as task control knowledge, which can be seen in Figure 30. This knowledge includes the order of subtasks, their activation target (usually referred to as a goal in the context of agents) and the amount of effort, which can be afforded. The target is the focus of the activity of the component and the effort is the component and the link states. For example, if the task *parse\_auction\_information* succeeds then the next task should be *check\_auction\_info* and its target should be *get\_bid\_info*. In addition, DESIRE awakes the relevant links that should be activated if the *check\_auction\_info* task is successful. DESIRE has the additional advantage that the specifications and their semantics can be expressed formally, using temporal logic.

#### 4. Comparison among these three approaches

##### 4.1. Comparison GAIA with the KGR approach

The first and most obvious difference between GAIA approach and KGR is one of scope. GAIA methodology does not attempt to unify the analysis and abstract design of a multi-agent system with its concrete design and implementation with a particular agent technology, regarding the output of the analysis and design process as an abstract specification to which traditional lower-level design methodologies may be applied. KGR, by contrast, makes a strong architectural commitment to BDI architectures and proposes a design elaboration and refinement process that leads to directly executable agent specifications. Given the proliferation of available agent technologies, there are clearly advantages to a more general approach, as GAIA. However, the downside is that it cannot provide a set of models, abstractions and terminology that may be used uniformly throughout the system life cycle. Furthermore, there may be a need for iteration of the AO analysis and design process if the lower-level design process reveals issues that are best resolved at the AO level. A research problem for GAIA and others like it is whether and how the adequacy and completeness of its outputs can be assessed independently of any traditional design process that follows.

A second difference is that in this work a clear distinction is made between the analysis phase, in which the roles and interaction models are fully elaborated, and the design phase, in which agent services and acquaintance models are developed. The KGR approach does not make such a distinction, proposing instead the progressive elaboration and refinement of agent and interaction models, which capture respectively roles, agents and services, and interactions and acquaintances. While both methodologies begin with the identification of roles and their properties, here we have chosen to model separately abstract agents (roles), concrete agents and the services they provide. KGR, on the other hand, employs a more uniform agent model which admits both abstract agents and concrete agent classes and instances and allows them to be organized within an inheritance hierarchy, thus allowing multiple levels of abstraction and the deferment of identification of concrete agent classes until late in the design process.

While both approaches employ responsibilities as an abstraction used to decompose the structure of a role, they differ significantly as to how these are represented and developed. Here responsibilities consist of safety and liveness properties built up from already identified interactions and activities. By contrast, KGR treats responsibilities as abstract goals, triggered by events or interactions, and adopts a strictly top-down approach to decomposing these into services and low level goals for which activity specifications may be elaborated. There are similarities however, for despite the absence of explicit goals in our approach, safety properties may be viewed as maintenance goals and liveness properties as goals of achievement. The notion of permissions, however, is absent from the KGR approach, whereas the notion of protocols may be developed to a much greater degree of detail, for example as in [22]. These protocols are employed as more generic descriptions of behavior that may involve entities not modeled as agents.

## 4.2. Comparison GAIA with the DESIRE approach

Compared to DESIRE approach, the GAIA approach has its advantage and disadvantage. The Advantage of GAIA is the level of comprehensible. The GAIA approach is easy to understand and use due to its simple models and their clarity. The Analysis and Design phases are distinctly defined. However, DESIRE has a wide range of modeling capabilities, which on the downside cause the difficulty to learn and implement it. Therefore, for the novice who will analysis and design the Agent-oriented system, the GAIA will be a better choice.

On the other side, in terms of applying in the complexity agent-based system, DESIRE models enable one to describe the required complexity of a software agent. DESIRE, somewhat like KGR, provides the inter-communication description and the description for agent intra-communication. While in GAIA, there is no hierarchical presentation or any other mechanism for complexity management. The system description is flat.

## 5. Conclusion

In this tutorial, an introduction for the different types of agent-based system analysis/design is presented. The relationship between Gaia [1] and other approaches based that build upon OO techniques, in particular the KGR approach [23, 24] developed by [D. Kinny, M. Georgeff and A. Rao, 1996] is also described. In section 2, why OO modeling techniques and design methodologies themselves are not directly applicable to multi-agent system design and why we need to extend OO modeling techniques to OA modeling methodology is summarized. In section 3, the KGR, Gaia Approach and Desire are introduced briefly. In section 4, the comparison between Gaia and KGR, the comparison between Gaia and Desire are discussed.

We could see that the GAIA is the simplest one which could be easily used but covered the minimum area on the agent-based system analysis and design, when compared to other two complexities approached. Therefore, it is a pretty good on for novice in the agent-based system area to learn and practice. In our project which is done in this term, we used GAIA to analysis and design our project. We really found that when we want to touch in-depth design, the GAIA is not enough. So we turned to OO approach to develop the detail information. If we could use DESIRE approach, maybe we could go further in the project in the agent-based system design, however, the trade-off is that we need put more effort to learn how to apply the DESIRE approach before we kicked off the project. And for this project, the KGR approach seems a little bit more complexity than what we need.

For GAIA approach, there has emerged another methodology named "ROADMAP"[37], which provide more models (such as use case model, knowledge model and environmental model) in the approach to enhance the GAIA in the

application in more open complexity agent-based system. That's a further research area on GAIA.

**Reference:**

1. A. H. Bond and L. Gasser (Eds.), *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers: San Mateo, CA, 1988.
2. G. Booch, *Object-Oriented Analysis and Design*, 2nd ed., Addison-Wesley: Reading, MA, 1994.
3. F. Brazier, B. Dunin-Keplicz, N. R. Jennings, and J. Treur, "Formal specification of multi-agent systems: a real-world case" in *Proc. First Int. Conf. on Multi-Agent Systems (ICMAS-95)*, San Francisco, CA, June 1995, pp. 25-32.
4. B. Burmeister, "Models and methodologies for agent-oriented analysis and design" in *Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems*, K. Fisher (Ed.), 1996, DFKI Document D-96-06.
5. C. Castelfranchi, "Commitments: from individual intentions to groups and organizations" in *Proc. First Int. Conf. on Multi-Agent Systems (ICMAS-95)*, San Francisco, CA, June 1995, pp. 41-48.
6. D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes, *Object-Oriented Development: The fusion Method*. Prentice Hall International: Hemel Hempstead, England, 1994.
7. A. Collinot, A. Drogoul, and P. Benhamou, "Agent oriented design of a soccer robot team" in *Proc. Second Int. Conf. on Multi-Agent Systems (ICMAS-96)*, Kyoto, Japan, 1996.
8. M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge, "A formal specifications of dMARS" in *Intelligent Agents IV (LNAI vol. 1365)*, M. P. Singh, A. Rao, and M. J. Wooldridge, (Eds.), Springer-Verlag: Berlin, Germany, 1997, pp. 155-176.
9. J. Ferber, and O. Gutknecht, "A meta-model for the analysis and design of organizations in multi-agent systems" in *Proc. Third Int. Conf. on Multi-Agent Systems (ICMAS-98)*, Paris, France, 1998, pp. 128-135.
10. M. Fisher and M. Wooldridge, "On the formal specification and verification of multi-agent systems" *Int. J. Cooperative Inf. Syst.*, vol. 6(1), pp. 37-65, 1997.
11. The Foundation for Intelligent Physical Agents, see <http://www.fipa.org/>.
12. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, Addison-Wesley: Reading, MA, 1995.
13. L. Gasser. "Social conceptions of knowledge and action: DAI foundations and open systems semantics" *Artif. Intell.*, vol. 47, pp. 107-138, 1991.
14. L. Gasser, C. Braganza, and N. Hermann, "MACE: A flexible testbed for distributed AI research" in *Distributed Artificial Intelligence*, M. Huhns, Ed., Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1987, pp. 119-152.
15. C. Hewitt. "Open information systems semantics for distributed Artificial intelligence" *Artif. Intell.*, vol. 47, pp. 79-106, 1991.
16. C. A. Iglesias, M. Garijo, and J. C. Gonzalez, "A survey of agent-oriented methodologies" in *Intelligent Agents V - Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98)*, Lecture

- Notes in Artificial Intelligence*, J. P. Muller, M. P. Singh, and A. S. Rao, (Eds.), Springer-Verlag: Heidelberg, 1999.
17. C. Iglesias, M. Garijo, J. C. Gonzalez, and J. R. Velasco, "Analysis and design of multi-agent systems using MAS-CommonKADS" in *Intelligent Agents IV (LNAI Volume 1365)*, M. P. Singh, A. Rao, and M. J. Wooldridge, (Eds.), Springer-Verlag: Berlin, Germany, 1998, pp. 313-326.
  18. T. Ishida, L. Gasser, and M. Yokoo, "Organization self design of production systems" *IEEE Tran. Knowledge Data Eng.*, vol. 4(2), pp. 123-134, April 1992.
  19. N. R. Jennings, J. Corera, I. Laresgoiti, E. H. Mamdani, F. Perriolat, P. Skarek, and L. Z. Varga, "Using ARCHON to develop real-world DAI applications for electricity transportation management and particle acceleration control" *IEEE Expert*, vol. 11(6), pp. 60-88, December 1996.
  20. N. R. Jennings, P. Faratin, M. J. Johnson, T. J. Norman, P. O'Brien, and M. E. Wiegand, "Agent-based business process management" *Int. J. Cooperative Inf. Syst.*, vol. 5(2-3), pp. 105-130, 1996.
  21. C. B. Jones, *Systematic Software Development using VDM*, 2nd ed., Prentice Hall: Englewood Cliffs, NJ, 1990.
  22. D. Kinny, "The AGENTIS agent interaction model" in *Intelligent Agents VóProc. Fifth Int. Workshop on Agent Theories, Architectures, and Languages (ATAL-98), Lecture Notes in Artificial Intelligence*, J. P. Muller, M. P. Singh, and A. S. Rao (Eds.), Springer-Verlag: Heidelberg, 1999.
  23. D. Kinny and M. Georgeff, "Modeling and design of multi-agent systems" in *Intelligent Agents III (LNAI Vol. 1193)*, J. P. Muller, M. Wooldridge, and N. R. Jennings (Eds.), Springer-Verlag: Berlin, Germany, 1997, pp. 1-20.
  24. D. Kinny, M. Georgeff, and A. Rao, "A methodology and modeling technique for systems of BDI agents" in *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modeling Autonomous Agents in a Multi-Agent World, (LNAI Vol. 1038)*, W. Van de Velde and J. W. Perram, (Eds.), Springer-Verlag: Berlin, Germany, 1996, pp. 56-71.
  25. D. Kinny, *The Distributed Multi-Agent Reasoning System Architecture and Language Specification*, Australian AI Institute, Level 6, 171 La Trobe Street, Melbourne, Australia, 1993.
  26. M. Luck, N. Griffiths, and M. d'Inverno, "From agent theory to agent construction: A case study" in *Intelligent Agents III (LNAI Vol. 1193)*, J. P. Muller, M. Wooldridge, and N. R. Jennings (Eds.), Springer-Verlag: Berlin, Germany, 1997, pp. 49-64.
  27. A. Pnueli, "Specification and development of reactive systems" in *Information Processing 86*, Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1986.
  28. Y. Shoham, "Agent-oriented programming" *Artif. Intell.*, 60(1), pp. 1-92, 1993.
  29. R. G. Smith, "The CONTRACT NET: a formalism for the control of distributed problem solving" in *Proc. Fifth Int. Joint Conf. Artificial Intelligence (IJCAI-77)*, Cambridge, MA, 1977.
  30. R. G. Smith, *A Framework for Distributed Problem Solving*, UMI Research Press, 1980.

31. M. Spivey, *The Z Notation*, 2nd ed., Prentice Hall International: Hemel Hempstead, England, 1992.
32. M. Wooldridge, "Agent-based software engineering" *IEEE Proc. Software Eng.*, 144(1):26-37, February 1997.
33. M. Wooldridge and N. R. Jennings, "Intelligent agents: theory and practice" *Knowledge Eng. Rev.*, vol. 10(2), pp. 115-152, 1995.
34. M. Wooldridge and N. R. Jennings, "Pitfalls of agent-oriented development" in *Proc. Second Int. Conf. on Autonomous Agents (Agents 98)*, Minneapolis/St Paul, MN, May 1998, pp. 385-391.
35. M. Wooldridge, N. R. Jennings, and D. Kinny, "A methodology for agent-oriented analysis and design" in *Proc. Third Int. Conf. on Autonomous Agents (Agents 99)*, Seattle, WA, May 1999, pp. 69-76.
36. O. Shehory and A. Sturm, "Evaluation of Modeling Techniques for Agent-Based Systems"
37. T. Juan, A. Pearce, and L. Sterling, "ROADMAP: Extending the Gaia Methodology for Complex Open Systems"