An AI Architecture Based on Message Passing

Deepak Kumar
Department of Computer Science
State University of New York at Buffalo, Buffalo, NY 14260
kumard@cs.buffalo.edu

Abstract

We present an AI architecture used for modeling computational rational cognitive agents— agents that can reason, and act in an environment based on their representations of beliefs, acts, and intentions. A unified representational formalism is employed for representation of propositions (beliefs), goals (desires), actions, and plans. The representational formalism is described using a conceptual object-oriented hierarchy. An object-oriented design facilitates a uniform method (message) protocol that is naturally amenable to concurrency. The reasoning and acting behavior of the modeled agent is defined by an actor-like [Agha 1986, Agha and Hewitt 1987] message passing scheme. Semantically speaking, messages can be viewed as meta-level propositional attitudes (corresponding to 'belief' and 'intention') that are sent to/from propositions and acts in parallel. The message passing model, called a *Rational Engine* (as opposed to an inference engine), implements a BDI-architecture[1].

1 Motivations

A survey of AI systems would reveal that it is somewhat awkward to do acting in reasoning (or logic-based) systems (but it is convenient to talk about representational and reasoning issues using them), and it is awkward to study reasoning and representational issues in systems designed for acting/planning. Thus, most "good" planning/acting systems are "bad" KRR systems and vice versa. For example, in a recent symposium on "Implemented KRR Systems" [Rich 1991b] out of a total of 22 KRR systems presented only 4 systems had capabilities for representation and reasoning about actions/plans (RHET [Allen 1991], CYC [Lenat and Guha 1991], CAKE [Rich 1991a] and SNePS [Shapiro 1991]). The work presented in this paper presents an approach that bridges this "representational/ behavioral gap." We extend the ontology of an intensional KRR system to be able to represent and reason about acts and plans. I.e. a computational cognitive agent modeled using the extended ontology has representations for beliefs, acts, and plans, and is able to reason about them. I.e., the agent is able to represent its beliefs and desires (the 'B' and the 'D' of 'BDI').

In most current AI architectures reasoning is performed by an inference engine and acting is done under the control of some acting executive (or a plan/act interpreter). Our approach is based on the viewpoint that logical reasoning rules implicitly specify the act of believing. Thus the inference engine can be viewed as a *mental actor*. This enables us to establish a closer relationship between rules of inference and rules of acting (or planning). Believing is a state of knowledge; acting is the process of changing one state into another. A reasoning rule can be viewed as a rule specifying an act—that of believing some previously non-believed proposition—but the believe action is already included in the semantics of the propositional connective. McCarthy (1986) also suggested that inference can be treated as a mental action. This has lead us to integrate our models of inference and acting by eliminating the acting executive (plan/act interpreter). These ideas are used in developing a computational model— called a *Rational Engine*, that is a unified model of acting and inference and can be used for modeling rational cognitive agents and their behavior. Acting and reasoning about beliefs, actions, and plans is performed by a single component— the Rational Engine. The rational engine implements the underlying logic as well as notions of intentionality (the 'I' of 'BDI').

[1] for Beliefs, Desires, and Intentions

2 An Object-Oriented Representational Formalism

The representational formalism is described as a conceptual object-oriented hierarchy. This is depicted in Figure 1. In an intensional representational framework, anything a cognitive agent can think about is termed a "mental concept" or a conceptual entity. More specifically these can be individuals, beliefs (propositions), or acts. In addition to standard beliefs that an agent is able to represent, we also define a special class of beliefs called *transformers*. A *transformer* is a propositional representation that accounts for various notions of inference and acting. A transformer is a representation that specifies a belief/act transformation. It has two parts—($\langle a \rangle$, $\langle b \rangle$), where both $\langle a \rangle$ and $\langle b \rangle$ can specify either beliefs or some act. Transformations can be applied in forward and/or backward chaining fashion. Using a transformer in forward chaining is equivalent to the interpretation "after the agent believes (or intends to perform) $\langle a \rangle$, it believes (or intends to perform) $\langle b \rangle$." The backward chaining interpretation of a transformer is, "if the agent wants to believe (or perform) $\langle b \rangle$, it must first believe (or perform $\langle a \rangle$." Since both $\langle a \rangle$ and $\langle b \rangle$ can be sets of beliefs or an act, we have four types of transformers— *belief-belief, belief-act, act-belief,* and *act-act.* Detailed descriptions of transformers and beliefs can be found in [Kumar 1989, Kumar and Shapiro 1991a, Kumar and Shapiro 1991b, Kumar 1993]. Specific transformers are depicted in Figure 1 (arrows against transformers indicate the directions— forward, backward chaining, they can be used in). Notice that transformers capture the notions of reasoning rules, act preconditions, effects, plan and goal decompositions, and reactivity.

Our present model of acting is based upon a state-change model (see [Kumar and Shapiro 1991a]). We identify three types of states —external world states, mental states (belief space), and intentional states (agent's current intentions). Accordingly, we identify three classes of actions —physical actions, mental actions, and control actions that bring about

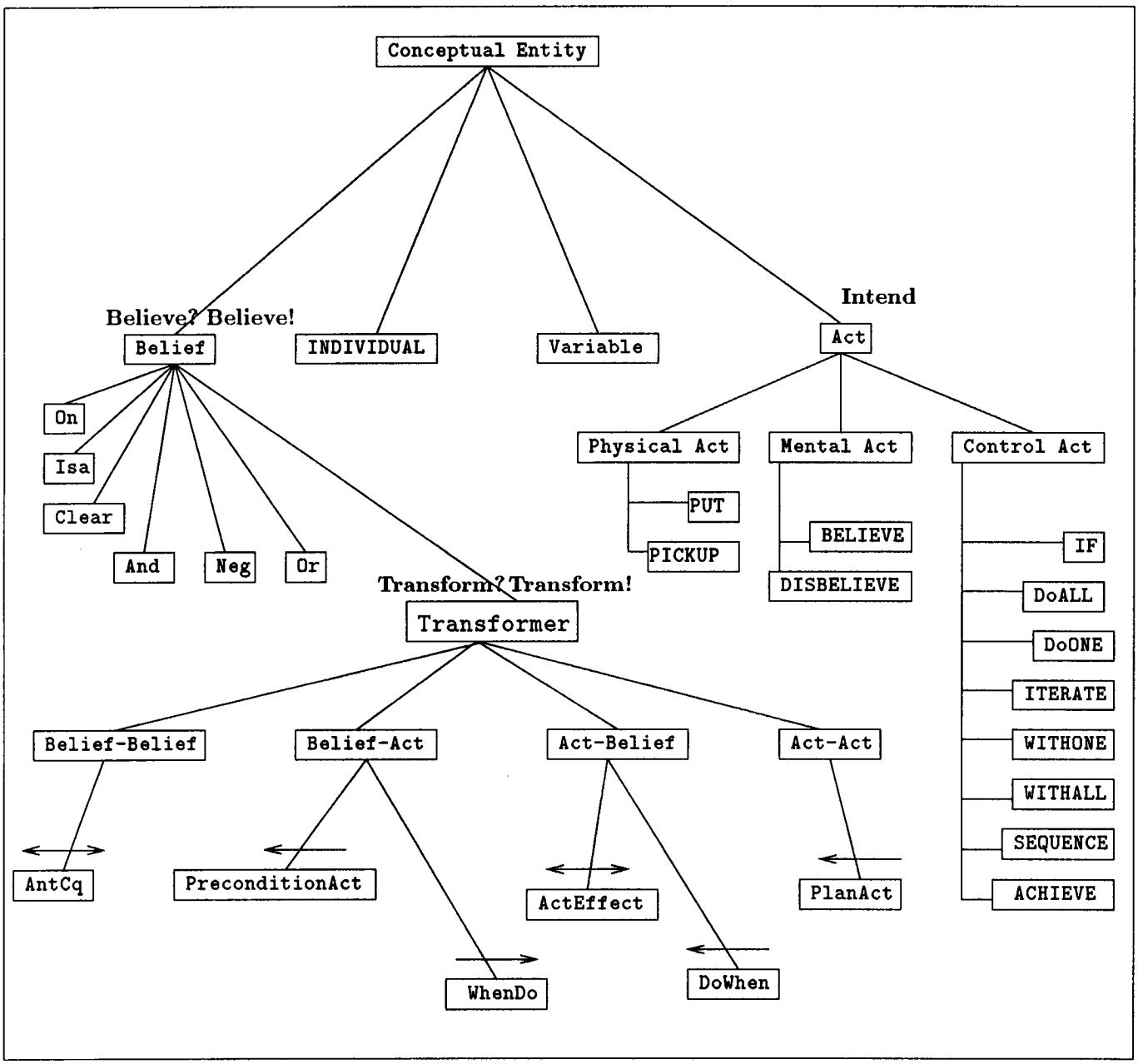

Figure 1: A Conceptual Object-Oriented KR Hierarchy

changes in their respective states. Thus PICKUP is a physical action, we have BELIEVE and DISBELIEVE as mental actions whose objects are beliefs, and control actions are described below. Physical acts can be *primitive* or *complex* (not shown in the figure). A primitive act has an effectory procedural component which is executed when the act is performed. Complex acts have to be decomposed into plans. A *plan* is a structure of acts. The structuring syntax for plans is described in terms of control actions. Our repertoire of control actions includes *sequencing, conditional, iterative, disjunctive, conjunctive,* and *selective* acts (once again, see [Kumar and Shapiro 1991b, Kumar 1993] for details).

As defined above, the representational formalism is isomorphic to the SNePS [Shapiro and Rapaport 1987, Shapiro and Group 1989] formalism. As such the representational formalism is inherently independent of any logical (or teleological) commitments. This is an advantage of object-oriented design. The formalism can be easily extended (by defining additional representational entities as well as methods) to accommodate any reasoning or acting theory. Reasoning and acting is accomplished by three methods— *Believe?, Believe!,* and *Intend.* The first two implement backward and forward chaining, respectively; and the third implements the modeled agent's intentions to perform specific acts. All the methods are inherited down the hierarchy (and may or may not be specialized further). The main aspect of this architecture is that it yields a uniform protocol for reasoning and acting that makes it amenable for concurrency. In the absence of a concurrent object-oriented system we have been experimenting with a parallel message passing implementation which is described next.

3 The Message Passing Model

We now present a computational model of the rational engine —the component that uses/interprets the representations resulting in the agent's reasoning and acting behavior. The rational engine employs a parallel message passing model that accounts for various notions of acting and inference. Entities pass messages to each other which in a SNePS-like formalism represents a kind of spreading of activation. However, unlike traditional notions, spreading of activation is governed by the underlying logic and the nature of entities.

3.1 Channels

Communication (or message passing) between entities takes place along *channels.* An entity can send and receive messages only via channels. We define two types of channels that may exist between any two entities:

Match channels: the two entities unify.

Transformer channels: the two entities are connected to each other by a transformer.

Only the proposition, act, and transformer objects are capable of processing messages. Channels may also exist between the user and the agent. These are set up at the time the user issues a request or a query or when something has to be reported to the user. Processing a message involves some book keeping and sending out some messages. The nature of book keeping and outgoing messages depends on the type of the entity, the type and content of the incoming message, and the entity's agenda. These are summarized in Tables 1, 2, and 3.

3.2 Messages

There are three types of messages—

Believe(p)? where p is a proposition (belief). Also called a *request,* it denotes that the sender is asking (requesting) the receiver about the assertional status of p.

Believe(p)! where p is a proposition and the message indicates that the sender is confirming the assertional status of p (i.e. the agent, by being informed, or via inference, now believes p). These messages are also called *reports.*

Intend(a) where a is an act and the message indicates that in the current state the act a is being intended to be performed.

In addition to the descriptions above, each message may also contain additional information pertaining to variable bindings, quantifier specifications, and some TMS related stuff. Processing of a message is determined by the type of message (request, report, or intend), the type of the entity (proposition, act, or rule), and the entity's current agenda. This results in the various notions of inference and acting (forward chaining, backward chaining, deductive retrieval of preconditions, effects, plan decompositions, and reactivity).

3.3 Agendas

The agenda of an entity determines its role in the acting and inference process. Different types of entities have different agendas. Each entity, after receiving a message, depending on its current agenda, may perform some book keeping actions and respond by sending out one or more messages. Since only propositions, acts, and transformers partake in message passing we have agendas defined for each type of entity along with its message handling behavior (See Tables 1, 2, 3). Due to space limitations it will not be possible to present a detailed example. Please see [Kumar 1993] for more details and examples. Here, we would like to present an actor-based view of the model.

4 ACTOR Systems

Like actor systems [Agha 1986, Agha and Hewitt 1987] the Rational Engine model fully exploits message-passing as the basis of concurrent computation. The model is implemented using MULTI— A Lisp-based multi-processing system [McKay and Shapiro 1980a, McKay and Shapiro 1980b] with a sequential implementation. Serialization only affects message processing events— one message is processed at a time. Actors in MULTI are called *processes* which basically respond to incoming messages. Thus message passing is the only primitive available in the system. Channels, described above, help each process determine *mail addresses* of outgoing messages. Return addresses, as in actor systems, are included in the message. Messages are propagated during reasoning and acting to pursue all possible branches in parallel. Thus, during inference and acting, this is akin to a focused spreading of activation. Changing the agenda of an entity is similar to an actor specifying a replacement behavior in actor systems. In an AI architecture, there are a small number of different types of entities and thus there are only a limited types of different messages. Moreover, most entities of the same type have a similar message processing "script".

	Incoming Message	Agenda	Response
1	*Believe*(p)?	ASSERTED	Send message *Believe*(p)! to s.
2	*Believe*(p)?	UNASSERTED	Send message *Believe*(p)? to all match channels and all belief-belief (Cq) channels (standard backward chaining) and all act-belief transformer channels (i.e. DoWhen) if any.
3	*Believe*(m)!	any	If $p = m$ then update its agenda to ASSERTED. Send *Believe*(m)! to all requesters, all match channels, all belief-belief (Ant) transformer channels. (standard forward chaining) and all belief-act transformer channels (i.e. WhenDo) if any.

Table 1: Message processing by beliefs (p is the receiver, and s the sender)

	Incoming Message	Agenda	Response
1	*Believe*(p)?	any	Sent by a node (p) in the consequent position (over a Cq transformer channel). Change the agenda to ACTIVE. Send a request *Believe*(Ai)? to all the antecedent nodes (Ais) over transformer channels. (standard backward chaining)
2	*Believe*(m)!	NONE	Antecedents reporting some belief. Change the agenda to ACTIVE. Send a *Believe*(Ai)? to all the remaining antecedents so as to confirm believing its consequences (starting a forward inference)
3	*Believe*(m)!	ACTIVE	Antecedents answering requests. If firing criteria is satisfied send a *Believe*(Ci)! message to all the consequent Cis and change the agenda to NONE.

Table 2: Message processing by belief-belief transformers.

	Incoming Message	Agenda	Response
1	*Intend*(a)	START	Change agenda to FIND-PRECONDITIONS. Send request *Believe*(PreconditionAct(p, a))?
2	*Intend*(a)	FIND-PRECONDITIONS	Change agenda to FIND-EFFECTS. Send request *Believe*(ActEffect(a, p))?
3	*Intend*(a)	TEST-PRECONDITIONS	Change agenda to START. Send message *Intend*(d) to the act (d) of achieving all the preconditions of a.
4	*Intend*(a)	FIND-EFFECTS	Change agenda to EXECUTE. Send message *Believe*(Primitive(a))?
5	*Intend*(a)	EXECUTE	Change agenda to FIND-PLANS. Send request *Believe*(PlanAct(a, p))?
6	*Intend*(a)	FIND-PLANS	No plan decompositions for a are known. Perform classical planning (not implemented).
7	*Believe*(m)!	FIND-PRECONDITIONS	m is a PreconditionAct(a, p) proposition. Change agenda to TEST-PRECONDITIONS. Send message *Believe*(p)?
8	*Believe*(m)!	TEST-PRECONDITIONS	Some precondition (m) of a is satisfied. If all preconditions are satisfied, change agenda to FIND-EFFECTS. Send message *Believe*(ActEffect(a, p))?
9	*Believe*(m)!	FIND-EFFECTS	(m) is an ActEffect(a, e) proposition. Change agenda to EXECUTE. Send message *Believe*(Primitive(a))?
10	*Believe*(m)!	EXECUTE	The act (a) is primitive. Execute its effector component. Send *Intend*(d) to d, the act of believing a's effects.
11	*Believe*(m)!	FIND-PLANS	m is a PlanAct(a, p) proposition. Change the agenda to DONE. Send message *Intend*(d) to d the act of doing one of the plans (p).

Table 3: Message passing by acts

This implies that behavior can be inherited. This, unlike actor systems, is a desireable trait of such an architecture— best of both worlds. This implies that in some actor models, it is possible to have inheritance. This also implies that such a representational and behavioral view could be easily programmed using an actor language, thus, fulfilling one of the long term goals of actor systems— building actor-based AI architectures. Notice that the actor-view of an AI architecture presented here arises out of considerations involved in making some representational as well as some behavioral commitments. Only the latter being similar (or conforming) to the actor view of computation. Details of different types of transformers are not presented here but one important consideration that transformers help capture is the overall embedded nature of the architecture. The belief-act transformers `WhenDO` and the backward chaining act-belief transformer `DoWhen` model reactivity of the agent and the agent's ability to perform actions in order believe something. These, too, form desireable traits of actor systems (in lieu of open systems).

5 Concluding Remarks

We have outlined an object-oriented design of a BDI architecture. The conceptual object-oriented hierarchy helps identify specific methods required to implement reasoning and acting processes. In order to facilitate concurrent processing we view methods as parallel messages rather than function invocations. Parallelism is easily facilitated by the message passing model. Message passing entities in such an architecture can be construed as actors in the ACTOR formalism. Thus, while we started out by designing an architecture as an object-oriented system, we ended up with a computational model very similar to that of the actor formalism. Other than Carl Hewitt's original motivations that lead to the evolution of ACTOR systems not much work has been done to explore the actor paradigm for implementing intelligent architectures. We hope that the architecture presented here serves as a candidate. We are willing to pursue further research in this direction.

References

[Agha and Hewitt 1987] Gul Agha and Carl Hewitt. Concurrent programming using actors. In Akinori Yonezawa and Mario Tokoro, editors, *Object-Oriented Concurrent Programming*, pages 37–53. the MIT Press, 1987.

[Agha 1986] Gul Agha. An overview of actor laguages. *ACM SIGPLAN Notices*, 21(10):58–67, October 1986.

[Allen 1991] James Allen. The RHET System. In Charles Rich (Guest Editor), editor, *SIGART BULLETIN Special Issue on Implemented KRR Systems*, pages 1–7, June 1991.

[Kumar and Shapiro 1991a] Deepak Kumar and Stuart C. Shapiro. Architecture of an intelligent agent in SNePS. *SIGART Bulletin*, 2(4):89–92, August 1991.

[Kumar and Shapiro 1991b] Deepak Kumar and Stuart C. Shapiro. Modeling a rational cognitive agent in SNePS. In P. Barahona, L. Moniz Pereira, and A. Porto, editors, *EPIA 91: 5th Portugese Conference on Artificial Intelligence, Lecture Notes in Artificial Intelligence 541*, pages 120–134. Springer-Verlag, Heidelberg, 1991.

[Kumar 1989] D. Kumar. An integrated model of acting and inference. In D. Kumar, editor, *Current Trends in SNePS–Semantic Network Processing System: Proceedings of the First Annual SNePS Workshop*, pages 55–65, Buffalo, NY, 1989. Springer-Verlag.

[Kumar 1993] Deepak Kumar. A Unified Model of Acting and Inference. In *Proceedings of the Twenty-Sixth Hawaii International Conference on System Sciences.* IEEE Computer Society Press, Los Alamitos, CA, 1993.

[Lenat and Guha 1991]
Douglas B. Lenat and Ramanathan V. Guha. The Evolution of CYCL, The CYC Representation Language. In Charles Rich (Guest Editor), editor, *SIGART BULLETIN Special Issue on Implemented KRR Systems*, pages 84–87, June 1991.

[McCarthy 1986] John McCarthy. Mental situation calculus. In Joseph Y. Halpern, editor, *Theoretical Aspects of Reasoning about Knowledge—Proceedings of the 1986 Conference*, page 307, 1986.

[McKay and Shapiro 1980a] * D. P. McKay and S. C. Shapiro. MULTI: A LISP based multiprocessing system. Technical Report 164, Department of Computer Science, SUNY at Buffalo, 1980. (Contains appendices not in item 15).

[McKay and Shapiro 1980b] D. P. McKay and S. C. Shapiro. MULTI — a LISP based multiprocessing system. In *Proceedings of the 1980 LISP Conference*, pages 29–37. Stanford University, Stanford, CA, 1980.

[Rich 1991a] Charles Rich. CAKE: An Implemented Hybrid KR and Limited Reasoning System. In Charles Rich (Guest Editor), editor, *SIGART BULLETIN Special Issue on Implemented KRR Systems*, pages 120–127, June 1991.

[Rich 1991b] Charles Rich. Special Issue on Implemented Knowledge Representation and Reasoning Systems— Letter from the Guest Editor. *SIGART Bulletin*, 2(3), June 1991.

[Shapiro and Group 1989] S. C. Shapiro and The SNePS Implementation Group. *SNePS-2 User's Manual.* Department of Computer Science, SUNY at Buffalo, 1989.

[Shapiro and Rapaport 1987] S. C. Shapiro and W. J. Rapaport. SNePS considered as a fully intensional propositional semantic network. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier*, pages 263–315. Springer-Verlag, New York, 1987.

[Shapiro 1991] Stuart C. Shapiro. Case studies of SNePS. *SIGART Bulletin*, 2(3):128–134, June 1991.