A Framework For Designing, Modeling and Analyzing Agent Based Software Systems

Krishna M. Kavi, The University of North Texas
Mohamed Aborizka, The University of Alabama in Huntsville
and David Kung, The University of Texas at Arlington

Abstract

The agent paradigm is gaining popularity because it brings intelligence, reasoning and autonomy to software systems. Agents are being used in an increasingly wide variety of applications from simple email filter programs to complex mission control and safety systems. However there appears to be very little work in defining practical software architecture, modeling and analysis tools that can be used by software engineers. This should be contrasted with object-oriented paradigm that is supported by models such as UML and CASE tools that aid during the analysis, design and implementation phases of object-oriented software systems. In our research we are developing a framework and extensions to UML to address this need. Our approach is rooted in the BDI formalism, but stresses the practical software design methods instead of reasoning about agents. In this paper we describe our preliminary ideas

Index Terms: Agent-Oriented programming, Object-Oriented programming, BDI, UML

1. Introduction

There appears to be very little work in defining software architecture, modeling and analysis tools that can be used by software engineers. This should be contrasted with object-oriented paradigm that is supported by modeling languages such as UML and a variety of CASE tools that aid during the analysis, design, implementation and validation phases of object-oriented software systems: all of which contributed to the universal acceptance of object-oriented paradigm. Only recently there have been a few proposals for Agent-oriented Software Engineering (see, [15],[17]) and extensions to UML (e.g., AUML [2], [8]). In our research, we are developing a framework and extensions to UML to address this need. Our approach is rooted in the BDI formalism, but stresses practical software design methods instead of reasoning theories. In this paper we describe our preliminary work on the framework and illustrate its usefulness. Our approach, when fully developed, can be customized to specific application domains by modifying domain specific aspects of our framework (specification level annotations, patterns, templates and reasoning mechanisms). It is our hope that our work and other related work will lead to the codification of a systematic framework for the design of agent-oriented software systems, just as the efforts of Booch, Raqmbaugh and Jacobson have led to UML for object-oriented systems.

1.1. Agents versus Objects

Agent-oriented programming draws heavily from object-oriented paradigms, but also introduces a number of new concepts that are alien to object-oriented programming. *We contend that agent-oriented programming is the next expected evolution of object-oriented programming.* The reader is referred to [3], [6] or [10] for detailed and formal definition of object-oriented methodology.

Although there is no accepted definition of “agent-based” or “agent-oriented” programming, there is a generally accepted list of characteristics associated with agents: situatedness, autonomy and flexibility [5]. The situatedness implies that agents receive input from an environment and perform actions that may change the environment. Autonomy implies that the software system should operate without the direct intervention of human being or other agents. Agents must be flexible in the sense that they should be both reactive and proactive. Reactivity implies that agents must take timely actions in response to changes in the environment. Proactivity indicates that agents not only react, but also exhibit goal-oriented behavior.

The autonomy of agents implies that they are created with an autoprocess (or thread). The beliefs of an agent may be viewed akin to object state. However, in a general agent-oriented system, it is possible to share beliefs among agents (via a knowledge-base or a blackboard), which is contrary to the object encapsulation philosophy. Agents have desires (or goals), including proactive goals, requiring autonomous actions without any explicit method invocations. *The behavior of an agent (or the outcome of executing a method) may be different at different times (and may be non-deterministic)—since the proactive goals* *of an agent may lead to changes in its reactive behavior.* The different behaviors may result from the current beliefs and goals of an agent. An agent may not respond to method invocations by other agents (or objects). Such refusal behavior is not inherent in object-oriented systems. Agent-oriented systems do permit inheritance, but such inheritance must be distinguished from inheritance in object-oriented systems. Agents may inherit plans (or actions, which are similar to methods), beliefs (which are similar to instance variables) or goals (for which there is no direct counter part in OO).

In general beliefs may be shared and modified by other agents. This can be achieved either by direct communication (using Knowledge Query Manipulation Language KQML [4] messages), using shared knowledge bases or blackboards (for example using Linda or extensions to Linda such as LIME [7], [9]). Plans can be proactive or reactive -- proactive plans reflect the desires or goals of an agent. These goals may impact how an agent reacts to external events (including the possibility of ignoring external stimuli). Reactive plans reflect how an agent can be situated in an environment.

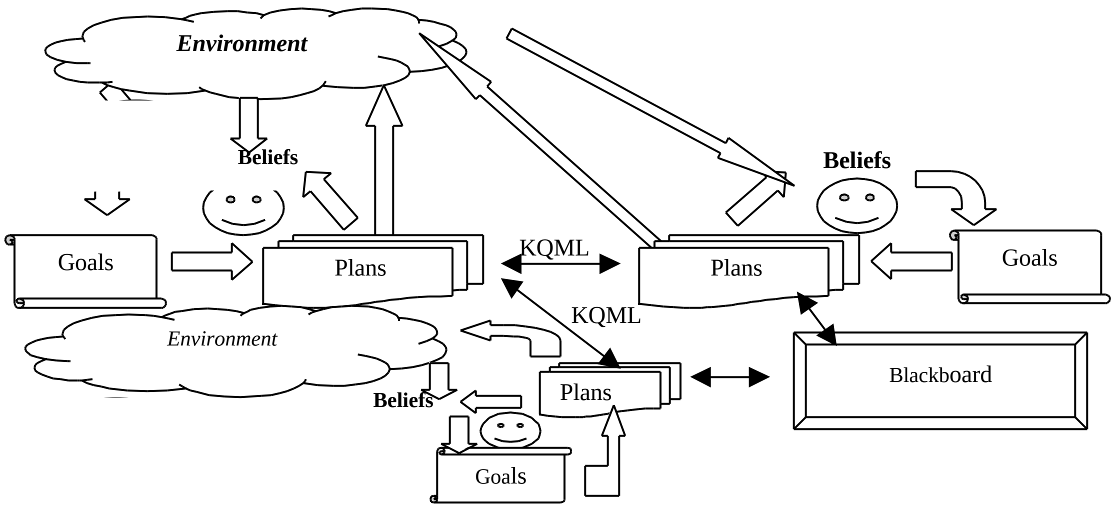

Figure 1. A conceptual model for a multi-agent system

1.2. BDI Formalism[1].

Given the above overview of agents, we will briefly summarize a related formalism that has been widely accepted in the AI community. The BDI architecture associates with agents, beliefs (typically about the environment and other agents), desires or goals to achieve, and intentions or plans to act upon to achieve its desires. Although sound formalism supports the BDI, in our research we are interested in practical modeling, design and implementation of agent-oriented systems using the BDI. In practical terms, beliefs can be viewed as the state of the world (at least the state as viewed by agents, even if this information is inaccurate or outdated). Beliefs may be represented as simple variables and data structures or, complex systems such as knowledge bases. Desires (or goals) may be associated with a value so that desires can be prioritized. Intentions reflect the actions that must be exercised to achieve the goal values. In our view, the BDI formalism may be used to model intelligent, autonomous, situated agents as shown in Figure1

[1] For a more detailed and formal treatment the reader is referred to any of the numerous publications on BDI (for example, [11-13]).

2. A Framework For Agent-Oriented Software Engineering

In this section we will outline our initial ideas on the specification, modeling, analysis and implementation of Agent-Oriented Software systems.

2.1. Agent Specification Language.

Utilizing the BDI framework, we propose the following language structure[2] to describe agent-oriented systems. We use an informal BNF like notation here.

```
Agent ::= <agent-name> "{" <Belief_Specification>
                <Goal_Specification>  <Plan_Speceification>
                protected <active-process> "}" /* see note 8 below */
<Belief_Specification> ::= "{" public   <Beliefs_Structure>*
                                private    <Beliefs_Structure>* "}"
<Goal_Specification> ::=  "{"   public   <Goals_Structure>*
                                private   <Goals_Structure>*"}"
```

[2] There have been others who have either used the term agent-oriented programming (AOP) [14], or defined languages for specifying agent behavior [16]. Our intention in the project is not define a completely new programming language, but to define a structure that can be used to extend existing languages such as Java. Our goal is to develop appropriate preprocessing tools so that the agent specifications can be converted into Java programs.

```
<Plan_Specification> ::=   "{"  public    <Plans_Structure>*
                                 private   <Plans_Structure>*"}"
<Beliefs_Strucure> ::= <Belief_Body> <Beliefs_Structure> |
            NULL
 <Belief_Body>::= <Belief_Name> <Belief_Spec>
                        /* notes 1 and 2 */
           <$annotation_name>=<string_value>*<Goals_Effected>
                        /* notes 3 and 4 */
<Belief_Spec>::= <Variable_Dec> | <Belief_Eval_Function> /*
                        note 2
<Goal_Effected>::= <Goal_Name>*
<Goals_Structure> ::= <Goal_Body> <Goals_Structure> | NULL
<Goal_Body> ::= <Goal_Name> <Goal_Spec> <Goal_Value>
                     <Plans_To_Execute> /* notes 5 and 6 */
<Goal_Spec> ::= <Goal_Decl> <Goal_Eval_Function>
<Plans_Structure> ::= <Plan_Body> <Plans_Structure> | NULL
<Plan_Body> ::= <Plan_Name> <Invoke_Trigger>
            <Context_Preconditions><Sequence_Of_Statements>
<Sequence_Of_Statements> ::= <Statement>
                     <Sequence_Of_ Statements>  | NULL
<Statement> ::= <Simple_Statement> | <Compund_Satement>
                     /* note 7 */
```

Notes:

1. Belief_Name is for identification and indexing purposes. Same with Goal_Name and Plan_Name.
2. Belief_Spec is specific to a belief and defines the necessary data structure to store the values of observations. These structures together can be viewed as a description of the state of the world. This structure can be a simple variable, data structure, a database or a knowledge base. The specification can provide a means of obtaining the value for the belief using the evaluation function (<Belief_Eval_Function>).
3. Annotations can be specified with beliefs for the purpose of analysis of the specification. The actual annotation name and value is domain specific. For example, for a real-time system, an annotation can be "Sampling_Frequency" and the specified value describes how often a sensor value should be sampled. Another example of annotation can be "Probability_of_Change" and the value indicates the likelihood of a change of a belief value. Such annotations can be used for feasibility or correctness analyses of the specification. The example annotations for real-time systems can be used for schedulability analyses. The annotations can also be used by runtime systems or middleware to schedule agent plans.
4. Goals_Effected links the goals that must be updated when the value of a belief changes.
5. Goal_Spec is specific to the goal and includes an evaluation function that can be used to update the value of the goal. Goal_Value is typically a real number indicating how valuable this goal is to the overall goals of an agent (or system). If a goal is not achievable, the value will be either zero or negative. It is necessary to select values to permit some selection and prioritization among goals. Goal Specification can be similar to a decision tree; based on the current state of the observed values, the decision tree is traversed to obtain a goal value. The path through the decision tree can also be viewed as the sequence of plans needed to achieve goal value. New goal valuations may lead to abortion of previously scheduled plans.
6. Plans_To_Extecute links to one or a sequence of plans that must be executed to achieve the goal (and its associated value). Both reactive and proactive goals can be defined using our syntax: reactive goals often receive higher goal values than proactive such that reactive goals will be executed in a timely manner The above syntax implies that goals are triggered when belief values change. The changes can be either due to external triggers, messages from other agents or changes to a knowledge base.
7. Compound Statements for describing a plan can include any programming language statement. It is straightforward to include KQML [4] messages in our language structure, since a <Compound Statement> of a plan can be a KQML statement. KQML represents communication among agents. The communication is typically for one of the following purposes (we can map these to KQML performatives).

Assert a Belief value. This could be either because of a previous request, a routine sensing of the state of the world, or other reasons. This message will have the same effect as if the agent is sampling the state of the world and setting its belief values; which in turn may require re-computation of goal values.

Ask for a Belief value. An agent who does not have the responsibility of sampling for a particular state value can request another agent for the value. The reply will contain the value or the reply can take the form of "Assert a Belief Value" message.

Ask for a service. An agent may execute a plan in response to this or ignore the request.

8. Autoprocess. An agent can be designed with one or more threads to meet the scheduling requirements. If a single thread is used the structure of autoprocess may look like (using ADA like choice)

```
autoprocess() {
  do {
      Select
         (If_Reactive_Message_Queue_Not_Empty)
            Perform_Reactive_Plan
     or
      Select   (If_Belief_Sampling_Is_Due)
            Sample_External_State();
            Update_Goal_Values()
            Select_A_Plan_To_
               Achieve_Optimal_Goal_Value();
    or  Proactive_Goal_Plan();}}
```

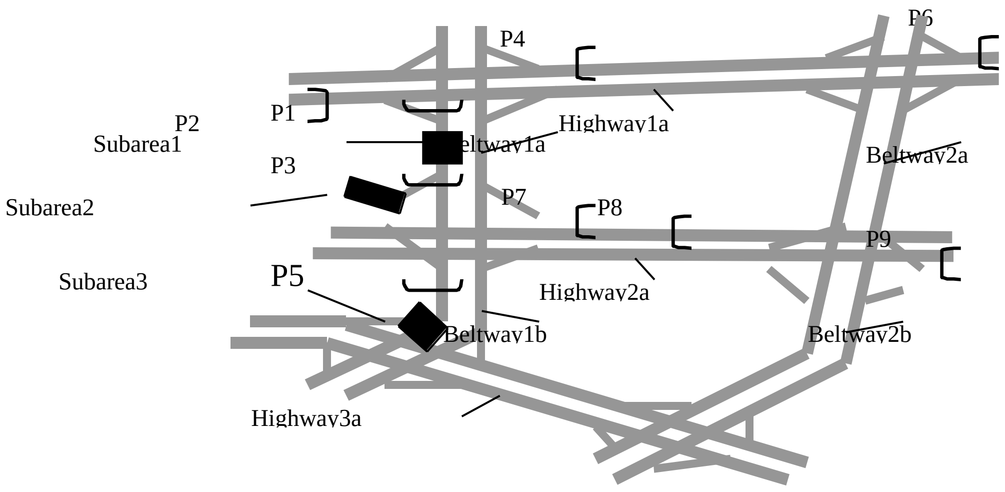

Figure 2. Traffic Control Example

If multiple threads can be utilized, we can associate separate threads to respond to belief changes, external stimuli, external service requests and proactive goals. Internal mechanisms are needed to assure proper coordination and synchronization of concurrent threads (e.g., using mutual exclusion).

2.2. An Example.

In order to illustrate the use of our framework for the design of agent-oriented software systems, we describe a portion of a complex real-time system using our specification language (for a complete description of the example as well as other examples see [1]). Here we describe the use of a multi-agent based system for controlling traffic flow along several major highways (or beltways). The agents are responsible for detecting traffic flows (using sensors to detect the number and rate of flow of cars) and controlling traffic signals as well as posting warning messages as well as suggested alternate routes on Variable Message Panels (VMP). Figure 2 shows a block diagram of the system. Here each agent is responsible for a section of a highway and they are identified as Beltway1a, Beltway 1b, etc., P1, P2,.. are VMPs. Here we outline the agent using our Agent Specification Language.

```
Beliefs
    Public:
        Belief_1 = (Belief_1, Sensor_1(Working),
            sampling_frequency =hour,
            probability_of_chage  =0.001, Goal_2)
        Belief_2 = (Belief_2, Circulation_Regime
            (Beltway_1b, "Congested"),
             sampling_frequency =hour,
             probability_of_chage=.0.5, Goal_1)
        Belief_3 = (Belief_3,Saturation_Level
             (Beltway_1b, Critical"),
             sampling_frequency =hour,
             probability_of_chage=.0.5, Goal_1)
        Belief_4 = (Belief_4, Rush_Hour(YES,
             Lunch), sampling_frequency =hour,
              probability_of_chage=.0.2, Goal_3)

    Private:
        Belief_5 = (Belief_5, Traffic_Flow
             (Beltway_1a,120), sampling_frequency =1
             minute, probability_of_chage=.0.5, Goal_3)
         Belief_6 = (Belief_6, Sensor(Beltway_1,
             Sensor_1), NULL)
         Belief_7 = (Belief_7, VMP_1(" Slow Traffic
              Ahead"), sampling_frequency =15
              minutes,  probability_of_chage=.0.5,
              NULL)
         Belief_8 = (Belief_8, Traffic_Light_1
              (Green_Period = 3 min, Red_Period = 1
               min, Yellow_Period = .05 min), Goal_3)
Goals
    Public:
        Goal_1 = (Goal_1, Smooth_Traffic
             (Beltway_1), 10,  Identify_Problem,
             Diagnose, Configure_Singal_Plan)
       Goal_2 = (Goal_2, Fix(Sensor_1), 20,
            (Inform_Maintenance, Write_To_VMP))
    Private:
        Goal_3 = (Goal_3, Traffic_Flow
             (Beltway_1a) < 80, cars/min, 15,
             Emergency, Rush_Hour, Congestion,
             Message_Consistency)
Plans
    Public:
        Plan_1 = (Inform_Maintenance {
                 Invoke on Fix(Sensor_1)
                 With Context Sensor_1(Not Working)
                  Do
                    (tell:sender Beltway_1a  :language
                     prolog  :content Sensor_1(NOT
                    Working) :receiver  Maintenance
                     Person :reply-with xyz ))
    Private
      Plan_2  = (Configure_Signal_Plan{
```

```
                    Invoke on Traffic_Flow (Beltway_1a)
                     < 80 cars/min With context …
                  Do
                    (tell :sender Beltway_1a  :language
                    prolog :content Congestion at Subarea
                    3 :receiver VMP_1 :reply-with abc) )
   Private
        Plan_3 = ( Rush_Hour{ Invoke on Traffic_Flow
                    (Beltway_1a) < 80 cars/min
                    With context Rush_Hour(YES, Lunch)
                 Do
                    SET Traffic_Light_1(Green_Period = 2
                    min,  Red_Period = 2 min,
                    Yellow_Period = .05 min )

Autoprocess() {
       Do {
       Select (If_Reactive_Message_Queue_Not_Empty)
                    Perform_Reactive_Plan
      Or
      Select(If_Belief_Sampling_Is_Due)
             Sample_External_State(); Update_Goal_Values();
             Select_A_Plan_To_Achieve_Optimal_Goal_Val;
      Or
      Select
              Proacative_Goal_Plan();
            } }
```

3. Summary and Conclusions

In this paper we outlined our preliminary ideas on a framework for the specification, analysis and design of agent oriented software systems. Our approach is based on BDI – however, we use the formalism for developing a practical software engineering framework. We have also developed extension to UML but due to space limitations we have not included them in this paper.

4. References

[1] M. Aborizka. An Architectural Framework for the Specification, Analysis and Design of Intelligent Real-Time Monitoring Agent Based Software Systems, *PhD Dissertation* (in Preparation), Dept of ECE, University of Alabama in Huntsville.

[2] B. Bauer, J. P. Muller and J. Odell. "Agent UML: A formalism for specifying multi-agent software systems", *First International Workshop on Agent Oriented Software Engineering*, Limerick, Ireland, June 2000, pp 91-104.

[3] G. Booch. *Object-Oriented Analysis and Design*, 2nd edition, Addison-Wesley, Reading, MA.

[4] T. Finn, Y. Labrou and J.Mayfield. "KQML as an agent communication language", in *Software Agents*, edited by J. Bradshaw, MIT Press, Cambridge, 1977.

[5] N. R. Jennings, K. Sycara and M. Wooldridge. "A roadmap of agent research and development", in *Autonomous Agents and Multi-Agent Systems*, Kluwer Academic Publishers.

[6] B.Meyer. *Object-oriented software construction*, 2nd edition, Prentice-Hall, Englewood Cliffs.

[7] A. Murphy, G. Picco and G.-C. Roman. "LIME" A middleware for physical and logical mobility", Proceeding of the 21st International Conference on Distributed Computing Systems (ICDCS), April, 2001, pp 524-533.

[8] J. Odell, H. Van Dyke Parunak and B. Bauer. "Representing Agent Interaction Protocols in UML", *First International Workshop on Agent Oriented Software Engineering*, Limerick, Ireland, June 2000, pp 121-140.

[9] G. Picco, A. Muphy and G.-C.Roman. "LIME: Linda meets mobility", *Proceedings of the 21st International Conference on Software Engineering*, May 1999.

[10] J. Rumbaugh, et. al. *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

[11] A. Rao and M. Georgeff. Modeling rational agents within a BDI architecture. *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, MA, 1991, pp 473-484.

[12] A. Rao, M. Georgeff. An Abstract Architecture for Rational Agents. *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, Boston, MA, 1992, pp 439-449.

[13] A. S. Rao and M.P. Georgeff. "BDI agents; From theory to practice", *Proceedings of the first international conference on multi-agent systems (ICMAS-95)*, San Francisco, pp 312-319.

[14] Y. Shoham. "Agent-Oriented programming", *Artificial Intelligence*, (Vol 60), pp 51-92.

[15] A. Tveit. "A survey of agent-oriented software engineering", First CS Graduate Students Conference, http://www.csgsc.org

[16] G.Wagner. "Agent-Oriented-Relationship Modeling", Proceedings of 2nd Internaitonal Symposium –From Agent Theory to Agent Implementations, in connection swith EMCRS 2000, April

[17] M.F. Wood M. F. and S. A. DeLoach, ``An Overview of the Multiagent Systems Engineering Methodology," *The First International Workshop on Agent-Oriented Software Engineering* (AOSE-2000), 2000.

Acknowledgement. This research is supported in part by NSF Digital Government Grant, EIA 0087076.