1996-1

Reprinted from

Decision Support Systems
The International Journal

Decision Support Systems 16 (1996) 3–19

The SNePS BDI architecture

Deepak Kumar *

Department of Mathematics and Computer Science, Bryn Mawr College, Bryn Mawr, PA 19010, USA

N·H
ELSEVIER

ELSEVIER

Decision Support Systems 16 (1996) 3–19

Decision Support Systems

The SNePS BDI architecture

Deepak Kumar *

Department of Mathematics and Computer Science, Bryn Mawr College, Bryn Mawr, PA 19010, USA

Abstract

Most AI systems are effective either for inference or for acting/planning but not for both. The SNePS BDI architecture uses propositional semantic network representations of beliefs, goals, acts, plans, and a representational concept called *transformers*. Transformers help capture a unified approach to acting and inference. They can be used to represent reasoning rules, reactive desires, desires for acquiring knowledge, preconditions and effects of actions as well as plan decompositions. A *rational engine* module operates on these representations and is responsible for the agent's reasoning and acting behavior. SNeRE, the SNePS rational engine, employs a quasi-concurrent message passing scheme, implements the underlying logic as well as the action theory, has forward, backward and bidirectional inference and acting. It also incorporates the notions of truth maintenance and spreading activation.

Keywords: Knowledge representation and reasoning; Acting; Planning; Semantic networks; BDI architectures; Logic modeling

1. Introduction

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" knowledge representation and reasoning systems and vice versa. For example, in a recent symposium on "Implemented Knowledge Representation and Reasoning Systems" [18] out of a total of 22 knowledge representation and reasoning systems presented only 4 systems had capabilities for representation and reasoning about actions/plans (RHET [1], CYC [13], CAKE [17] and SNePS [25]). The work presented in this paper presents an approach that bridges this "representational/behavioral gap". We extend the ontology of a knowledge representation and reasoning system to be able to represent and reason about acts and plans. A computational cognitive agent modeled using the extended ontology has representations for beliefs, acts and plans, and is able to reason about them. The modeled agent is able to represent beliefs and desires (the "B" and the "D" of "BDI"). We also describe a unified model of acting and inference that is based on our investigations of the relationships between beliefs, acts, plans, reasoning and

* Corresponding author. Email: dkumar@cc.brynmawr.edu

SSDI 0167-9236(94)00053-0

acting. The computational model, called a *Rational Engine*, is based upon spreading of activation and uses message passing to accomplish acting and inference. These ideas are implemented using SNePS (for *S*emantic *Ne*twork *P*rocessing *S*ystem) [24,22]: an intensional, propositional, semantic network system used for modeling cognitive agents. SNePS-based cognitive agents have network representations for individuals, propositions, deduction rules, actions, acts, and plans. Acting and reasoning about beliefs, actions, and plans, is performed by a single component, SNeRE: the *SNe*PS *R*ational *E*ngine, whose computational model is described here.

2. Motivations

In the past, most efforts of researchers of the SNePS research group have centered around representation and reasoning about beliefs derived from natural language interaction with the user. Our work extends the SNePS approach to modeling cognitive agents by integrating the notions of acting and planning (the "I" of "BDI"). The basic motivations underlying our approach can be summed by the following quote from Georgeff [3]:

"Another promising approach to providing the kind of high-level goal-directed reasoning capabilities, together with the reactivity required for survival in the real world, is to consider planning systems as rational agents that are endowed with the psychological attitudes of belief, desire, and intention. The problem that then arises is specifying the properties we expect of these attitudes, the way they interrelate, and the ways they determine rational behavior in a situated agent."

Architectures that enable the modeling of an agent's beliefs, desires, and intentions have come to be called BDI architectures. The SNePS BDI architecture presented here attempts to satisfy the above concerns using a unified approach to inference and acting. This involves employing a uniform representational formalism as well as a closer relationship between the processes of acting and inference. These are discussed next.

2.1. Uniform representations

Research in planning and acting has progressed independently of research in knowledge representation and reasoning. Traditional planners typically use three different levels of representations (each using a different representation language):

- a representation for world model (typically a FOPL);
- a representation for operators/actions (like the operator schema of STRIPS [2], or the operator description language: ODL of SIPE [28]); and
- a representation for plans (like NOAH's [19] and SIPE's procedural networks).

As a consequence, the system has to perform reasoning at three different levels:

- reasoning within the world model;
- reasoning about actions (used in the planning component); and
- reasoning about plans (as done by *procedural critics* of NOAH and SIPE).

Facts stored in the world model correspond to the agent's beliefs. Reasoning done on these beliefs is limited to basic retrieval. Sometimes, using simple inference rules (which may or may not be expressed in the same language, e.g., Wilkin's deductive operators [28]) simple consequences of current beliefs can be derived. The state of the art in knowledge representation and reasoning is much more advanced than that. Current knowledge representation and reasoning systems are capable of dealing with issues in natural language understanding, representing beliefs of the agent as well as others, belief revision using truth maintenance procedures, and other subtle issues. Some of these representations also deal with beliefs about agents performing actions and events taking place.

In the approach followed here, beliefs, goals, acts, plans, and rules, are represented in the same language: intensional, propositional semantic networks (SNePS). All representations are designed keeping in mind that at all times we are involved in modeling rational cognitive agents that are capable of natural language interaction,

representing and reasoning about their own beliefs as well as those of others, as well as act in the world in which they are situated.

2.2. The relationship between acting and inference

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. John McCarthy [15] has also suggested that inference can be treated as a mental action. This suggests that we can integrate our models of inference and acting by eliminating the acting executive (typically the module responsible for act/plan execution). 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 in SNePS is performed by a single component, SNeRE, the SNePS Rational Engine, whose computational model is described here. The computational model is based on a focused spreading activation and uses message passing to accomplish acting and inference.

3. Intensional representations

The modeled agent's beliefs, acts, plans, and rules are represented in the SNePS semantic network formalism. In the quote above, Georgeff mentions the importance of modeling rational agents by giving an intensional account of the notions of belief, desire, and intention. SNePS is an intensional propositional semantic network system [24]. Structurally, the semantic network is comprised of nodes and arcs. Nodes in the semantic network represent conceptual entities. A conceptual entity is anything a cognitive agent can think about. We explicitly identify the following types of conceptual entities:

Individuals. These are the named entities in the domain of discourse. For example, a node labelled John could be used to represent an individual named "John". Individuals form basic terms in the underlying logic.

Structured Individuals. These are nodes that have arcs coming out of them. Arcs represent structural links between the nodes (thus identifying a node with a set of arcs coming out of it as a specific structured individual). The labels on arcs determine the semantic interpretation of the entity being represented. Structured individuals also form terms in the underlying logic. As we will see below, structured individuals can be used to represent beliefs, acts, plans, as well as rules.

Variables. Variable nodes denote arbitrary conceptual entities. They can be used to represent generic individuals (e.g., a "thing"), general propositions (e.g., "a thing that is a block"); generic propositions (e.g., "something that John believes"); and generic acts (e.g., "picking up a thing"). Syntactically, variables can be bound (by a quantifier) or they can be free (i.e., unbound). We will see examples in later sections.

A basic principle of SNePS is the Uniqueness Principle: that there be a one-to-one mapping between nodes of the semantic network and concepts (mental objects) about which information may be stored in the network. These concepts are not limited to objects in the real world, but may be various ways of thinking about a single real world object, such as The Morning Star versus The Evening Star versus Venus. They may be abstract objects like properties, propositions, Truth, Beauty, fictional objects, and impossible objects.

3.1. Beliefs

The modeled agent's beliefs are represented as propositional nodes in the semantic network.

For example, a representation of the sentences, "A is a block" and "A is clear" is shown in Fig. 1. Informally, the nodes labelled M21 and M20 represent individuals which the agent will express as BLOCK and A, respectively. The nodes at the head of lex arcs are the agent's link to the external, extensional world. I.e., our (the user's) interpretation of the node at the tail. M22 is the structured individual node that represents the proposition that the individual represented by the node at the end of the member arc is a member of the class of objects represented by the node at the end of the class arc (thus, A is a member of the class of blocks). Similarly, M23 represents the proposition that A is clear. Rather than drawing networks, in later sections, we will write these in linear notation as predicates. We will express the propositions M23 and M22 as

M23!:Clear(A),

M22!:Isa(A,BLOCK),

respectively. We will express predicates by writing their name beginning with an uppercase letter followed by a combination of lower and uppercase letters. The proposition, if believed by the agent, is considered asserted in the agent's belief space and is indicated by writing the exclamation (!) following the node name (i.e., M22!). One of the important characteristics of such a representation is that the node M22!, which represents the proposition, is itself a term in the logic underlying SNePS. Any concept represented in the network may be the object of propositions represented in the network giving properties of, or beliefs about it. This allows for representations of nested beliefs.

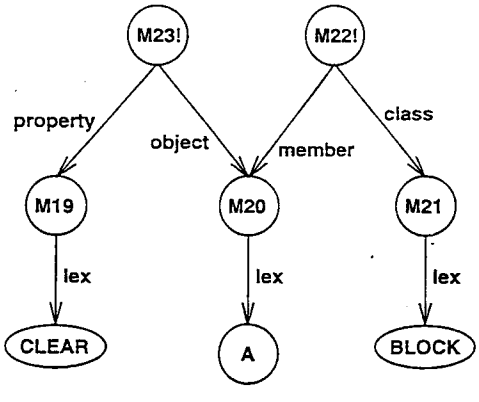


Fig. 1. SNePS representation of "A is a block" and "A is clear".

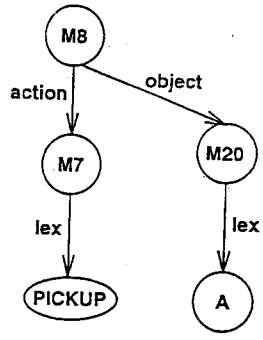


Fig. 2. SNePS representation of the act of picking up an object.

3.2. Acts and plans

In Refs. [7,10,21,23] we describe the SNePS propositional representations for plans and acts. An act, in SNePS, is represented as a structured individual node (i.e., it is also an intensional object). In Fig. 2, node M7 represents the action which the agent will express as PICKUP. The node M8 represents the *act* of picking up the object represented by the node at the end of the object arc (which is A). An *action* is that component of an act that is what is done to the object(s). By the Uniqueness Principle, a single act must be represented by a single SNePS node, even if there are several different structures representing propositions that several different actors performed that act at different times. Thus, M8 represents the act of picking up A. Any beliefs about the act of picking up A will have arcs coming into the node M8. See Refs. [7,23] for a more detailed discussion on the intensional aspects of these representations. In later sections we will denote acts using a linear notation. Thus, M8 will be expressed as

PICKUP(A).

We are following the convention that acts will be written mostly in all uppercase letters. Our

present model of acting is based upon a state-change model which is defined in Refs. [9,10]. We identify three types of states:

- External (or Physical) states.
- Mental states, also called *belief spaces*, are the beliefs held by the agent in that state.
- Intentional states (the agent's current intentions).

Accordingly, we identify three classes of actions:

Physical Actions: Actions that bring about changes in the physical world.

Mental Actions: Actions that bring about changes in the agent's belief space.

Control Actions: Actions that bring about changes in the agent's intentional states.

Thus, PICKUP is a physical action. We have BELIEVE and DISBELIEVE as mental actions whose objects are propositions. Control actions are described below. Additionally, acts can also be *primitive* or *complex*. A primitive act has an effectory procedural component which is executed when the act is performed. Complex acts are performed by deriving and executing appropriate 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 *qualifier acts*. These are summarized in Table 1. New acts of any kind can be defined by users depending upon the specifics of the modeling situation at hand.

3.3. Transformers

In Refs. [6,7,9] we introduced the generalized notion of a *transformer* as a propositional repre-

Table 1
Summary of control actions

Control Action	Description
SNSEQUENCE(a_1, a_2)	*Sequencing Act*: The acts a_1 and a_2 are performed in sequence. Example: SNSEQUENCE(PICKUP)(A), PUT(A, TABLE)) is the act of first picking up A and then putting it on the table.
DoONE(a_1,...,a_n)	*Disjunctive Act*: One of the acts a_1,...,a_n is performed. Example:DoONE(PICKUP(A),PICKUP(B)) is the act of picking up A or picking up B.
DoALL(a_1,...a_n)	*Conjunctive Act*: All acts $a_1, \ldots, a_n$ are performed in some order. Example: DoALL(PICKUP(A), PICKUP(B)) is the act of picking up A and picking up B.
SNIF(((p_1,a_1),...,(p_n,a_n))	*Conditional Act*: Some act a_i whose p_i is believed is performed. Example: SNIF (Clear(A), Pickup(A)),(Clear(B), PICKUP(B))) is the act of picking up A (if A is clear) or picking up B (if B is clear).
SNITERATE((p_1,a_1),...,(p_n,a_n))	*Iterative Act*: Some act in a_i whose corresponding p_i is believed is performed and the act is repeated. Example: SNITERATE((Clear(A), PICKUP(A)),(Clear(B),PICKUP(B))) is the act of picking up A (if A is clear) and picking up B (if B is clear).
ACHIEVE(p)	The act of achieving the proposition p. Example: ACHIEVE (Clear(A)) is the act of achieving that A is clear.
WITHSOME(x_1...)[p(x_1,...)]a(x_1...)	*Qualifier Act*: A single-object qualifier act. Find some x_1,...etc., that satisfy p(x_1,...) and perform the act a on it. Example: WITHSOME(x)[Clear(x)]PICKUP(x) is the act, "pickup a clear block"
WITHALL(x_1...)[p(x_1,...)]a(x_1...)	*Qualifier Act*: A multiple-object qualifier act. Find all x_1,... etc., that satisfy p(x_1,...) and perform the act a on them. Example: WITHALL(x)[Clear(x)]PICKUP(x) is the act, "pick up all clear blocks"

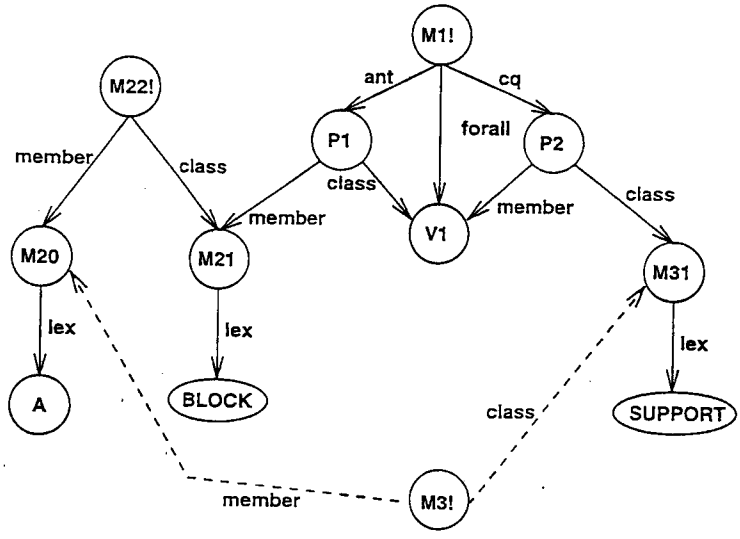


Fig. 3. SNePS representation of "A is a block", "All blocks are supports", and "A is a support". M1 is a belief–belief transformer. The antecedent belief is indicated by the ant arc emanating from M1 and the consequent belief is indicated by the cq arc. The forall arc is a quantifier arc binding the variable V1. P1 is the pattern that V1 is a member of the class represented by M21 (blocks). Similarly, P2 represents that V1 is a member of the class represented by M19 (supports). M3 is the derived proposition (See Section 4).

sentation that accounts for various notions of inference and acting. In general, a transformer is a pair of entities (⟨a⟩, ⟨b⟩), where both ⟨a⟩ and ⟨b⟩ can specify beliefs or acts. Thus, when both parts of a transformer specify beliefs, it represents a reasoning rule. When one of its parts specifies beliefs and the other acts, it can represent either an act's preconditions, or its effects, or a reaction to some beliefs, and so on. What a transformer represents is made specific by specifying its parts. When believed (remember, a transformer itself is a proposition!), transformers can be used during the acting/inference process, which is where they derive their name: they transform acts or beliefs into other beliefs or acts and vice versa.

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 ⟨a⟩, it believes or intends to perform ⟨b⟩". The backward chaining interpretation of a transformer is "if the agent wants to believe (or know if it believes) or perform ⟨b⟩, it must first believe (or see if it believes) or perform ⟨a⟩". Since both ⟨a⟩ and ⟨b⟩ can be sets of beliefs or an act, we have four types of transformers: *belief–belief, belief–act, act–belief*, and *act–act.*

Belief–belief transformers

These are standard reasoning rules (where ⟨a⟩ is a set of antecedent belief(s) and ⟨b⟩ is a set of consequent belief(s)). Fig. 3 shows a SNePS representation of "All blocks are supports". Such rules in SNePS can be used in forward, backward, as well as bidirectional inference to derive new beliefs. SWM [1], the underlying logic of SNePS inference, has a built in specification for assumption-based truth maintenance. The inference mechanism employed is a natural deduction system. Hereafter, we will call these AntCq transformers and use the linear notation ⟨a⟩ → ⟨b⟩ to write them. For example M1 in Fig. 3 will be written as

M1!:∀v1[Isa(v1,BLOCK) → Isa(v1,SUPPORT)],

[1] SWM stands for Shapiro, Wand and Martins [14]

where v1 is a universally quantified variable (its network representation is shown in Fig. 3). In addition to the connective above (which is also called an or-entailment), the vocabulary of SNePS connectives includes and-entailment, numerical-entailment, and-or, thresh, and non-derivable. Other quantifiers include the existential, and the numerical quantifiers (see [22]).

Belief–act transformers

These are transformers where ⟨a⟩ is a set of belief(s) and ⟨b⟩ is a set of acts. They enable representation of preconditions of actions as well as rules for reactivity. Used during backward chaining, these can be propositions specifying preconditions of actions, i.e., ⟨a⟩ is a precondition of some act ⟨b⟩. We will call them PreconditionAct transformers and write them as predicates PreconditionAct(⟨a⟩, ⟨b⟩). For example, the sentence "Before picking up A it must be clear" may be represented as

M26!:PreconditionAct(Clear(A),PICKUP(A)).

Using AntCq transformers the agent can also represent general preconditions for actions. For example, "Before picking up any block make sure that it is clear" can be represented as

M15!:∀x[Isa(x,BLOCK)

→ PreconditionAct(Clear(x),PICKUP(x))]

from which M26 above can be derived.

Used during forward chaining, these transformers can be propositions specifying the agent's desires to react to certain situations, i.e., the agent, upon coming to believe ⟨a⟩ will form an intention to perform ⟨b⟩.

We will call these WhenDo transformers and denote them as WhenDo(⟨a⟩, ⟨b⟩) predicates. For example, the sentence representing the desire, "When A is clear pick it up" can be represented as

M16!:WhenDo(Clear(A),PICKUP(A)),

or general desires like, "whenever something is broken, fix it" can be represented as

M17!:∀x[WhenDo(Broken(x),FIX(x))],

where FIX may be some complex act.

Act–belief transformers

These are the propositions specifying effects of actions as well as those specifying plans for achieving goals. They will be denoted as ActEffect and PlanGoal transformers, respectively. The ActEffect transformer will be used in forward chaining to accomplish believing the effects of act ⟨a⟩. For example, the sentence, "After picking up A it is no longer clear" is represented as

M30!:ActEffect(PICKUP(A),¬Clear(A)).

It can also be used in backward chaining during the plan generation process (classical planning). The PlanGoal transformer is used during backward chaining to decompose the achieving of a goal ⟨b⟩ into a plan ⟨a⟩. A goal is simply a proposition that the agent wants to achieve. For example, "A plan to achieve that A is held is to pick it up" is represented as

M56!:PlanGoal(PICKUP(A),Held(A)),

or the general plan, "a plan to achieve that a block is held is to pick it up" can be represented as

M57!:∀x[Isa(x,BLOCK)

→ PlanGoal(PICKUP(x),Held(x))].

This transformer can also serve a useful purpose during plan recognition: if the agent believes that ⟨a⟩ is a plan for the goal ⟨b⟩ then if some act a_i is part of the plan a and the agent believes that someone performed the act a_i it might be that they were engaged in carrying out the plan a in order to achieve the goal b.

Another backward chaining interpretation that can be derived from this transformer is "if the agent wants to believe ⟨b⟩, it must perform ⟨a⟩", which is represented as a DoWhen transformer. For example, "look at A to find out its color" can be represented as

M60!:DoWhen(LOOKAT(A),Color(A,?y)),

or for the general form, "look at an object if you want to know its color" we will have

M59!:∀x[DoWhen(LOOKAT(x),Color(x,?y))],

where, in both the above cases, ?y is an unbound variable.

Act–act transformers

These are propositions specifying plan decompositions for complex actions (called PlanAct transformers), where ⟨b⟩ is a complex act and ⟨a⟩ is a plan that decomposes it into simpler acts. For example, in the sentence, "to pile A on B first put B on the table and then put A on B" (where piling involves creating a pile of two blocks on a table), piling is a complex act and the plan that decomposes it is expressed in the proposition.

M71!:PlanAct(SNSEQUENCE(PUT

(B,TABLE),PUT(A,B)),PILE(A,B)).

Some aspects relevant to the implementation model will become clear later. The important thing to remember is that there are propositional representations for beliefs, rules, plans, and acts and that forward or backward chaining through them denotes various notions of reasoning, planning, acting, and reacting. Transformers help capture situations where beliefs may lead to actions and vice versa. In what follows, we first give an informal presentation of the message passing model of inference in SNePS. The abstract model will then be presented for inference as well as acting. We have not presented a discussion of the underlying rules of inference. This is to stress the idea that a commitment to a uniform ontology does not necessarily commit one to a specific logic. In other words, the architecture makes certain semantic and ontological commitments, not necessarily logical ones. Interested readers can see Ref. [14] for a formal presentation of SWM logic, the logic underlying SNePS.

4. An example: Inference

Informally, inference is accomplished by means of message passing between nodes in the network. Nodes "talk" to each other by sending messages. To understand the process, indulge with us and picture the following "conversation" between nodes in the network of Fig. 3.

Sender	Receiver	Message
USER	P2	What is a support?
P2	M1	HELP!
M1	P1	What is a block?
P1	M22	What is a block?
M22	P1	A is a block.
P1	M1	A is a block.
M1	P2	Since A is a block, and all blocks are supports, A is a support. (A node, M3, is built respresenting the derived proposition.)
P2	M3	A is a support!
M3	USER	A is a support.

The above depicts a case where a user query backchains through a belief–belief transformer to produce an answer. The network, after the query is answered, is shown in Fig. 3. The agent now believes that "A is a support" (the proposition M3!, shown in Fig. 3 with dotted lines) and along with that proposition, the truth maintenance system will associate the assumptions (M1! and M22!) used in deriving it. In SWM, the logic underlying the inference and belief revision system, M1 and M22 are called *supported wffs* (swffs). Associated with each swff is a *support* containing an *origin tag*, which is hyp for hypotheses, and der for derived swffs; an *origin set*, which contains those (and only those) hypotheses used in the derivation of the swff; and a *restriction set*, which records inconsistency information. All beliefs of the agent reside in a *belief space* which is a set of all the hypotheses and all the swffs derived from them. Thus, the propositions M1 and M22 shown in Fig. 3 are hypotheses, and M3 is a derived proposition. Together, they form the agent's current belief space. The support sets for M22, M1 and M3 will be ⟨hyp, {M22!}, {}⟩, ⟨hyp, {M1!}, {}⟩ and ⟨der, {M1!, M22!}, {}⟩, respectively.

The very same rule can be used, similarly, in forward (as well as bi-directional) inferences. In the remainder of the paper we present details of the message passing model by defining types of messages and types of nodes. We introduce the idea of an agenda of a node which, together with

the type of node and its incoming message, determines the role of the node in the spreading activation process. The message passing model accomplishes inference, acting, deductive plan retrieval, hierarchical plan decomposition, determining the preconditions and effects of an action, as well as some notions of reactivity.

5. The rational engine

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. SNeRE, the SNePS Rational Engine: an embodiment of this model will be described here. It should be noted that the model presented here is not limited only to SNePS. In the presence of any unified representational formalism described on the lines of Section 3 above a rational engine can be developed (See Section 7 below). The rational engine employs a quasi-concurrent [27] message passing model that accounts for the various notions of acting and inference. Message passing is a parallel activity in a SNePS network thus representing a kind of spreading of activation. However, unlike traditional notions of spreading activation, the spreading of activation is governed by the underlying logic and propositional nature of the nodes in the network.

5.1. Channels

Communication (or message passing) between nodes takes place along *channels*. A node can send and receive messages only via channels. We define two types of channels that may exist between any two nodes in the network:

Match channels. The molecular structure under the two nodes unifies. For example, in Fig. 3 there is a match channel between the nodes M22 and P1 as the structure under them unifies (with binding {M20/V1}).

Transformer channels. The two nodes are connected to each other by a transformer. For example, a transformer channel exists between M1! and P1 in Fig. 3 since P1 is in antecedent position of a *belief–belief* transformer.

Only the proposition, act, and transformer nodes are capable of processing messages. Channels may also exist between nodes in the network and the user to enable interaction 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. Upon receiving a message a node may perform some book keeping and respond by sending out some message. The nature of book keeping and outgoing messages depends on the type of the node, the type and content of the incoming message, and the node's agenda. These are detailed below.

5.2. Messages

There are three types of messages:

Believe(p)?, where p is a molecular node in the SNePS network. Also called a *request*, it denotes that the sender is asking (requesting) the receiving node about the assertional status of p.

Believe(p)!, where p is a propositional node 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 node and the message indicates that in the current state the act a is being intended to be performed.

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

5.3. Agendas

The agenda of a node determines the node's current role in the acting and inference process.

Table 2
Message processing by proposition nodes (p is the receiver, and s the sender)

	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.

Different types of nodes have different agendas. Each node, 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 proposition, act, and transformer nodes partake in message passing we will describe the agendas defined for each type of node along with the node's message handling behavior.

5.4. Message handling: Proposition nodes

Proposition nodes can have the following agendas:

ASSERTED. The node is asserted in the current belief space, i.e., the represented proposition is believed by the agent. This is indicated by writing an exclamation mark (!) after the name of the node (as in M1!).

UNASSERTED. The node is not asserted in the current belief space.

Proposition nodes only send/receive belief requests and reports. Table 2 defines the message handling capabilities of proposition nodes.

5.5. Message handling: Belief–belief transformers

Belief–belief transformers, or rule nodes, are also propositions and, hence, may or may not have an assertional status. Only when a rule node is asserted does it get involved in inference. Asserted rule nodes have the following agendas:

NONE. The node is asserted but does not have any agenda.

ACTIVE. The node is asserted and is currently involved in some ongoing inference (like trying to determine if the antecedents are believed).

When a rule node receives a request from one of its consequents (i.e., a Cq transformer channel) a backward chaining inference is in process. When a rule node receives a report from its antecedents (Ant transformer channel) either an earlier back-

Table 3
Message processing by belief–belief transformers

	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(p)*	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.

ward inference may be completed (if its agenda is ACTIVE), i.e., the rule may be fired, or a forward chaining is in progress. Table 3 defines the message processing capabilities of rule nodes. Note that the same rule is being used for both types of inference.

5.6. Message handling: Act nodes

Act nodes typically get involved in message passing when they receive an *Intend* message. This may be due to a direct request by a user to perform some act or any of the various ways beliefs may get transformed into intentions (belief–act transformers). Several things need to be accomplished in order to do an act. These form the agenda of an act. Following agendas have been defined for an act node:

START: signifies the beginning of an attempt by the agent to perform the act,

FIND-PRECONDITIONS: the agent is in the process of determining the act's preconditions,

TEST-PRECONDITIONS: the agent is testing to see if it believes the preconditions of the act,

FIND-EFFECTS: the agent is trying to determine the effects of the act,

FIND-PLAN: the agent is trying to find a plan to do the act,

EXECUTE: the agent is ready to execute the action,

DONE: the action has been performed, wind up.

The message handling capabilities of act nodes are defined in Table 4. Notice that inferences are performed to deduce the preconditions, and effects of the act, as well as to determine plans for complex acts. If an inference started by an act node is not successful the act node receives the *Intend* message again. The agenda helps the act node in determining what to do next. Notice that

Table 4
Message passing by act nodes

	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-EFFECT	Change agenda to EXECUTE. Send message *Believe*(Primitive(a))?
5	*Intend*(a)	EXECUTE	Change agenda to FIND-PLANS Send request *Believe* (Plantct (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.
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).

in Table 4 entries 7 and 8 are the specification of backward chaining transformation through the PreconditionAct transformers, entry 10 is the forward chaining specification through the ActEffect transformer, and entry 11 is backward chaining through the PlanAct transformers. The overall process of acting is a pure deductive approach to acting. Since the TMS maintains a consistent belief space at all times, we claim the agent acts *rationally* based on its beliefs.

The above specifications of message processing behavior of nodes realizes an implementation of an agent which, using its propositional representations of beliefs, plans, and acts, is able to perform the tasks of reasoning and acting using the uniform reasoning component.

5.7. Example revisited: Inference

The example of Section 4 can now be presented with a little more detail.

Sender	Receiver	Message	Description
USER	P2	*Believe*(P2)?	What is a support?
P2	M1	*Believe*(P2)	Help!
M1	P1	*Believe*(P1)?	What is a support?
P1	M22	*Believe*(M22)?	What is a block?
M22	P1	*Believe*(M22)!	A is a block!
P1	M1	*Believe*(M22)!	A is a block!
M1	P2	*Believe*(M3)!	Since A is a block, and all blocks are supports, A is a support.
P2	M3	*Believe*(M3)!	A is a support!
M3	USER	*Believe*(M3)!	A is a support.

First, USER sends a request to P2. Since P2 is an unasserted proposition node, entry 2 of Table 2 indicates that the message is passed to all rules nodes connected via consequent arcs (see Fig. 3), i.e., to M1. M1 is an asserted rule node, thus entry 1 of Table 3 applies: M1's agenda is changed to ACTIVE; and a *Believe*(P1)? message is sent to the antecedent P1. P1's agenda is UNASSERTED, entry 2, Table 2 specifies that a request be sent to match channels. There is a match channel to M22, thus the message *Believe*(M22)? is dispatched. M22 is ASSERTED, entry 1, Table 2 specifies that the message *Believe*(M22)! be sent to P1. Entry 3, Table 2 applies to P1 so the message is forwarded on to M1. M1 (using entry 3, Table 2) confirms that all the antecedents of the rule are satisfied, thus the rule can be fired, a node M3 representing the consequent is added to the network and the message *Believe*(M3)! is sent to P2. P2 forwards it to M3 and USER. M3 gets asserted and USER gets the response.

6. Example: Acting and inference

We will demonstrate the process of integrated acting and inference using a simplified model of the act of picking up a block in a blocksworld. We inform the agent about the action by first saying

All blocks are supports.
Picking up is a primitive action.

which results in the propositions represented by M1 and M2 (see Fig. 4). Preconditions of acts are also represented as propositions. Thus, the input

Before picking up a block the block must be clear.

is interpreted as a generic rule specifying a precondition for picking up a block. The rule is

No.: Formula	Support
M1! : ∀x[Isa(x, BLOCK) → Isa(x, SUPPORT)]	< hyp, {M1!}, {} >
M2! : Isa(PICKUP, PRIMITIVE)	< hyp, {M2!}, {} >
M3! : ∀x[Isa(x, BLOCK) → PreconditionAct(Clear(x), PICKUP(x))]	< hyp, {M3!}, {} >
M6! : ∀x[Isa(x, BLOCK) → ActEffect(PICKUP(x), ¬Clear(x))]	< hyp, {M6!}, {} >
M11! : ∀x[Isa(x, BLOCK) → ActEffect(PICKUP(x), Held(x))]	< hyp, {M11!}, {} >
M22! : Isa(A, BLOCK)	< hyp, {M22!}, {} >
M23! : Clear(A)	< hyp, {M23!}, {} >
M26! : PreconditionAct(Clear(A), PICKUP(A))	< der, {M22!, M3!}, {} >
M29! : ActEffect(PICKUP(A), Held(A))	< der, {M22!, M11!}, {} >
M30! : ActEffect(PICKUP(A), ¬Clear(A))	< der, {M22!, M6!}, {} >

Fig. 4. The agent's belief space after the preconditions and effects of PICKUP(A) have been deduced. (M1: All blocks are supports. M2: Picking up is a primitive action. M3: Before picking up a block the block must be clear. M6: After picking up a block is not clear. M11: After picking up a block the block is held. M22: A is a block. M23: A is clear. M26: A precondition of picking up A is that A is clear. M29: An effect of picking up A is that A is held. M30: An effect of picking up A is that A is no longer clear.)

No.: Formula	Support
M1! : ∀x[Isa(x, BLOCK) → Isa(x, SUPPORT)]	< hyp, {M1!}, {} >
M2! : Isa(PICKUP, PRIMITIVE)	< hyp, {M2!}, {} >
M3! : ∀x[Isa(x, BLOCK) → PreconditionAct(Clear(x), PICKUP(x))]	< hyp, {M3!}, {} >
M6! : ∀x[Isa(x, BLOCK) → ActEffect(PICKUP(x), ¬Clear(x))]	< hyp, {M6!}, {} >
M11! : ∀x[Isa(x, BLOCK) → ActEffect(PICKUP(x), Held(x))]	< hyp, {M11!}, {} >
M22! : Isa(A, BLOCK)	< hyp, {M22!}, {} >
M26! : PreconditionAct(Clear(A), PICKUP(A))	< der, {M22!, M3!}, {} >
M29! : ActEffect(PICKUP(A), Held(A))	< der, {M22!, M11!}, {} >
M30! : ActEffect(PICKUP(A), ¬Clear(A))	< der, {M22!, M6!}, {} >
M28! : ¬Clear(A)	< hyp, {M28!}, {} >
M27! : Held(A)	< hyp, {M27!}, {} >

Fig. 5. Belief space of the agent after the act PICKUP(A) is performed. (M1: All blocks are supports. M2: Picking up is a primitive action. M3:Before picking up a block the block must be clear. M6: After picking up a block the block is not clear. M11: After picking up a block the block is held. M22: A is a block. M26: A precondition of picking up A is that A is clear. M29: An effect of picking up A is that A is held. M30: An effect of picking up A is that A is no longer clear. M28: A is not clear. M27: A is held.)

represented by node M3 in Fig. 4. It could be paraphrased as "For all x, if x is a block then the act of picking up x has the precondition that x is clear". Effects are similarly represented. Thus, the following

After picking up a block the block is not clear and the block is held.

is represented by two rules: one specifying the effect that the block is no longer clear (M6); and the other specifying that the block is held (M11). Further, the agent is told

A is a block.
A is clear.

which are represented as M22! and M23! in Fig. 4 [2]. When the user asks the agent to perform the act

Pickup A.

[2] Note that in this simplistic model we have purposely avoided the mention of a table and that A is on the table. The purpose of this example is to illustrate the computational rather than the representational features of the model. A complete example of the blocksworld can be found in [10].

an *Intend* message is sent to PICKUP(A), its current agenda is START, it changes its agenda to FIND-PRECONDITIONS and sends a message

Believe(PreconditionAct(p, PICKUP(A)))?

over a match channel to the consequent of M3! thus starting a backward chaining inference. The inference proceeds as outlined in Section 5.7, is successful and the act PICKUP(A) receives the message

Believe(PreconditionAct(Clear(A),

PICKUP(A)))!

Entry 7 of Table 4 specifies that the act node change its agenda to TEST-PRECONDITIONS and send a message

Believe(Clear(A))?

I.e., the agent is testing to see if the act's preconditions are satisfied (via backward chaining though a belief–act transformer). The proposition M23 is asserted in the agent's belief space (Fig. 4) thus it responds to PICKUP(A) with the message

Believe(Clear(A))!

Entry 8 of Table 4 specifies that its agenda be changed to FIND-EFFECTS and it sends a message

Believe(ActEffect(PICKUP(A),e))?

Upon conclusion of this inference the act node receives the messages

Believe(ActEffect(PICKUP(A),¬Clear(A)))!

and

Believe(ActEffect(PICKUP(A),Held(A)))!

The agent's belief space is now as shown in Fig. 4 where M26, M29, M30 are the inferred propositions so far.

Since the agent has determined that the act's preconditions are satisfied and it has also determined the effects of the act, the agenda of the act is changed to EXECUTE (entry 9, Table 4) and a *Believe* message is sent to determine if the agent believes that the act is primitive (which is represented by the proposition M2!). Entry 10 specifies that the effector component of the act be executed (thus bringing about a change in the external world) and an *Intend* message be sent to the act

DoALL(BELIEVE(¬Clear(A)),
BELIEVE(Held(A)))

The two acts are similarly carried out, and as a result the agent has the beliefs shown in Fig. 5.

In the SNePS acting system we define two *mental* actions: BELIEVE and DISBELIEVE, that are used to update the beliefs of the agent after an action is performed. The effectory components of the two actions are the TMS operations add-to-context and remove-from-context, respectively. The TMS facilitates automatic revision of the belief space after a hypothesis is removed as a result of some DISBELIEVE action (all derived beliefs having the disbelieved hypothesis in their origin set are also removed). This implements the *extended* STRIPS assumption [3]. There are several advantages to this approach. They are presented in Ref. [11].

The agendas for the WhenDo and DoWhen transformers are similar to those of the belief–belief transformers. The WhenDo transformer upon becoming active during forward chaining results in an intention to perform its action, thus modeling reactivity. Similarly, the DoWhen transformer activates during backward chaining inference and results in an action to be performed, which could lead to an acquisition of new beliefs. Thus, the notions of reactivity and knowledge acquisition are not only represented, they are utilized by the rational engine during the course of inference and acting. The next section illustrates the use of the DoWhen transformer.

7. Acting in service of inference

In the example presented above, it is clear that inference is being used in service of acting. This is the way all AI architectures are built. In our architecture, it is also possible for acting to be performed in service of inference. Suppose the agent has the following belief:

Red colored blocks are wooden.

and is asked the query:

Is A wooden?

We will present the reasoning process below using english sentences (italicized ones are system responses) rather than messages. The agent, in response to the above query will proceed as follows:

I wonder if A is wooden.
I wonder if A is a block.
I wonder if A is red.
I know A is a block.
I do not know if A is wooden.

Typically, this would be a natural place for the agent to perform an act, like looking at A, in order to determine its color. Since inference is modeled as a subordinate process to acting in traditional systems, it would be difficult to model the desired behavior. However, in the SNePS BDI architecture, the agent can have a desire.

If you want to know the color of a block look at it.

which is represented using the DoWhen transformer shown in Section 3.3 above. The above query will now proceed as follows:

Is A wooden?
I wonder if A is wooden.
I wonder if A is colored red.
I wonder if A is a block.
I know A is a block.
Since A is a block I infer
If you want to know the color of A look at it.
I intend to do the act look at A.
I wonder if the act look at A has any preconditions.
...
Now doing: Look at A.
Sensory-add: *A is colored red.*
Since A is a block and A is colored red and all red colored blocks are wooden
I infer A is wooden.

Notice how, in the above example, a backward chaining query lead the agent to perform an action in order to answer the query. Thus, acting was performed in service of inference. See Ref. [12] for a detailed discussion on this aspect of the architecture. It is conceivable that the acts performed in service of inference could be the ones that query other agents (and or databases).

8. Future work

In this paper we have presented the design of the SNePS BDI architecture. The architecture uses a uniform formalism for the representation of beliefs, acts, plans, and rules. We also presented the design of SNeRE, the SNePS Rational Engine, and demonstrated how a parallel message passing model can be used to implement a unified model of acting and inference. Message passing between processes is accomplished using MULTI, a simulated multi-processing system [16] which is similar to the Manchester Data Flow Machine [26] and is implemented in Common Lisp.

We have experimented with this model in a blocksworld domain (with simulated graphics). We have also developed a prototype intelligent-agent user interface to a geographical information system (GIS) called ARC/INFO [20]. In this system the agent acts as a natural language front-end to the GIS. The operations on the GIS form the agent's domain of actions and plans. The model is also being used for testing and implementing communicative speech act theories [4] and natural language generation [5].

Our model can also be viewed as a meta-level implementation of propositional attitudes. This implies that, if interested, one can design more types of messages based on different propositional attitudes (so far we have only experimented with Belief, and Intentionality). We have adopted a very simplistic theory of acting which is not very different from most existing AI Planning/Acting systems from a behavioral point of view. However, we suggest that this model be viewed as a generic AI architecture rather than a specific embodiment of a theory of acting. The action ontology does not make any teleological commitment, only a representational one. The specification of the message passing behavior seems to commit us to the simplistic view of acting. However, the message passing behavior can be made user-specifiable (depending upon action theory being employed). For instance, one may wish to use the forward chaining ActEffect transformer only when it can be confirmed (maybe via sensory acts) that the external world does actually comply according to the results of the action just performed. This only involves a slight modification in Table 4 (entry 10). The model can be made extremely amenable to such extensions if reformulated using an object-oriented perspective [8]. Ref. [7] gives a concurrent-object-oriented specification of a BDI architecture based on these ideas.

9. Remarks

A basic premise of our approach stems from the empirical observation that, typically, good knowledge representation and reasoning systems are bad candidates for planning/acting modeling and vice versa. If one wishes to extend a good KR

system for planning/acting modeling one can take the easy way out by simply integrating a mutually exclusive off-the-shelf planning/acting system. This only results in paradigm soups. The approach we have taken is to extend the KR system by extending its ontology and at the same time preserving its foundations. Additionally, we have provided a unified computational model of inference and acting. The architecture provides a unique opportunity to research various AI faculties from an integrated perspective. The modeled agent in this architecture has natural language capabilities, is able to represent and reason about beliefs, actions, and plans. The agent is able to carry out its plans. Acting and reasoning is accomplished using a simple message passing scheme. The TMS and belief revision facilities are an integral part of the system. Benefits derived from the ATMS and discussed in Ref. [11]. The overall model is designed to be extendible. The main thrust of our research is that integration of various AI subsystems, if performed from a representational perspective can result in simpler, more manageable, elegant, yet powerful AI architectures.

Acknowledgements

The author wishes to thank Richard Wyatt and Hans Chalupsky for discussion and feedback on this paper. Special thanks also to the anonymous reviewers of this paper for making suggestions that enhanced the focus of this paper.

References

[1] James Allen, The RHET system, In Charles Rich (Guest Ed.), SIGART BULLETIN Special Issue on Implemented KRR Systems, pp. 1–7 (1991).

[2] Richard E. Fikes, and Nils J. Nilsson, STRIPS: A new approach to the application of theorem proving to problem solving, Artificial Intelligence, 5, pp. 189–208 (1971).

[3] Michael P. Georgeff, Planning, In Annual Reviews of Computer Science, Vol. 2, pp. 359–400 (Palo Alto, CA, Annual Reviews Inc., 1987).

[4] Susan Haller, Intention and action in natural language, In Deepak Kumar, Hans Chalupsky, and Syed S. Ali (Eds.), Current Trends in SNePS, Proceedings: 1990 Workshop (1990).

[5] Susan Haller, Interactive generation of plan justifications, In M. Zock, and K. DeSchmed (Eds.), Proceedings: 4th European Workshop on Natural Language Generation (1993).

[6] D. Kumar., An integrated model of acting and inference, In D. Kumar (Ed.), Current Trends in SNePS-Semantic Network Processing System: Proceedings: 1st Annual SNePS Workshop, pp. 55–65, Buffalo, NY (1990).

[7] Deepak Kumar, From beliefs and goals to intentions and actions: An amalgamated model of acting and inference, PhD thesis, State University of New York at Buffalo (1993).

[8] Deepak Kumar, Susan Haller, and Syed S. Ali, Towards a unified AI formalism, In Proceedings: 27th Hawaii International Conference on System Sciences, IEEE Computer Society Press, Los Alamitos, CA (1994).

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

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

[11] Deepak Kumar, and Stuart C. Shapiro, Deductive efficiency, belief revision and acting, Journal of Experimental and Theoretical Artificial Intelligence (JETAI), 5(2) (1993).

[12] Deepak Kumar, and Stuart C. Shapiro, Acting in service of inference (and vice versa), In Proceedings: 7th Florida AI Research Symposium (FLAIRS 93), The Florida AI Research Society (1994).

[13] Douglas B. Lenat, and Ramanathan V. Guha, The evolution of CYCL: The CYC representation language, In Charles Rich (Guest Ed.), SIGART BULLETIN Special Issue on Implemented KRR Systems, pp. 84–87 (1991).

[14] J.P. Martins, and S.C. Shapiro, A model for belief revision, Artificial Intelligence, 35(1), pp. 25–79 (1988).

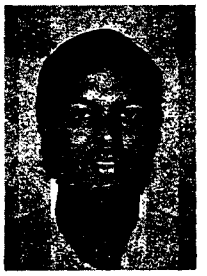
[15] John McCarthy, Mental situation calculus, In Joseph Y. Halpern (Ed.), Theoretical Aspects of Reasoning about Knowledge, Proceedings: 1986 Conference, page 307 (1986).

[16] D.P. McKay, and S.C. Shapiro, MULTI: a LISP based multiprocessing system, In Proceedings: 1980 LISP Conference, pp. 29–37, Stanford University, Stanford, CA (1980).

[17] Charles Rich, CAKE: An implemented hybrid KR and limited reasoning system, In Charles Rich (Guest Ed.), SIGART BULLETIN Special Issue on Implemented KRR Systems, pp. 120–127 (1991).

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

[19] Earl D. Sacerdoti, A Structure for Plans and Behavior (Amsterdam, Elsevier Science B.V., 1977).
[20] Stuart C. Shapiro, Hans Chalupsky, and Hsueh-Cheng Chou, Connecting ARC/INFO and SNACTor, Technical Report 91–13, Department of Computer Science, SUNY at Buffalo, Buffalo, NY, 22 pages (1991).
[21] S. C. Shapiro, Representing plans and acts, In Proceedings: 3rd Annual Workshop on Conceptual Graphs, pp. 3.2.7-1-3.2.7-6. The American Association for Artificial Intelligence, Menlo Park, CA (1988).
[22] S. C. Shapiro, and The SNePS Implementation Group, SNePS-2 User's Manual,. Department of Computer Science, SUNY at Buffalo (1989).
[23] S. C. Shapiro, D. Kumar, and S. Ali, A propositional network approach to plans and plan recognition, In Proceedings: 1988 Workshop on Plan Recognition, p. 21, Los Altos, CA (1989).
[24] S. C. Shapiro, and W. J. Rapaport, SNePS considered as a fully intensional propositional semantic network, In N. Cercone, and G. McCalla (Eds.), The Knowledge Frontier, pp. 263–315 (Berlin, Springer-Verlag, 1987).
[25] Stuart C. Shapiro, Case studies of SNePS, SIGART Bulletin, 2(3), pp. 128–134 (1991).
[26] Ian Watson, and John Gurd, A practical data flow computer, IEEE Computer, 15(2), pp. 51–57 (1982).
[27] Peter Wegner, Dimensions of object-based language design, Proceedings: OOPSLA-87, pp. 168–182 (1987).
[28] David E. Wilkins, Practical Planning-Extending the Classical AI Planning Paradigm (Palo Alto, CA, Morgan Kaufmann, 1988).



Deepak Kumar obtained his PhD from the State University of New York at Buffalo in 1993. He is currently an assistant professor of computer science at Bryn Mawr College. His research interests include Knowledge Representation and Reasoning, Planning, and AI architectures. He is also responsible for starting a new computer science program at Bryn Mawr College.