Agent Programming Languages with Declarative Goals : A Survey

Shakil M. Khan

Technical Report CSE-2007-07

April 1 2007

Department of Computer Science and Engineering

4700 Keele Street Toronto, Ontario M3J 1P3 Canada

Agent Programming Languages with Declarative Goals: A Survey

(Qualifying Oral Document)

Shakil M. Khan

A document submitted to the Faculty of Graduate Studies
in partial fulfilment of the requirements
for the degree of

Doctor of Philosophy

Graduate Programme in Department of Computer Science and Engineering
York University
Toronto, Ontario
April 2007

Contents

1 Introduction

Research on intelligent agents can be divided into three related areas that focus on integrated treatments of agents: agent theories, architectures, and languages. All three of these areas are concerned with different aspects of the same enterprise, namely, the specification and implementation of intelligent behavior. Agent theories use formal logic to model the agent's mental attitudes (e.g. beliefs, desires, intentions, etc.) and how they relate to each other and to the agent's behavior. These theories has also grappled with problems associated with multiple agents, such as the coordination of and communication between a group of agents, and the modeling of joint mental attitudes. An agent architecture provides mechanisms for managing the mental and physical resources of an agent to meet the demands of complex, dynamic environments. Agent programming languages attempt to bridge the gap between theory and practice. The term "agent-oriented programming language" was coined by Shoham [150]. He defined agent programming languages as programming languages with representational primitives corresponding to various mental state components of the agent, and where reasoning is performed on these primitives. The key issue in agent programming language design concerns the "right way" of programming an agent, i.e. an autonomous, reactive, proactive, situated, and interacting computing element [189, 9]. In other words, what kind of programming language components should these languages contain so as to allow the programmer to design an agent in the most convenient, most natural, most succinct, most efficient, and most comprehensible way, and how to effectively execute these agent programs. Shoham's ideas have been later taken up and modified by various researchers. As we will see, many agent programming languages do not explicitly represent intentional attitudes, and have only very little intellectual debt w.r.t. the original vision by Shoham.

According to [122], a key problem of agent programming research is to show a one-to-one correspondence between the underlying agent theory and the abstract interpreter that the agent programming language provides. Bridging the gap between agent theories and agent programming frameworks has proven to be a lasting problem in agent research. The omission of so called *declarative goals* from agent programming frameworks seems to be the primary reason for this gap. To be specific, most agent programming frameworks have focused mainly on plans or 'goals-to-do' (where goals are defined as a set of procedures which are executed to achieve the goal), in contrast to 'goals-to-be' (where goals are specified as a declarative description of the state of the world which is sought) [34]. These user written plans or programs speed up the computation in these systems making them more practical. However, a declarative notion of goal is also necessary if one wants to fully model the pro-activity of an agent. With the omission of declarative goals, the reason for executing plans is lost. In addition to providing a new abstraction mechanism, these goals can be used to decouple plan failure/success from goal failure/success. An agent may be able to successfully execute a plan. However, this does not necessarily mean that the agent was successful in achieving the associated goal, since the environment may interfere with this goal. Similarly, external interference may render a user-defined plan impossible to execute; but this does not necessarily mean that the agent will never be able to achieve the associated goal (see Section 5.1 for a concrete example). Thus, these declarative goals are essential for monitoring goal achievement and performing recovery when a plan has failed. Since these goals capture the reason

for executing plans, its not hard to see that they are also necessary to model rational deliberation and action, and to model rational response to changes in goals that result from communication, e.g. requests.

Recently, there has been a fair amount of work on establishing a link between agent logics and agent programming frameworks by incorporating declarative goals in agent programming frameworks [71, 172, 174, 34, 135, 185, 134]. Research in agent programming with declarative goals is still at an early stage. As such, most of these frameworks suffer from various problems. For instance, many of these frameworks do not provide a formal semantics for declarative goals. The dynamics of these goals are usually specified using some sort of syntactic manipulation directed by the 'operational semantics' of the language. Often, there are no requirements for an intended plan to be consistent with an intended declarative goal. Also, these frameworks often assume complete information, and thus do not formalize sensing actions. There has been very little work on deliberation and planning in these frameworks. The primary reason for these deficiencies can be identified and explained by the fact that there is a tradeoff between efficiency and expressiveness in these languages, and that these agent programming languages are supposed to be more practical than their agent theory counterparts. Thus defining practical rational agent programming languages is much harder than simply developing an agent theory.

This survey reviews research related to agent programming languages with declarative goals. It is organized in four sections. In the next three sections, relevant work on agent theories, agent architectures, and agent programming languages broadly speaking are considered in turn. Then in Section 5, we review work specifically on agent programming languages with declarative goals. Finally, we conclude this survey with some possible future research directions.

2 Agent Theories

Viewing an entity as an agent involves ascribing high-level cognitive attitudes such as beliefs, goals, desires, and intentions to agents. This is called taking an *intentional stance*. Often, we design agents by representing such attitudes explicitly and implementing reasoning procedures over them. Theories that formalize various aspects of agents are often known as belief-desire-intention (BDI) theories.

Most of the existing agent theories attempt to deal with one or more of following questions:

- What are the connections between agents' informational attitudes and their actions, i.e. what are the informational preconditions of actions and what effects do actions have on these attitudes?
- What motivational attitudes (e.g. goal, choice, and intention) are necessary, and how should these attitudes be formalized? How should agents' motivational attitudes change as a result of actions?
- What inter-attitudinal constraints are required to manage the rational balance needed among an agent's beliefs, goals, plans, commitments, and intentions?

- How should one formalize ability, i.e. under what conditions can one expect an agent to succeed in achieving her intentions?
- What does it means for an agent to behave rationally? How should an agent's future behavior depends on her current mental attitudes?
- How can multiple agents coordinate to achieve a common goal?

In the following, we discuss previous work that attempts to address these questions.

2.1 Informational Attitudes and Action

One concern of BDI logic theories has been to formalize informational attitudes, such as knowledge and belief, and their relationship to action. This relationship can be broken down into two aspects, namely, the informational preconditions of actions, and the effects of actions on agents' information. In BDI logic theories, the informational attitudes of the agents are almost always modeled using accessibility relations on possible worlds [82, 83, 84]. However, along with the associated computational complexity, possible worlds reasoning suffers from a number of other problems, such as *logical omniscience* (i.e. that of knowing all valid formulae, and that of knowledge being closed under logical consequence). To deal with this, various other approaches has been proposed in the literature [47].

Moore [105, 106] was a pioneer to integrate knowledge and action into a single framework. In his formal theory, he accomplished this by formalizing Hintikka's modal logic of knowledge [72] within McCarthy's first order situation calculus [102]. Variants of this model were later proposed in [91, 136, 35]. Others added informational modalities to other logics of action, such as dynamic logic [170] and CTL* [124, 158].

Moore's main concern was to study the problem of knowledge preconditions for actions, that is, the question of what an agent needs to know in order to be able to perform some action. Moore formalized the ability to perform an action using an agent's knowledge of the referent of the action. In his framework, an agent knows the referent of an action, if it denotes the same action in all of the agent's possible worlds. If the agent knows that the action is also executable, then she is able to perform the action.

Moore, and later others [133, 91, 158, 35], also define what it means for an agent to satisfy the knowledge preconditions of *complex actions*. The underlying concept in these accounts is that to know how to do a parameterized action, the agent must know the "procedure" (i.e. the action function), and know the value of the arguments of the action. For instance, an agent can perform the action of dialing the combination of a safe $Safe_1$ (i.e. $dial(combOf(Safe_1))$) if she knows the procedure the $dial$ action refers to, and knows the value of the term $combOf(Safe_1)$. Also, an agent can perform a complex action if she knows that she can execute a sequence of primitive actions that implements the complex action. However, the agent is not required to know in advance the exact sequence of actions she will execute, since the sequence could depend on information the agent gathers along the way. At all time points the agent must know how to execute the current step of the complex action, know that she will eventually complete the execution of the

complex action, and that she will know when the execution is complete. Both Moore's and Singh's theories are limited to determinate complex actions, but Davis, Sardiña et al., and Lespérance et al. handle indeterminate complex actions. Although Moore's framework allows multiple agents, the definition of ability with respect to a complex action involves only action by a single agent. Also, Moore does not address the frame problem for actions, that is, how to specify what remains unchanged after an action is performed. Finally, the framework ignores the formulation of ability to achieve a goal and its relation with being able to perform a complex action. Others, however, do mention how it is a simple matter to add.

A different account was presented in [165], where van der Hoek et al. introduces a primitive capability operator in a propositional modal logic. This operator indicates, for each primitive action and world, whether an agent is capable of performing that action in that world. In this account, the capabilities of complex actions are defined in terms of capabilities of primitive actions. This approach is more flexible than the others, since concepts such as moral capacity can be easily incorporated in this account. However, in this account, one has to specify for each agent, world, and primitive action whether the agent is capable of performing that action in that world. In other words, every instance of a procedure/parameterized action needs to be handled distinctly in this account.

An agent is unable to perform some action when she does not satisfy the informational prerequisites of that action. In that case, it would certainly be useful if the agent had the means to acquire the necessary information. Thus, any agent framework should formalize actions that an agent can use to increase her information. These actions are often known as *informative actions* [106], *test actions* [168], and *knowledge producing actions* [136]. In most agent frameworks, the effect of these actions on agents' knowledge are handled in a similar manner. The result of performing an action that tests the value of a proposition p is that the worlds that disagree with the value of p in the real world are removed from the epistemic accessibility relation. With the situation calculus based formalisms [106, 136, 137], there is an additional requirement that after performing any action a in situation s, the situations that are epistemically related to s are projected/extended by a to get the updated state. In other words, the situations which have not resulted from performing a in an epistemic alternative in s cannot be in the epistemic accessibility relation at $do(a, s)$. So, in this case, even "noninformative actions" affect the knowledge of the agent in the sense that the agent gets to know that it has just performed a. On the other hand, in [168], noninformative actions do not affect the epistemic accessibility relation; also, in their dynamic logic based language, there is no way to say that the agent has just performed action a.

Most of the work already mentioned assumes that new information is consistent with existing knowledge (this is called *belief expansion*). There has also been much work on *belief revision* (and *contraction*) where this may not be the case [61, 169]. Also various researchers have proposed mechanisms for iterated belief change, possibly with noisy sensors [61, 148, 142].

2.2 Motivational Attitudes

Along with agents' informational states, a general theory of agency must also take their motivations into account, since agents are expected to act to achieve their goals. To this end, the general

trend followed in the literature is to specify a primitive motivational attitude, and then to define compound and more useful motivational attitudes using this. There are two main categories of motivational primitives, namely *goal* [26, 124] (variously known as *choice* [129], *wish* [16], and *preference* [170]), and *intention*. While goals are sometimes allowed to be inconsistent and thus difficult to formulate [16], intentions are mostly considered to be consistent. Another difference between these attitudes lies in the agent's level of commitment towards them. Intention is sometimes primitive [124, 81, 68] and sometimes a defined concept, specified in terms of goals [26, 129, 158]. In his philosophical work [16], Bratman argues that intention is different from goals. He identifies the following important properties of intention:

1. Intentions pose problems for agents; they need to determine a way of bringing about their intentions.

2. Intentions provide a filter for adopting new intentions; intentions that are incompatible with an agent's currently held intentions can not be adopted.

3. Agents will maintain an intention if they attempt to achieve it, the attempt fails, but they still believe the intention is achievable.

Previous work on motivational attitudes mostly concern two aspects of these attitudes, namely, when goals are satisfied, and how long should goals persist. The former differentiates *maintenance goals* and *achievement goals*, while the later can be used to specify different levels of commitments to a goal. Maintenance goals are propositions that are currently true, and the agent wants it to remain true. Achievement goals, on the other hand, are propositions that are currently false, which the agent would like to be true eventually. Most of the research in the literature focuses on achievement goals. In the literature, there have been various proposals characterizing different types of persistence of motivational attitudes.

In their linear time temporal model, Cohen and Levesque [26] define a primitive goal modality with its own accessibility relation G similar to the belief modality. Since their model does not have branching futures, it cannot be used to distinguish between some/all branches; rather it can only be used to talk about the actual future. Intuitively, the G-accessible worlds are the ones where all the goals of the agent are satisfied. The goals of the agent are formally defined to be the propositions that are true in all the agent's G-accessible worlds. According to Cohen and Levesque's definition, an agent has a proposition p as an achievement goal if she has the goal that p eventually be true, and believes that it is currently false. Others have adopted a similar primitive motivational operator [124, 129, 170].

Cohen and Levesque point out, following Bratman [16], that since an agent's goals should be compatible with her beliefs, her goal worlds should be constrained to be a subset of the believed worlds. This constraint is known as *realism*. It ensures that the agent does not have an impossible goal. We discuss whether this constraint is actually desirable in the next section.

Cohen and Levesque [26] also investigated the persistence of achievement goals. To ensure that agents do not procrastinate forever, they assume that eventually all achievement goals get dropped. Using this assumption, they then define two types of persistence. According to them, an agent has p as a *persistent goal* if p is one of her achievement goals, and if she does not drop the goal until

either she believes the goal has been achieved, or believes that the goal will never be achieved. Cohen and Levesque acknowledge that agents with persistent goals are fanatically committed to their goals. To remedy this, they therefore define the notion of a *relativized persistent goal*, which is a persistent goal with the further condition that the agent may drop the goal if she comes to know that some proposition q has become false. Typically, the goal p is a means to achieving the super-goal q, or q can be some requester agent's intention. The idea behind this is that the agent adopted the goal relative to the condition q being true, so if q becomes false the agent can freely drop the goal. By Cohen and Levesque's definition, it is not known whether an agent has a persistent goal until the agent drops the goal.

Cohen and Levesque [26] define intention as a special kind of persistent goal. In their framework, they distinguish between an intention to perform an action and an intention to achieve a proposition. An agent intends to perform action a, if she has a persistent goal to perform a, immediately after believing she would perform a. Here, the constraint about the agent's belief is required to prevent the agent from intending to accidentally or unknowingly perform the action. An agent intends to achieve proposition p if she has the persistent goal to perform some sequence of actions a, after which p is true. Moreover, immediately before performing a, it is required that the agent believes that she is about to perform some (possibly different) sequence of actions a_0, after which p holds, and that she does not to have the goal that it not be the case that p is true immediately after doing a. Note that the believed sequence of actions can be different from the actual sequence. The reason for this is that it allows the agent to intend to bring about a state of affairs without knowing in advance exactly how to do it. It is also required that in at least one of the agent's chosen worlds, the agent performs a, i.e., the sequence of actions that the agent really does, after which p holds. This is meant to rule out cases in which the agent is trying to do, say, a_1 to bring about p, but in the course of doing a_1, she accidentally ends up bringing about p in a completely unforeseen way before completing a_1. Both these definitions of intention can be relativized to a condition q by replacing the persistent goal with a relativized persistent goal. Cohen and Levesque [26] show that their definition of intention satisfies the properties of intention given by Bratman [16].

Rao and Georgeff [124] adopt a primitive intention modality in addition to the goal modality. Their formal language is based on a first-order modal logic that is a variant of the Computation Tree Logic (CTL*) framework [44]. This is a branching-time temporal logic, so it can refer to possible or optional futures (i.e. to formulae that hold in at least one branch among the paths emanating from the current situation) and inevitable futures (i.e. formulae that hold in all branches emanating from the current situation), in contrast to Cohen and Levesque's account, where one can only talk about the actual/inevitable future. However, unlike in the latter, there is no way to talk about the actual future, i.e. the path that represents the actual evolution of the world. In their framework, the intention accessible worlds are the worlds that the agent is *committed* to trying to actualize. The intentions of the agent are then defined as the propositions true in all the intention accessible worlds.

Rao and Georgeff [124] studied the persistence of intentions rather than the persistence of goals. Like Cohen and Levesque, they assume that all intentions eventually get dropped. They define three types of commitments, namely, *blind commitment*, *single-minded commitment*, and *open-minded commitment*. An agent is blindly committed to an intention if she maintains her

intention to achieve a goal until she believes that the goal holds. An agent is single-mindedly committed to an intention if she maintains her intention to achieve a goal until she believes the goal is true, or until she doesn't believe the goal might eventually be true. Finally, open-minded commitment is the same as single-minded commitment, except that the agent can drop the intention if she drops the goal that the proposition might eventually be true, instead of dropping that belief.

Sadek [129] revises Cohen and Levesque's account to incorporate a branching time temporal logic. He uses a primitive goal modality called *choice*. One problem with Cohen and Levesque's realism constraint is that there is no way to distinguish between the goals due to realism (i.e. a goal that ϕ provided that the agent believes that ϕ is inevitable) and the goals that the agent actually wants. To distinguish between agents' free choices and choices due to the realism constraint, Sadek [129] introduces the concept of *relevant choice*. According to Sadek, an agent i relevantly chooses that ϕ, if she chooses that ϕ, and that if she does not believe that ϕ is not the case, then she chooses that ϕ (i.e. $RC(i,\phi) \doteq C(i, \phi \wedge (\neg B(i, \neg\phi) \supset C(i,\phi)))$). However, this does not fix the "problem". Sadek's definition of relevant choice along with his KD45 logic of choice implies that any chosen proposition is a relevantly chosen one. Thus Sadek's choice modality seems to be similar to Cohen and Levesque's goal modality.

Sadek uses a weaker definition of achievement goals than Cohen and Levesque. In his branching time temporal model, he only requires that in every chosen world, p will eventually be true in *some* possible future. In contrast, Cohen and Levesque requires p to be eventually true in all possible futures (i.e. in their case, the only, inevitable future) of every chosen world. Sadek [129] modified Cohen and Levesque's definition of a persistent goal, by requiring that the agent *choose* not to drop the goal until either the agent believes that the goal has been achieved, or she believes that the goal will never be achieved. Sadek's definition is meant to allow agents to have some awareness that they were adopting a persistent goal. However, a problem with this definition is that it does not guarantee that the persistent goal will actually persist. This is due to the fact that an agent may have a persistence goal at some time-point, but at a later time-point she might change her preferences and thus drop the goal.

In [129], Sadek presents a definition of an intention to achieve a proposition p, where an agent is allowed to include actions by other agents in her plan to achieve p (as long as the initial actions are done by the agent herself). Unfortunately, this definition has some problems. For example, despite claims to the contrary, the given definition allows the agent to intend p when there is no sequence of actions that the agent believes will bring about p.

Another account of both achievement and maintenance goals was presented in [145, 141] where Shapiro et al. define goal using a knowledge accessibility relation K and a primitive "want" accessibility relation W (or H). Intuitively, the W-accessible worlds for an agent are the *happy worlds* where all her goals are satisfied. They then define goal accessible worlds G as the intersection of W and the knowledge accessible relation K, in the sense that a G-accessible world is a W-accessible world that has a K-accessible world in its history. The reason for imposing this constraint is that this assures that agents goals are realistic, i.e. an agent does not have a goal that she knows is impossible to achieve (this also holds in Cohen and Levesque's framework, as discussed below). Goals are then defined to be all the formulae that are true over the interval $[now, then]$, where now is a K-accessible world, and $then$ is a G-accessible world. Their account is more flexible than

others as it handles both types of goals in a uniform way.

One principle that a logic of intention should satisfy is the *side-effect-free* principle, i.e. that an agent should not necessarily intend all the believed "side-effects" of their intentions. For instance, consider the following example (adopted from [26]): suppose that an agent has the intention to go to the dentist to get her teeth fixed, and also believes that getting her teeth fixed will always involve pain. A normal modal logic[1] of intention along with the realism constraint entails that the agent also have the intention to have pain. Thus, even if after the surgery, she finds out that the procedure didn't cause any pain, she will actively pursue her intention to have pain! This causes problems for many of the normal modal logics of intentions seen so far. We discuss this in detail in the next section. To avoid the problem altogether, various researchers have proposed to use non-normal model logics to model motivational attitudes.

Konolige and Pollack's [81] only motivational operator is a primitive intention operator. The main advantage of their account is that it follows directly from their non-normal modal semantics that intention is not closed under logical consequence and conjunction. Thus it does not suffer from the side-effect problem, unlike Cohen and Levesque and Sadek's account. In their semantics, intentions are associated with a set of *scenarios* $\mathcal{I}$. A scenario is the set of possible worlds that satisfies some sentence in the non-modal sub-language of their language. An interpretation satisfies $Intend(p)$ if there is an $I \in \mathcal{I}$ that is a scenario for p, i.e. p is true in all the worlds in I and every world that satisfies p is in I. Their non-normal semantics of intention is equivalent to the minimal model semantics of Chellas [25]. Unfortunately, in their framework they have no requirements that intentions persist, and thus their intention modality is closer to what others use as the goal modality.

In an attempt to obtain a minimal logic of intention, Herzig and Longin [68] model intention using a primitive modal operator. Their semantics of intention modality is a non-normal one, and thus in their framework intention is neither closed under logical truth, nor under logical consequence, conjunction, and material implication.

Other researchers have incorporated a procedural motivation component in their framework; sometimes these are used to define the agent's intentions, and sometimes to model the agent's commitment towards actions. In his branching future logic, Singh [158] offers such a model of intention. The underlying temporal logic of his framework is a very expressive one, and one can model true concurrent execution of actions (modeled not just as interleaved actions, but parallel execution of actions), and actions with varying durations. Moreover, unlike Rao and Georgeff's model, Singh's interpretations single out a branch that corresponds to the actual future. Thus, his language can talk about what will really happen. Singh also introduces a procedural motivation component, called a *strategy*. A strategy is an abstract plan or program built-up from constructs such as primitive actions, waiting for conditions, sequences, conditional strategies, and conditional loops, and is viewed as a description of what the agent is currently trying to achieve. In this framework, agents are assigned a strategy in each state. This is modeled using a function C^Y that associates a strategy with an agent at a world and a time. Singh defines intentions in terms of

[1] A normal modal logic is one that satisfies the K axiom, i.e. $G(\phi \supset \psi) \supset (G\phi \supset G\psi)$. Modal operators with a classical possible worlds semantics satisfy this axiom.

the strategy the agent is following. An agent is said to intend proposition p if it is a necessary consequence of executing her strategy. Singh uses a strong notion of the persistence of intentions, and stipulates that agents do not change their strategies as long as they are able to continue to follow them.

van Linder et al. [170, 103] use a primitive preference modality called *wish*. Wishes are interpreted as a necessity operator over an accessibility relation W. Thus, agents can have contradictory wishes. They offer a strong definition of achievement goals, where the agent is required to know that there is a sequence of primitive actions that she is able perform and whose execution will achieve the goal, i.e. that the goal is *implementable*. In their framework, achievement goals are known to be currently false. Furthermore, they constrain that a goal must be known to be an *explicit preference* of the agent (defined using the *awareness approach* of [46]). To this end, they introduce a special action called $select$. The result of performing $select\ \phi$ marks the wish ϕ as selected. They then define achievement goals as selected wishes that are unfulfilled and implementable. The advantage of this formalization of goals is that it does not suffer from the side-effect problem, and the *transference* problem, i.e. the problem that all logical tautologies are goals of the agent. This is due to the fact that goals are defined not just as wishes, but explicitly chosen wishes/preferences. van Linder et al. require that agents never drop any of their wishes. However, in their framework, goals are dropped as soon as they are fulfilled or become non-implementable, as required by their definition of goals. Due to this, their goals are really intentions.

To model an agent's commitment, van Linder et al. use a semantic primitive called an *agenda*. They define two meta-actions, *commit(a)* and *uncommit(a)*, that update the agenda of the agent so that the agent becomes committed to and uncommitted from the complex action a. An agent can only be committed to a single complex action at any one time. To commit to a complex action, the agent must be able to perform this action and it must achieve one of her goals; she must also have finished executing her previous commitments (i.e. the agenda must be empty). An agenda is a function that maps an agent and a world to a finite sequence of primitive actions and test actions. This action sequence corresponds to one of the terminating executions of the complex action that the agent is committed to achieving. The result of committing to a complex deterministic action (built from primitive actions, tests, sequences, conditional compositions, and iterations) in a given world w is defined in way such that it updates the agent's agenda by adding the appropriate sequence of primitive and test actions to the agenda not only in w, but also in all the knowledge-accessible worlds in w, and in all the worlds that lie along the execution trajectory of the action (i.e. that can be reached from these worlds by performing some prefix of this sequence). This ensures that commitments are known, and that a commitment to a complex action is linked to commitments to its constituents. On the other hand, an agent can uncommit from a complex action if it is no longer useful in achieving any of her goals. Thus the agenda along with the two meta-actions can be used to model an agent's commitment towards actions that achieve one of her goals. While this account links an agent's declarative intentions with her intended actions, it abstracts from how an agent comes to know that a plan is appropriate for a goal. Also, nothing in the framework prevents an agent from performing something that is not in her agenda.

2.3 Interattitudinal Constraints and Properties

In order to capture some of our intuitions about the intentional attitudes and to prove desirable properties about them, the primitive attitudes discussed in previous sections need to be constrained. One constraint introduced by Cohen and Levesque [26] that received some attention is the *realism constraint*. This is a semantic constraint that says that the goal worlds should be a subset of the believed worlds. The reason for imposing this constraint is that it is unintuitive for a rational agent to adopt an achievement goal that she believes will never hold (this property is also known as *intention-belief inconsistency* [16]). This constraint rules out these states of affairs. Since, each of the agent's goals must be satisfied in at least one of the believed worlds, the agent cannot believe the negation of any of its goals. This constraint is also adopted by Sadek [129] and Konolige and Pollack [81], although Konolige and Pollack constrain the intended worlds (in contrast to the goal worlds, as in Cohen and Levesque's framework) to be a subset of the believed worlds. As mentioned earlier, this constraint also follows from Shapiro et al.'s [145] definition of goal.

Rao and Georgeff [123, 124] introduce two variants of the realism constraint, namely, *weak realism* and *strong realism*. Weak realism constrains the intersection of the believed worlds and the chosen/intended worlds to be non-empty, and is thus weaker than realism. Rao and Georgeff [123] also adopt a similar constraint between chosen worlds and intended worlds. The strong realism constraint [124], that can only be defined for worlds that have branching futures, requires that for every believed world, there is a goal world that is a sub-tree of the believed world with the same truth assignments and accessibility relations at all timepoints. They also adopt a similar constraint between goal worlds and intended worlds. This constraint is stronger than realism in the sense that it restricts the agent to choose only goals that it believes it will always be able to achieve, regardless of what happens in the future. Singh [158] imposes a constraint that ensures that if an agent intends a proposition p, then it does not believe that p will never hold in the real future.

As discussed earlier, another property of intentions proposed by Bratman [16] is that agents need not always intend the believed side-effects of their intentions. There are various forms of this "side-effect-free" principle. The weak version states that an agent should be able to consistently intend that p, believe that p always implies q, and not intend that q. Cohen and Levesque [26] and Singh [158] show that their theories are consistent with this. A stronger version of the principle says that an agent should be able to consistently intend that p, and *always* believe that p always implies q, without intending that q. Cohen and Levesque [26], Rao and Georgeff [123, 124] and Sadek [129] show that their theories are consistent with this version of side-effect-free principle. However, for Cohen and Levesque, this holds for the wrong reason, i.e. because the agent may believe that the side-effect already holds, and thus will not have the persistent goal that the side effect hold, and hence the intention to achieve it. Rao and Georgeff's weak and strong realism constraints are meant to ensure that side-effects need not be intended; these constraints do not rule out the existence of an intention/goal-accessible world that does not agree with the belief-accessible worlds on the truth values of a formula, and thus there may be goal/intention-accessible worlds where $p \supset q$ does not hold. However, their theory is still closed under conjunction. Both van Linder et al.[170] and Konolige and Pollack [81] show that their frameworks are consistent with yet another stronger version of the side-effect-free principle, which says that if p logically

implies q, and the agent prefers or intends p, then the agent may consistently not prefer q.

Rao and Georgeff [123] argue that although agents' goals should be potentially achievable, no rational agent should be forced to choose all her beliefs; they call this property *belief-goal transference*. They point out that Cohen and Levesque's realism constraint sanctions this property. Since the chosen worlds are a subset of the believed worlds, the agent chooses all of its beliefs. Sadek [129] argues that transference is not problematic, if one can differentiate between choices made due to the realism constraint and choices made freely by the agent. Rao and Georgeff [123], on the other hand, argue that transference is irrational, and show that both strong and weak realism avoid transference. One could claim that the realism constraint causes problems in Cohen and Levesque's framework because their logic deals with the actual future, rather than possible futures. For instance, in Shapiro's branching future framework, the realism constraint seems unproblematic. What the realism constraint sanctions in Shapiro is that you can't have the goal that p if you believe that $\neg p$ is inevitable, and this seems unproblematic.

2.4 Success Theorems and Rational Action

One important aspect of an agent theory is the connection between beliefs, goals, and intentions and the agent's action. Theorems that link these attitudes to an agent's future behavior and the achievement of her goals have been called *Success Theorems* [158], since they characterize conditions under which one can expect an agent to succeed in fulfilling her intentions. Success theorems are important for motivating goal delegation via communication. For instance, to plan a complex action that involves delegating sub-goals to other agents, waiting for those sub-goals to hold, and then resume working on the goal, it is necessary for the agent to know that the delegated sub-goals will eventually become true. Using a success theorem, it is adequate for the agent to know about the other agent's intentions and abilities to infer this.

In [26], Cohen and Levesque present a success theorem which states that if an agent has proposition p as a persistent goal, is always competent with respect to p (i.e., whenever the agent believes p, p is true), and it is not the case that the agent will believe p will never occur before she drops p, then eventually p will hold. To relate agents' intentions with their actions, Cohen and Levesque assume that all intentions eventually get dropped. This implies that the agents do not procrastinate indefinitely with respect to their intentions (AKA the *no infinite deferral* assumption). According to the definition of persistent goal, it can only be dropped if the agent believes that the goal has been achieved or is impossible to achieve. The latter case is ruled out by the premise of the theorem, therefore the agent eventually believes p. Moreover, since the agent is competent with respect to p, it follows that p eventually holds. Note that, this theorem does not imply that the agent will eventually act, because some external event might achieve p. But Cohen and Levesque claim that if the agent knows that it is the only one that can act to realize p, then under certain (unstated) circumstances, the agent will act. Nevertheless, this account can be criticized since the no infinite deferral assumption should follow from other axioms of the theory, rather than be imposed separately [78]. Rao and Georgeff [124] also offer similar theorems for their three forms of commitments to intentions.

Singh [155] criticizes these success theorems as being too powerful. In particular, he points out

that they do not take into account the abilities of agents. He requires that agents always perform actions that they know will ensure the eventual success of their strategies. Using this assumption, he showed that if an agent knows how to follow her strategy, and if her strategy necessarily leads to p (and thus she intends that p), then eventually p will hold. Both Shapiro et al. [144] and Khan and Lespérance [78] also prove success theorems in which the agent is only guaranteed to achieve her intentions if she is *able* to achieve them. The latter consider multiple agents, in the sense that agents are allowed to delegate actions to other agents.

One important concept that has largely been left out of agent theories is that of rational choice of action. If agents are not acting rationally, then they cannot be expected to achieve their intentions even if they have the required commitment. Singh's agents are trying to "achieve strategies". His assumption that agents perform actions that they know will achieve their strategies actually ensures that agents act rationally. In their earlier work, Shapiro et al. [144] formalize strategies as functions from situations to actions (called *Action Selection Function (ASF)*). They use the agent's goals to provide an ordering on ASFs for each situation. For a given situation s, an ASF σ_1 is said to be as good as another ASF σ_2 iff σ_1 achieves all of the agent's goals in all the alternative situations where σ_2 does. They then define an ASF σ to be a rational course of action for a given situation s iff it is maximal in the ordering for s. In [135], Sardiña and Shapiro extended this concept of domination of strategies to deal with *prioritized goals*. Building on Shapiro et al.'s model of goals in [145, 141], Khan and Lespérance [78] also use an agent's goals to provide an ordering on complex actions (i.e. plans) for each situation. Plans in their framework can include actions by other agents. They then define a plan to be rational for an agent in a situation iff the plan in maximal in that ordering for that agent in that situation, and the plan is *epistemically and intentionally feasible* for the agent. The additional condition on epistemic feasibility ensures that the agent fulfills the knowledge preconditions required to execute the plan. In case the plan includes actions by others, the agent is also required to know that the executing agent fulfills the knowledge preconditions of the actions delegated on her, and that she intends to execute the appropriate actions in the plan when it is her turn to act. In other words, the plan must be both epistemically and intentionally feasible with respect to the agent. Most work on rational action selection has been done in the decision theory setting, where the outcomes of actions are assigned numerical probabilities and utilities [15, 181, 41]. Boutillier [15] provides a framework for analyzing rational action selection in a logical setting using qualitative orderings for preferences and likelihoods.

2.5 Multiagent Systems: Collective Mental Attitudes, Communication, and Coordination

So far, we have mainly discussed work on modeling individual agents. The study of collective mental attitudes and collaborative action has also received attention. The motivation for such collaborative action is that it is often the case that although no individual member of a group can bring about some goal, collectively they can achieve it if they coordinate and cooperate with each other. Work in this field is variously known as teamwork, cooperative problem solving, team activity, and cooperative plans.

Researchers in multiagent systems have formalized various collective mental attitudes. Common knowledge can roughly be modeled using infinite nesting of a group knowledge (i.e. "ev-

eryone in the group knows that") modality. It is possible to formulate common knowledge as a fixed point formula [67]. Joint intentions are often formalized as a non-primitive construct built using concepts such as mutual belief and intention [139, 29, 30, 65, 66, 76, 77, 95, 160]. Although joint intentions imply individual commitments for the team members, these collective intentions are usually not reducible to summation of individual intentions. Others formalize group intention from an external perspective by utilizing an explicit social structure for a group [152, 157]. Examples of work on joint ability include [65, 94, 114, 115, 154, 166, 184, 188, 190].

When multiple agents are working together to solve a problem, they need some mechanism for exchanging information, such as beliefs, plans, intentions, synchronization information, and so on. Communication plays an important role in the coordination of multiple agents. Agent communication theories are founded on *Speech act theory* [3, 138], which originated in the philosophy of language. The key idea in speech act theory is that communication acts can be viewed as regular actions. While most actions are performed to change the physical state of the world, communication actions are mainly done to alter the hearer's mental states. Speech acts can be categorized into assertives (e.g. informing), directives (e.g. requesting), commissives (e.g. promising), declaratives, permissives, and prohibitives.

The literature on speech act theory is quite extensive. Cohen and Levesque showed that in a BDI framework, many properties of speech act theory can be derived from an independently motivated theory of rational interaction, which is in turn grounded in the rational theory of action [27, 28]. Following this approach, they proposed formal semantics for speech acts. They model speech acts as attempts to bring about some effects by performing some sequence of events, but with the intent to produce at least some result. For instance, a *request* to achieve ϕ is considered as an attempt by the speaker to have the hearer bring about ϕ, with the intent of at least making the hearer believe that it is mutually believed that the speaker has the goal that the hearer brings about ϕ. Singh [156], on the other hand, argued that the semantics of speech acts corresponds to their satisfaction conditions, and identified these conditions for different types of speech acts. For instance, an assertive is satisfied if its propositional content is true at the time of the utterance. A directive is satisfied if its proposition comes to hold at a later point in the future, and the hearer has the know-how and the intention to achieve it. A commissive is almost like directive except that the role of hearer and speaker is switched, and so forth for permissive, prohibitive, and declarative speech acts. Herzig and Longin [68] proposed some cooperative principles, and showed how these rules can be used to infer the effects of a *yes-no* question and that of a *request* from that of an associated assertive. Their model provides a simpler logical account where only assertives are primitive.

In the ARTIMIS rational agent model [130], Sadek formalizes planning for communicative actions using a backward chaining planning mechanism that utilizes the *feasibility preconditions* (FP) and *perlocutionary effects* (PE) of the speech acts. Here, the FP specify the conditions that have to be satisfied in order to plan for the act, and the PE correspond to the rational effects of the action. For instance, consider the action of i informing j that ϕ is true. The preconditions for this inform action is that i should believe that ϕ holds, and i should not know that j knows that ϕ holds. On the other hand, the rational effects of this action are that j will learn that ϕ holds. Sadek argues that rational effects of a communicative act serve as the reason for planning that act, in the sense

that an agent should select an action only if she needs to achieve the rational effect of that action.

One problem with this account is that it fails to specify the conditions under which the rational effects become actual effects; one cannot reason about these conditions. Moreover, the planning mechanism in [130] is incomplete and many rational plans cannot be inferred. Louis [97] recently extended this framework to incorporate a more general model of planning (state space planning by regression and hierarchical planning) and plan adoption. His framework is more complex, and uses defaults (as does Sadek's). The approach supports multiagent plans and has been implemented. But there is no formalization of epistemically feasible plans, and no success theorem. Also, commitment to a plan is modeled using a special predicate rather than using the intention attitude.

Examples of other interesting agent communication theories include [183, 182, 116, 31, 117, 85]. Using these theories, researchers have developed artificial languages for agent communication (e.g. FIPA-ACL [53], KQML [45]), proposed semantics for agent communication protocols [49, 50, 51, 52, 79], and implemented cooperative spoken dialogue systems (e.g. ARTIMIS [131, 132]).

Recently, there have been many proposals for semantics of communication acts based on social commitments [159, 193, 60, 177]. The commitments associated with a conversation would be accessible to an observer and relevant social rules could be enforced. While it is very important to capture and enforce the social aspects of agent communication, i.e. the obligations that go with membership in an agent society, it should be noted that communication cannot be reduced to this public social commitments level [78]. There has also been a suggestion that public social commitment semantics support more efficient reasoning and are more "tractable". However, it has been also pointed out that this is an orthogonal issue [78].

Another way of coordinating agents is to use social laws [5, 151, 109]. These laws are often modeled by incorporating various deontic logic notions [24], such as obligations, prohibitions, and rights into the framework.

3 Agent Architectures

The aim of agent architectures is to shift the emphasis from theory to practice. Thus, researchers in this field are concerned with issues surrounding the construction of computer systems that satisfy the properties specified by agent theorists. In classical planning, the agent is given a model of the actions available and their preconditions and effects on the domain states and a goal, and her job is to find a sequence of actions whose execution brings about this goal. Thus classical planning assumes a static environment. However, real environments tend to be dynamic, that is, they often change in unexpected ways at run-time. They may include exogenous actions (i.e. actions by other agents or natural events) and the world may change during planning. The initial state could change before the agent starts executing the plan. The world might not change as a result of plan execution as expected due to the occurrence of exogenous actions. Thus, a classical planning agent will often not do well in such environments. This is one of the areas where work on agent architecture contributes, by taking into account the resource limitations of the agent. Researchers in this area are concerned with designing agents that may have incomplete information about the current state of the world, are not always able to accurately predict the effects of their actions, can deal with

external interference, and do not have arbitrary time to deliberate.

One important issue addressed by the agent architecture community is the tradeoff between commitment vs. intention reconsideration, i.e. how strongly should an agent be committed to her intentions in a changing environment. Generally, intentions are considered to be persistent, and are only dropped when they are achieved or they become impossible to achieve, as discussed in Section 2.2. However, prior intended plans may be subject to reconsideration or abandonment when the agent's beliefs change in various ways, for instance, when the agent becomes aware of a more attractive way of achieving her goal. But, according to Bratman [17], "...if an agent constantly reconsiders her plans, they will not limit her deliberation in the way they need to for a resource-bounded agent." Nevertheless, if an opportunity with very high utility arises, the agent should take advantage of this by weighing competing alternatives and reconsidering her current intentions. Thus, there exists a tension between the stability of intended plans that is required for practical reasoning, and the revocability inherent in these plans, as these are often formed based on incomplete information.

Another related issue in agent architecture is the tradeoff between reactivity and deliberation. While agents need some mechanism to support goal-directed reasoning and deliberation, they must also be able to react rapidly to unanticipated changes in the environment. Moreover, since they only have incomplete information about their environment, it is not always possible for them to produce a complete plan for a given goal. Rather, information about how to best achieve a goal can often be acquired after executing some initial part of the plan.

Researchers have proposed various agent architectures that differ depending on how the agent's ability to act is realized. Previous work on agent architecture can be classified roughly into four categories, namely, deliberative architectures, reactive architectures, reactive plan execution architectures, and hybrid architectures. In the following we discuss these categories.

3.1 Deliberative Architectures

A deliberative architecture is one that contains an explicitly represented symbolic model of the world (i.e. beliefs, desires, intentions, and actions), and where decisions, such as what actions to perform next, are made via logical reasoning (i.e. planning). Thus deliberative architectures are based on Newell and Simon's [111] *physical-symbol system hypothesis* –they use a physically realizable set of symbols that can be combined to form structures and are capable of running processes that operate on those symbols according to symbolically coded sets of instructions, in order to produce intelligent action. Most innovations in deliberative architecture design have come from the AI planning community. Since deliberative architectures have a planning process as their central component, these architectures can deal with unanticipated goals. However, the disadvantage of a purely deliberative approach lies in the computational complexity of planning: the agent may not be able to find plans in a timely manner. Also, the plans generated by a deliberative architecture often fail in a dynamic environment. Examples of deliberative architectures whose primary component is a planner include the Integrated Planning, Execution, and Monitoring (IPEM) system [1] (based on a sophisticated non-linear planner), and Wood's AUTODRIVE system [186] (a traffic simulation with planning agents).

One particularly interesting class of deliberative architectures is *plan-based* deliberative architecture. The role of commitment to adopted plans or intentions is a critical component in plan-based architectures. Thus, these architectures use adopted plans to limit practical reasoning. The range of reasoning modeled by these frameworks include means-end analysis (planning), choosing between alternative courses of action (decision analysis), checking consistency of plans and beliefs, and revising beliefs and goals in response to external events.

Building on Bratman's philosophical work in [16], Bratman et al. [17] consider adopted partial plans to structure and focus practical reasoning in their (mostly) deliberative architecture IRMA (Intelligent Resource-bounded Machine Architecture). IRMA has four key data structures: a plan library, and explicit representations of beliefs, desires, and intentions. In this architecture, once an agent adopts a plan, she becomes committed to executing this plan. The agent's commitment to a plan implies that she will not reconsider this adopted plan, unless the environment has changed in a relevant way, and reconsidering this plan will result in a reasonable increase in utility. Also, she will not adopt any further intentions that are inconsistent with achieving her adopted plans.

The adopted plans can be both temporally and structurally partial, meaning that these plans schedule actions for some time period, but not for others, and that these plans specify goals to be achieved leaving open the means to achieve these ends. As discussed above, the motivation for this is that often at plan time, the agent only has partial knowledge about the world, and thus it is not always possible to decide on a complete course of action. These adopted intentions limit the agent's deliberation since they focus means-ends reasoning, and they constrain the number of alternative options for actions that are feed to the decision process.

A partial plan needs to be filled out using means-ends reasoning. Thus, these adopted partial plans focus the means-ends reasoning of the agent. Given a partial plan, the means-end reasoning process outputs some options for courses of actions that refine this plan. But not all suggested courses of actions will be consistent with the already adopted plans. Thus, before these suggested courses of actions are supplied to the decision-making process, they need to be passed through a 'compatibility filter'. After filtering out the inconsistent plans, the compatible options are then feed to the decision-maker for further deliberation. Bratman et al. remarked that the compatibility filter must be computationally efficient relative to the deliberation process.

In addition to allowing deliberation, IRMA also attempts to provide some form of reactivity unlike most deliberative architectures. IRMA utilizes an 'opportunity analyzer' and 'filter override mechanism' in an attempt to model reactivity. The opportunity analyzer takes the agent's beliefs and goals as input, and watches for opportunities to satisfy the agent's desires when some change in the environment is detected. While doing this, it ignores the agent's adopted plans. When it detects such an opportunity, it suggests a course of action to fulfill the goal to the compatibility filter. While the compatibility filter's job is to detect and eliminate inconsistent plans, the filter override mechanism can be used to allow some of these inconsistent options to be passed to the decision-maker for deliberation. If the filter override mechanism passes such an option to the decision-maker, it must be the case that at least one of the agent's adopted plans are incompatible with this option. So, in that case, the decision-maker needs to decide to either ignore this incompatible option, or to revise the adopted plans that conflicts with this option. Note that, although this process is able to handle unanticipated changes in the environment, it is not a completely reactive

mechanism, due to the deliberation involved.

3.2 Reactive Architectures

Some architectures that have been proposed are completely reactive in nature. In these, all the deliberation is done in advance and compiled into the architecture itself. In some approaches, the designer is responsible for this compilation (e.g. [20]). Another way of doing this is to use an automatic compilation process [128]. Thus these architectures neither contain a symbolic model of the world nor a reasoner for manipulating rules and finding plans (i.e., they are not knowledge-based). Although such architectures are very efficient and can perform simple tasks quite well, a major problem is that they are ineffective in environments that deviate from those expected by the designer. The primary reason for this is that the behavior of the agents in these frameworks is essentially hardwired. Also, it is often hard to design agents with multiple complex goals in these frameworks.

Perhaps the best known reactive agent architecture is Brooks' subsumption architecture [21, 22, 23]. Brooks proposed that intelligent behavior can be generated without explicit representations of symbols and without explicit abstract reasoning, and that intelligence is an emergent property of certain complex systems. He identifies two key properties of intelligence – 'real' intelligence is situated in the world and not disembodied (such as theorem provers or expert systems), and intelligence arises as a result of a system's interaction with its environment.

To demonstrate the validity of his claims, Brooks built a number of robots using his subsumption architecture. A subsumption architecture consists of a hierarchy of task-achieving behaviors/layers. Each layer in this hierarchy is used to implement a certain goal of the robot. These behaviors compete with each-other to exercise control over the robot. Lower layers are used to encode more primitive kinds of behavior, and have precedence over the layers further up in the hierarchy. Each layer's goal subsumes that of the underlying layers. Each of these layers accesses some of the sensor data and generates actions for the actuators. It should be emphasized that the generation of actions in this system is extremely computationally efficient and does not involve any explicit reasoning, or even pattern matching. A layer can inhibit inputs or overrule outputs of the layers below it. This allows the lowest layers to work like fast-adapting mechanisms, while the higher layers control the main direction to be taken in order to achieve the overall goal. Thus this architecture is capable of reacting quickly to changes in the environment.

3.3 Reactive Plan Execution Architectures

A reactive plan execution architecture is one that includes a user-defined library of hierarchical plans. Each of these plans consists of a trigger condition (i.e. goal), a precondition (i.e. context), and a body. The trigger condition specifies what the plan is good for, that is, what goal can be achieved using the plan. The context condition describes the conditions under which the plan should be considered for execution. Another component of a reactive plan execution architecture is triggering events. An agent in this kind of architecture responds to events from an event-queue by adopting the appropriate plan, and by eventually executing it. In addition to primitive actions,

the body of a plan can contain sub-goals (i.e. events), which may in turn trigger the selection and execution of other plans (sub-plans). Thus, in these architectures, the changes in the environment determine which plans should be executed, and how these plans are decomposed.

The most well known version of this architecture is the Procedural Reasoning System (PRS) [63, 62]. A PRS agent consists of a belief-base, a goal-base, a set of plans, and a set of intention structures. Beliefs in PRS are facts about both the external world and the agent's internal state expressed in first-order logic. Goals are represented as temporal formulae, which include formulae for achieving a property, testing for a condition, waiting for a condition to hold, and preserving/maintaining a condition. These goals are meant to be used in the triggering event part of a plan. Like IRMA, PRS also uses plans to structure reasoning. Plans in PRS are complex structures called *Knowledge Areas* (KA). Each of these KAs is a rule, and consists of an invocation condition that specifies when it is applicable, and a body that describes a set of steps to be achieved. The body of a KA can be viewed as a graph with a single start node and possibly multiple end nodes. Arcs in this graph represent sub-goals. A successful execution of a KA amounts to achieving each of the sub-goals in a path between the start node and one of the end nodes. Intentions of a PRS agent consists of the set of active KA stacks, each of which keeps track of all the subgoals of the original KA.

PRS uses KAs to encode procedural knowledge about the domain. KAs may be activated in a goal-driven fashion, i.e. as a result of acquisition of a new goal, or in a data-driven/reactive fashion, i.e. as a result of some change in the agent's beliefs. These adopted KAs can be used to structure the practical reasoning, since they constitute the entire reasoning process. At each iteration of the PRS interpreter, the set of applicable KAs are determined (by unification) using the agent's beliefs and active goals. Then one or more of these KAs are selected and inserted into the intention structure for execution. Finally, one of the intentions from the (root of the) intention structure is selected, and a step of this intention is executed. This execution can involve an unelaborated subgoal; in that case, this goal is added to the goals of the system. The interpreter then loops to the next iteration, where a new set of applicable KAs are determined based on the perceived changes. Note that, if the selected KA arose as a result of the acquisition of a new goal (called *intrinsic goal*) or a change in belief, then it is inserted into the intention structure as a new intention. On the other hand, if the selected KA was triggered due to the execution of an already existing intention (called *operational goal*), this KA is pushed on top of the KA stack comprising that intention.

PRS uses a special class of KAs (namely, meta-level KAs) to update the beliefs, goals, and intentions of the PRS agent. Meta-level KAs can also be used to control the adoption of lower-level KAs (e.g. in case more than one KA is applicable), create new sub-goals, handle failures, reorder the intentions in the intention structure, etc. Thus KAs are very powerful and can be used to capture procedural domain knowledge as well as decision knowledge.

PRS can be used to guarantee some form of reactivity. In fact, it was shown in [63] that there exists a bound on the 'reaction time' of a PRS agent. Note that, in each iteration, the interpreter checks for applicable KAs and places one or more appropriate KAs in the intention structure. This process uses unification and is able to 'react' in a timely manner. Note however that, although an PRS agent is able to promptly recognize changes in the environment and adopt intentions accordingly, she may take arbitrarily long to 'react to the environment' by executing some action. This is

due to the fact that there is nothing in the framework that ensures that the process of hierarchical plan decomposition will quickly converge to a sequence of a primitive actions. In other words, the execution of a knowledge area may involve a long and possibly even an infinite chain of subgoaling. Thus the term 'react' above (as used by Georgeff et al. [63]) is used in a weak sense, and should be read as 'recognize'. Nevertheless, it should be noted that PRS assumes that the designer is responsible to ensure that plan decomposition completes in a reasonable time. In practice, these architectures generally respond fairly quickly to changes in the environment.

3.4 Hybrid/Cognitive Architectures

There has also been work on architectures that handle the tradeoff between reactivity and deliberation by implementing reactive mechanisms and a deliberation module in two different but interacting layers. Some architectures also include additional layers for plan execution and/or coordination. These architectures are known as layered architectures. A typical layered architecture works as follows: the reactive layer generates potential courses of action in response to time critical events that happen too quickly to be handled by the other layers. It is often implemented using a set of situation-action rules, and thus does not involve complex reasoning. The reactive execution layer (or scheduler) selects precompiled plans to achieve current goals and schedules them for execution. The deliberation layer uses an explicit model of the world and a planner to generate new plans. Finally, the multiagent coordination (A.K.A. the modeling) layer contains models of the cognitive states of other agents in the environment (including human agents). These models are used to manage the dependencies between the activities of the agent and those of other agents (e.g. to identify and resolve goal conflicts). Some of these architectures have control mechanisms to decide which layer controls execution at a given time. Depending on what layers are included in the framework, layered architectures are capable of providing a guaranteed level of responsiveness, performing resource-bounded deliberation to cope with exceptional events, as well as providing the flexibility to adapt ongoing plan as required by changes in the environment. Examples of layered architectures include Touring Machines [48], Inhabited Dynamical Systems [98], and InterRRap [110].

Some proposals were made to capture human-like functionalities and capabilities. Since these architectures model structures for performing a wide variety of cognitive tasks, they are often called cognitive architectures. Examples of cognitive architectures are SOAR [86], Homer [178], and OSCAR [121], to name a few. In addition to ascribing to the agents intentional modalities such as beliefs, goals, and intentions, these architectures often attempt to formalize learning, problem-solving, natural language processing and generation, planning, memory, defeasible reasoning, etc. However, cognitive architectures have not dealt with the main problems faced by researchers in agent architecture, namely, the tradeoff between deliberation and reactivity, and handling resource boundedness.

3.5 Relation to Agent Theories

In [125], Rao and Georgeff attempt to relate their agent theory in [124] to a simplified version of the PRS interpreter. This "abstract" interpreter operates on a (logically) closed and consistent set of beliefs, goals, and intentions. Also, the belief-base is closed w.r.t. an agent's plans, i.e. the agent knows all her plans, and all possible decompositions of her plans are pre-computed. Using these, it generates all the options for action in a single cycle. Then it selects an action for execution, executes this action, and updates the agent's mental states. Rao and Georgeff informally discuss how to constrain various procedures called by this interpreter, and thus implement some of the basic axioms of their theory in this architecture. This includes axioms that relate various mental states (such as belief-goal compatibility), and axioms that model various forms of commitment (e.g. blind commitment). They also have an axiom which is similar to Cohen and Levesque's 'no infinite deferral' assumption (i.e. that all intentions must be eventually dropped). Unfortunately, they do not consider axioms related to agents' abilities required to achieve goals, and axioms that deal with rationality (e.g. that agents should not adopt plans that are very unlikely to achieve a goal). They then present a "practical" interpreter that is similar to the PRS interpreter, in the sense that it operates on a knowledge-base of explicit (and grounded) beliefs and goals that is not closed under logical consequence, and that it computes the decomposition of the adopted plans over an arbitrary number of cycles of the interpreter. At every step of the interpreter, in response to an event, the option generator iterates through the plan library and returns the plans whose invocation condition matches this event and whose context condition follows from the agent's beliefs. The deliberator then utilizes meta-level rules to decide which of these options should be selected. In the next step, one of the intentions is executed. Like in PRS, the execution of an intention may involve triggering of another event, or execution of a primitive action. While Rao and Georgeff acknowledge that this practical interpreter does not obey all the axioms of their agent theory, no suggestions for revising the axioms were given. They did however hint that under certain circumstances, namely when no external events occur during the execution of a goal, the practical interpreter behaves like the abstract one, and satisfies these axioms. However, it is not clear that this is the case. For instance, there is no way of preventing the adoption of a plan that is inconsistent with the agent's adopted intentions, since no lookahead mechanism is incorporated in this practical interpreter. Thus the axiom which states that the agent's intentions should be consistent clearly does not follow from this interpreter.

4 Agent Programming Languages

The beginning of the current interest in agent programming languages (APLs, henceforth) might be attributed to Shoham's proposal of Agent-Oriented Programming (AOP) [149, 150], as a 'new programming paradigm based on a societal view of computation', and as a specialization of object-oriented programming. The key idea of this AOP paradigm is to use mentalistic and intentional notions formalized by agent theorists to design and program agents.

Another front that pushed the concept of agent-oriented programming is Rao and Georgeff's PRS architecture. As we have seen, the key concept in the PRS architecture is that of using events

for selecting hierarchically decomposed plans, and thus avoiding planning from scratch. In the following, we will see that many APLs in the literature are based on a simplified version of the PRS architecture.

Thus, most of the APLs in the literature can be classified into two classes, namely *deductive reasoning* languages and *reactive plan execution* languages. While the former was derived from various agent theories, logics, and calculi, the roots of the later can be traced back to reactive plan execution architectures (viz. PRS and dMARS [80]).

Logic based APLs are usually more expressive and strongly grounded into the underlying logic. The latter means that programs written in these can often be verified easily by theorem proving or model checking. However, this expressiveness and ease of verification usually comes at the cost of computational complexity. Most of these languages also suffer from other significant limitations, such as poor scalability and modularity, and no support for physical distribution of the computation, nor for the integration of external packages and languages. Examples of logic-based APLs include AGENT0 [149, 150] (based on modal and deontic logic), Concurrent METATEM [54, 55] (based on modal and temporal logic), the Golog family [96, 36, 37] (based on the situation calculus), FLUX [162] (based on the fluent calculus), and MINERVA [87, 88] (based on a non-monotonic logic).

In contrast, efficiency and modularity are two areas where the reactive plan execution languages shine. In addition, these languages provide means for encoding control knowledge by using user-defined plan/rule libraries. Nevertheless, most of these PRS-based languages often have limited expressiveness. Examples of reactive plan execution languages include AgentSpeak(L) [122], 3APL [70], and CAN [185].

APLs also differ on how they handle several issues. For instance, some of these incorporate BDI concepts such as beliefs, desires, goals, etc (e.g. AgentSpeak(L), 3APL, PLACA, etc.), while others do not (e.g. Golog and Concurrent METATEM). A few of these languages handle incomplete knowledge and sensing actions (e.g. Golog and FLUX). Some languages allow planning with lookahead; others only allow reactive plan selection (from a user-defined plan library) and execution. Examples of APLs that allow offline planning include Golog and CAN-PLAN [134]. Some of these languages allow modeled exogenous actions (e.g. ConGolog), have constructs to support communication (e.g. PLACA, Jason), support multiple agents (e.g. JACK), or provide a programming logic on top of the associated programming language to specify agent properties, such as liveness and safety properties (e.g. GOAL, Dribble).

Besides these two classes of APLs that we have identified, there has also been work on purely behavior-based or reactive languages [22, 23, 2, 100]. However, these are closer to agent architectures than APLs. Also, researchers have developed various agent-oriented software engineering methodologies (e.g. Prometheus [113] and Gaia [191]) and tools. Surveys of these can be found in [8] and [9].

In the following, we review work on APLs in our two classes. Later in Section 5, we focus on more recent work on agent programming languages that incorporate declarative goals.

4.1 Logic-Based/Deductive Reasoning Languages

AOP, AGENT0, PLACA

Shoham [150] identifies three essential components of an AOP language: a theory for defining the mental state of agents; an interpreted programming language for programming agents, whose semantics must be faithful to the theory; and an 'agentification' process, which wraps components of physical systems into agents. He acknowledges that the agentifier is not necessary for systems designed with AOP in mind. However, he envisions applying the AOP framework even to ordinary devices, such as watches and cameras, for which the agentification process is required.

AOP incorporates a quantified multi-modal logic with direct reference to time-points. The theory contains three modalities, namely, belief, commitment (also referred to as choice or decision), and ability. Commitment is a derived operator, and defined in terms of *obligation to oneself*. An example of a formula of the logic is as follows: $CAN_a^7\ open(safe)^9 \supset B_b^7\ CAN_a^7\ open(safe)^9$. This says that if at time 7, agent a can ensure that she is able to open a safe at time 9, then at time 7, agent b believes this. Unfortunately, AOP does not include a formal semantics for this modal logic.

Shoham's first attempt at an AOP language resulted in the AGENT0 programming language [150]. In AGENT0, an agent is specified in terms of a set of initial beliefs and commitments, a set of (fixed) capabilities, and a set of *commitment rules*. The set of commitment rules is a key component in AGENT0, and it determines how the agent acts. A commitment rule associates a message condition and a mental condition with an action. If the message conditions match an incoming message and the mental conditions are true in the agent's current mental state, then the corresponding commitment rule fires, and as a result, the agent becomes committed to the associated action. Commitment to an action in AGENT0 amounts to no more than scheduling an action. The AGENT0 interpreter maintains a database of committed to actions and their scheduled times, and when the appropriate time arrives, the action is executed. Along with *private* actions (i.e. internally executed subroutines), AGENT0 also provides *communicative* actions in the spirit of speech-act theory [3, 138, 31]. The basic loop of the AGENT0 interpreter consists of two steps: in the first step, it reads the current incoming message and updates the mental state (i.e. the agent's beliefs and commitments) by applying *all* applicable commitment rules; in step 2, it executes the commitments for the current time, possibly further updating the mental state.

In her 1993 Doctoral thesis [163], Thomas introduces the PLAnning Communicating Agents (PLACA) language as a more refined implementation of AGENT0. PLACA addresses a severe drawback to AGENT0, namely the inability of agents to plan and to communicate their *declarative* goals (via requests). The overall structure of PLACA is very similar to that of AGENT0. To handle planning, the initial mental state now also contains a *declarative* motivational intention-base, and a *procedural* plan-base. Also, commitment rules are now replaced with *mental-change rules*, each of which associates a set of mental state changes with a set of message conditions/mental conditions/(outgoing) message-list. At every tick of the global clock, these mental change rules are used to update the agent's declarative intentions. Plans on the other hand are fed to the system using an external planner, described as a black box in [164], which is responsible for updating

the plan-base at every tick of the clock. For longer deliberations, the planner may request the 'mental-rule checker' module to skip some cycles (while queuing the incoming messages) and allow uninterrupted planning. The architecture also utilizes a separate executor module that is responsible for sending outgoing messages, and for executing the scheduled actions when the time has come.[2] Thus, PLACA separates deliberation about which intentions to adopt from considerations of means of achieving the adopted intentions. While the agent program is used to formalize the former, the latter is modeled using a black-box external planner and not properly fleshed out (i.e. mostly unspecified) by the framework.

In [161], Tan and Weihmayer discuss an AOP-based framework for cooperative problem solving that integrates AGENT0 and the state-space planner PRODIGY [104]. The major difference between PLACA and this framework is that in the latter, planning occurs directly as a result of the firing of commitment rules, and thus is not interruptible. Therefore, planning in this account behaves like a single primitive action. Also, since many rules may fire during a cycle, several planning processes may be triggered, which is computationally demanding and may hamper reactivity.

All of these languages were only intended as prototypes. Thus, various simplifying assumptions were incorporated. For example, AGENT0 lacks a formal semantics. Also, agents can only commit to primitive actions that can be directly executed. AGENT0 and PLACA both assume a global clock. Although PLACA includes declarative goals that trigger planning, it does not formally specify how plan generation and commitment to plans are handled.

Concurrent METATEM and its BDI-Extensions

A problem with AGENT0 and PLACA is that no formal semantics for agent program execution is provided. Also, the execution of these languages cannot be said to truly execute the associated logic. A desirable property of any APL semantics is that it should be strongly coupled to the underlying (BDI) logic. In other words, the program execution semantics should satisfy the underlying logic. This ensures that these two are compatible with each-other; for instance, if the underlying theory sanctions that an agent is (physically and mentally) able to perform some action in some situation, then it is only intuitive that the APL execution semantics agrees with this and that one could derive that there is a legal transition of the agent program with this action in that situation. The Concurrent METATEM language proposed by Fisher and Barringer [56], and its extensions [54, 55] attempt to address these issues.

A Concurrent METATEM system is a collection of concurrently executing objects, whose behavior is specified directly using an executable temporal logic. These objects can communicate via asynchronous broadcast message passing. Each object is specified using an object-interface and a set of executable temporal rules. An object-interface identifies the messages that an object can recognize, together with the messages that it can produce. The temporal rules associated with the objects form the bulk of a Concurrent METATEM program. These have the following general form: 'past and present formula' $\supset$ 'present and future formula', and are assumed to hold at all time-points. In Concurrent METATEM, an object's specification is directly executed to generate

[2] Recall that every action is dated, i.e. has a time-stamp associated with it.

its behavior. The execution of an object involves iteratively constructing a model from the corresponding temporal formulae, in the presence of input from the program's environment. Starting from the initial state, at each step the program rules are consulted to check which of the rules have antecedents that are satisfied by the partial model constructed so far. The consequents of all such rules are collected together. These consequents represent constraints on present and future properties of this model, and these along with any outstanding constraints generated in some previous steps are used to construct the current state. Any outstanding constraints are passed to the next step. If at any point, a contradiction is generated, the system may backtrack (i.e. undo some actions) to a previous choice-point and attempt to construct a model for the program in a different way, giving up indicating an execution failure when no choice-points are remaining. Note that, sources of non-determinism (i.e. choice-points) include the execution of a rule whose consequent contains a disjunction or an $\Diamond$-formula, i.e. the execution of eventual satisfaction of some formula.

To summarize, Concurrent METATEM differs from other languages in that it based on a satisfiability point of view. As mentioned, an advantage of Concurrent METATEM is that in this framework, the theory and the programming language are strongly coupled. Also, the semantics of program execution is closely related to the semantics of the associated temporal logic. Thus, verifying agents' properties specified in the language is a viable proposition. In [55], Fisher uses a series of examples to demonstrate that Concurrent METATEM can be used to specify intelligent objects that exhibit an interesting range of behaviors, which include cooperation and competition, negotiation, obeying safety and liveness constraints, and so forth. Also it can be used to specify groups of objects (societies) and hierarchical problem solving objects. Unfortunately, Concurrent METATEM has some limitations. For instance, recall that building a model may require backtracking; it seems that this is not always possible, e.g., in some domains, the effects of actions may not be reversible. Also, like satisfiability, it requires complete knowledge.

Recently, Fisher and Ghidini [57] extended Concurrent METATEM by incorporating the notions of belief, ability, and the "motivational" operator *confidence*. Agents' beliefs are represented using a (KD45) *multi-context logic* [64]. Multi-context logic is a formal framework for modular representation of (nested) beliefs of multiple agents, and is based on the notion of belief contexts. A belief context is a representation of a collection of beliefs that an agent ascribes to herself and to other agents. For example, and agent i may have some beliefs about the world; in addition she may have some beliefs about another agent j, beliefs about j's beliefs about another agent k, etc. In a multi-context logic, each of these sets of beliefs is represented using a distinct formal language, and the interpretation of such a language is local to the belief context it is associated with. A context structure in a multi-context logic contains an infinite tree (where the root of the tree represents the belief context of the agent whose belief is under consideration), and allows one to represent arbitrarily nested beliefs. Although distinct, the contents of different belief contexts can be related. For instance, an obvious relation is the following: if a sentence of the form P is in the belief context for i's beliefs about j (i.e. in the context ij), then a sentence of the form "j believes that P" is in the belief context for i. Another relation says that a sentence of the form P is in ij, only if a sentence of the form "j believes that P" is in the context i. Depending on the relations among different contexts, one can model agents having different reasoning capabilities. As we can see from the above description the key feature of belief-contexts is modularity. Note that, the

extended Concurrent METATEM incorporates the appropriate relations so that agents' beliefs are KD45.

An agent's abilities are constant over time. The semantics of the ability operator is formalized using a function r that associates a belief context to a (fixed) set of formulae. Thus, e.g. if r associates $\{\phi\}$ with the context for agent i, then i has the ability that ϕ; on the other hand if r associates $\{\phi, \psi\}$ with the context for agent k's belief about agent j's belief about agent i, then k believes that j beliefs that i has the ability to achieve ϕ and ψ. Confidence in ϕ, which is a derived attitude, is defined as *believing that ϕ will eventually happen* (i.e. $B_i \Diamond \phi$). As in the original Concurrent METATEM, agents are specified using an agent-interface and a set of temporal rules in this framework. However, rules are now allowed to have modal operators; the rules must be in normal forms that only allow present and future temporal operators.

The key idea in this framework is that the language provides a mechanism for deriving concrete specification of motivations from more abstract ones. Consider the interaction between the following temporal goal formulae (in descending order in terms of abstractness): $B_i \Diamond \phi$, $B_i \Diamond_j \phi$, $\Diamond \phi$, and ϕ, that is, the agent i is confident about ϕ, i is confident about ϕ and believes that j is responsible for bringing about ϕ, ϕ will eventually hold, and ϕ is currently true, respectively. Fisher and Ghidini argue that by providing rules that can be used to derive a more concrete goal formula from one of these abstract goal formulae, we are essentially specifying a *rational* agent. One such rule is as follows:

$$(B_i \Diamond \phi \wedge A_i \phi) \supset \Diamond \phi.$$

This says that, if agent i is confident about ϕ, and is able to achieve ϕ, then ϕ eventually holds. Another example of this is that an agent may move from 'confidence' (i.e. $B_i \Diamond \phi$) to 'confidence in another agent' (i.e. $B_i \Diamond_j \phi$, where $i \neq j$), through deduction or communication. Again, moving from $\Diamond \phi$ to ϕ is essentially a matter of scheduling.

In this framework, various rules can be tailored and various constraints on B_i can be imposed to specify realism, strong-realism, and weak-realism properties (see Section 2). Also, rules can be used to implement sensing actions. Thus, this extension of Concurrent METATEM brings it closer to other BDI languages. Even more recently, Fisher et al. [59, 58] extended Concurrent METATEM to include groups of agents and show how agents can be efficiently organized to collectively solve problems.

One problem with these frameworks is that in these languages, the programmer needs to explicitly specify the behavior of the agents using temporal rules. Thus although verification of agent properties is relatively straightforward, programming even simple agents puts a heavy burden on the programmer. The examples in [55] and [57] show this. Also, while it is possible to write chaining rules (i.e. rules whose consequent fires other rules) in this language, these rules do not exactly correspond to hierarchically decomposed procedures/plans. Finally, this model of agent programming is problematic in the sense that although specifying rational actions (or in this case, rational temporal rules) is left to the programmer, there is nothing to prevent the programmer from writing inconsistent or non-terminating sets of rules.

Golog, ConGolog, and IndiGolog

Another style of agent programming is developed in the logic programming-based Golog family [96, 36, 37]. In Golog, the programmer first declaratively specifies the agent's knowledge of the dynamics of the world (i.e. preconditions and effects of actions), and the initial state of the world in a situation calculus dialect, a first-order language for dynamic domains which incorporates a solution to the frame problem due to Reiter [126, 127]. Then various Golog constructs, such as primitive actions, testing for a condition, sequences, non-deterministic branch, loops, etc. are used to write programs. Given a program and the domain axioms, the interpreter attempts to prove that the program has a terminating execution starting in the initial situation. A sequence of actions for executing the program is uniquely identified by the terminating situation. Once an action sequence is found, the agent executes the program, by executing one action at a time. Thus, Golog redefines the planning problem of 'looking for a legal sequence of actions to achieve some goal' as the problem of 'searching for a legal sequence of actions that amount to a legal execution of the high-level program'. The program can encode search control knowledge.

In contrast to most other logic-based APLs where the agent's state must be explicitly updated by the executing program, Golog and its successors employ an automatic state update mechanism using their background action theory for the domain. Also, unlike other APLs where the designer specifies the agent's behavior using some form of rules, Golog has the programmer specify a high-level non-deterministic program, and the underlying interpreter's task is to search for an execution of this program.

ConGolog extends Golog to include concurrency by providing constructs for non-deterministic iteration, concurrency (with and without priority), and interrupts, which makes it easier to write reactive programs. ConGolog also replaces the *evaluation semantics* of Golog with a *transition semantics*, since a single-step semantics is better suited for concurrency.

The ConGolog interpreter proves that some branch of the non-deterministic program yields a terminating state of the program, and thus resolves the non-determinism in an off-line style using lookahead planning. This offline planning cannot handle dynamically changing worlds too well, especially when sensing and exogenous actions are involved. For instance, consider the following program: $(a|b); sense_q;$ if q then Δ_1 else Δ_2 endIf$; \phi?$, which says that the agent should first non-deterministically choose between a and b and execute it, then sense the truth-value of q, and based on this value, should execute either Δ_1 or Δ_2, terminating successfully if the test $\phi?$ succeeds. Note that, an offline interpreter for this program cannot commit to either a or b in advance, since it does not know which of these will ensure that ϕ, and thus cannot use the sensing action to determine whether q would hold after the action. The only option available to the interpreter is to check if one of the actions a or b will lead to ϕ for *both* values of q. Thus, early occurrences of non-deterministic choices can result in unacceptable delay. The situation gets even worse when loops are involved. To deal with this, the language IndiGolog [37] was proposed. In IndiGolog, programs are executed incrementally to allow for interleaved action, planning, sensing, and exogenous events. Informally, the semantics of incremental execution is as follows: an incremental execution of a program finds a next possible action, executes it in the real world, obtains the sensing results afterwards, and repeats this cycle until the program is finished. Since this makes it possible to quickly execute the

actions without much deliberation, this approach is suitable for realistic changing worlds. However, since the program may contain non-deterministic choice points, some lookahead mechanism is required to avoid unsuccessful (dead end) executions. For this reason, a search operator Σ is introduced in IndiGolog. Intuitively, $\Sigma(prog)$ selects from all possible transitions of $prog$ one for which there exists a sequence of further transitions that leads to a terminating configuration. The IndiGolog interpreter automatically monitors the execution of a plan generated by such a search block, and re-plans when the current plan fails or is no longer appropriate due to changes in the environment. IndiGolog also supports a simple form of contingent planning, where the dynamic environment is modeled as a simple deterministic reactive program [92].

Thus, IndiGolog is a powerful language that is able to handle incomplete knowledge, sensing, and exogenous actions, and allows the specification of prompt reactive behavior as well as user-controlled deliberation. Note however that, the standard implementation of IndiGolog makes a *dynamic closed-world assumption*, i.e. it assumes that a program has sufficient knowledge to evaluate a query/test by the time it is evaluated, and if initially the answer to the query is not known, sensing actions will be executed before the query is made. Thus it is assumed that the on-line interpreter has complete knowledge of the relevant fluents by the time the query is evaluated. To avoid this limitation, an extension of the Golog formula evaluator was presented in [176, 175], where the evaluator keeps track of the *possible values* that functional fluents can take in a given history (i.e. a situation along with the sensing results obtained so far). A fluent is said to be known at some history h iff it has only one possible value at h. Note that this only handles limited forms of incomplete knowledge, namely, not knowing the value of a fluent; general disjunctive knowledge is not handled. This work is still at an early stage, and the issue of how to deal with efficient belief update in the presence of incomplete knowledge is still not completely clear.

FLUX

One disadvantage of IndiGolog and some other logic-based non-BDI languages is that the knowledge of the current state is represented indirectly using histories, i.e. via the initial conditions and the actions that the agent has performed so far. A consequence of this is that each time the agent needs to evaluate a condition, she has to consider the entire history of actions and perform regression. Thus these languages do not handle long running agents efficiently. The fluent calculus-based high-level programming language FLUX (FLUent eXecutor) [162] attempts to solve this problem using an explicit state representation and progressing it when an action is performed. FLUX incorporates an implementation of the fluent calculus, a language for reasoning about actions. The fluent calculus provides a basic solution to the frame problem using the concept of state update axioms. It also addresses a variety of other aspects such as, ramifications, qualifications, nondeterministic and concurrent actions, continuous change, and noisy sensors and effectors.

A FLUX agent program consists of three parts, namely a background theory that encodes the agent's internal model of the environment, a kernel that provides the agent with cognitive abilities to reason about her actions and acquired sensor data, and a strategy that specifies the agent's task oriented behavior. The types of incomplete knowledge FLUX can encode are restricted. The underlying inference engine of FLUX is sound but incomplete. However, it can be shown that

reasoning in FLUX is linear in the size of the internal state representation. Thus FLUX scales up well to long-term control. FLUX allows the use of full expressive power of logic programming in defining strategies. It also facilitates formal proofs of correctness of strategies.

MINERVA

The MINERVA agent programming language [87, 88] utilizes logic programming and several non-monotonic knowledge representation and reasoning mechanisms to provide a common multiagent framework. A MINERVA agent consists of several specialized concurrently running sub-agents performing various tasks. These agents can read and manipulate a common knowledge base specified in the Multi-dimensional Dynamic Logic Programming language (MDLP). In MINERVA, agents are driven by an observation-deliberation-action cycle. The behavior of these agents is specified in the Knowledge and Behavior Update Language (KABUL).

MDLP provides an extension of Answer Set Programming (ASP). In MDLP, an agent's knowledge is represented using logic programs arranged in an acyclic digraph, in which the vertices are sets of logic programs, and edges represent the relationship between these programs. MDLP benefits from the advantages of ASP, such as default negation, which can be used to deal with incomplete knowledge. Also, it can be used to represent the evolution of knowledge, and preferences.

KABUL is a logic programming language that can be used to specify updates to knowledge bases, and to the KABUL program itself. A KABUL program consists of a set of condition-action rules that encode the agent's behavior. Since actions in KABUL can update both the knowledge base represented in MDLP and the KABUL program itself, it can be used to specify agents that change their behavior over time. Conditions in these rules can refer to external observations, the epistemic state of the agent, as well as to occurrences of exogenous actions. Since no external stimuli are needed to trigger the behavior of the agent, KABUL can be used to specify both reactive and proactive behavior.

Other Logic-Based APLs

There has been work on various other logic-based APLs. Those that gained some popularity include APRIL [101], the deontic logic-based IMPACT [42, 43], the dynamic logic-based Dylog [4], the linear logic-based ε_{hhf} [38], the ambient calculus-based CLAIM [140], the Horn Clauses and Least Herbrand Models based DALI [32], and ReSpecT [112]. Surveys of some of these languages can be found in [99, 8, 9]. In the next section, we discuss some of the reactive plan execution languages.

4.2 Reactive Plan Execution Languages

AgentSpeak and its Variants and Implementations (Jason, JACK, Jadex)

As discussed in Section 2, there has been much work on agent theories, and current theories are quite mature and well established. However, there is a large gap between agent theories and BDI APLs.[3] In [122], Rao introduced the AgentSpeak(L) language as an attempt to show a one-to-one correspondence between the model theory, proof theory, and the abstract interpreter of this language. Here, model theory, proof theory, and abstract interpreter refers to the underlying BDI theory, the formal semantics of the programming language (often specified using a transition semantics, as discussed below), and the implemented interpreter for the language, respectively. Rao argues that there is a better chance of unifying theory and practice by taking a simple specification language as the execution model of an agent, and then ascribing mental attitudes to this agent. To this end, he used the Procedural Reasoning System (PRS) and its more recent incarnation the distributed Multi-Agent Reasoning System (dMARS) [80] as a starting point for the AgentSpeak(L) implemented system. To establish the link between agent theories and APLs, it is necessary to have a way of deriving the formal semantics of program execution from the underlying agent logic. To do this, Rao first defined program states of an APL using agent configurations. An agent configuration consists of a sentential description of an agent's beliefs and her motivational states, derived from associated components of the underlying agent theory. Intentions in a configuration is represented procedurally as in PRS. He then defined program execution or agent behavior as transitions from configuration to configuration. These transitions are guided by a set of transition rules (A.K.A. proof rules), which specify how an agent configuration and its components may change as a result of executing an action, and what actions can be executed. AgentSpeak(L) is based on reactive plan execution architectures, where rather than deliberating on which action to execute next, the agents utilize the changes in the environment to decide which given hierarchical plan should be adopted, and how to decompose and execute this hierarchical plan.

An AgentSpeak(L) agent consists of a belief-base, a set of plans, and a set of intentions. When an agent acquires a new goal, or notices a change in her environment, she may trigger additions or deletions to her goals or beliefs. These events are referred as *triggering events*. Agents in this framework respond to triggering events. Plans in AgentSpeak(L) are rules of the form: $(e : cc \leftarrow p)$. Intuitively, this says that in response to the event e, the agent should adopt the plan-body/intention that p, provided that the context condition cc follows from her belief. The plan-body can be built using sequences of goals and actions. Goals in AgentSpeak(L) are of two types, namely *achievement* goals, and *test* goals. Achievement goals are an abstraction mechanism, and serve the same purpose as procedure calls in imperative programming. In other words, the execution of an achievement goal triggers an event, and as a result, the agent adopts the appropriate plan as her intention, just as the execution of a procedure in imperative programming amounts to the execution of the procedure body. These plans may in turn include achievement goals in them, and

[3]Here, BDI APLs refers to APLs that incorporate concepts such as beliefs, desires, goals, and intentions. Also, note that the original proposal of Concurrent METATEM, the Golog family, and many other declarative APLs are tightly coupled to the associated logic/theory. However, most of these languages are not *per se* typical BDI-languages.

in that case when executed, they will trigger the adoption of other plans. Thus achievement goals and plans together provide a mechanism for event-invoked hierarchical decomposition of goals. AgentSpeak(L) also uses these plans to revise agents' beliefs and goals by generating primitive addition/deletion events. Test goals involve testing the belief-base and may be used to compute variable bindings. Intentions in AgentSpeak(L) are stacks of partially instantiated plans. At any time, the agent may have multiple intentions. Initially, each of these intention stacks contain only one element, namely, the plan that was adopted in response to an *external event* (i.e. due to a change in the external environment). The execution of these intentions may involve executing an achievement goal, which triggers the adoption of new intentions. In that case, this new intention is pushed on top of the intention stack that triggered the adoption of this intention.

The overall control flow of the system is determined by the AgentSpeak(L) interpreter, and goes as follows. The interpreter uses a given selection function $S_{\mathcal{E}}$ to determine which pending event to process next. Then it computes the *relevant plans* by checking the plans whose associated event matches (i.e. can be unified) with this event. From these plans, it then computes the set of *applicable plans* by checking whether an instance of the context condition follows from the agent's beliefs. Another selection function $S_{\mathcal{O}}$ is used by the interpreter to choose one of the applicable plans, and this plan is then added to the intention base. The interpreter uses a third selection function $S_{\mathcal{I}}$ to decide which of these intentions should be executed next. As discussed above, these adopted intentions can in turn post so called internal events. Internal events are processed similarly to regular or external events. However, rather than adding the selected applicable plan to the intention base, it is now pushed on top of the intention-stack that posted this event. Only the plans that are on top of an intention stack are considered for execution, and only one of them are executed in each cycle.

For instance, suppose that an agent has the following plans in her plan-base:

$$+!\phi_1 : true \leftarrow !\psi_a; a_1; a_2,$$
$$+!\psi_a : true \leftarrow a_3; a_4.$$

Here $+$ and $!\phi$ refers to an addition event and an achievement goal ϕ, respectively. The first plan says that in response to the event where the agent acquires the goal to achieve ϕ_1, she should adopt the plan to achieve the goal ψ_a first, and then execute the primitive action a_1 followed by a_2. Similarly, the second plan says that in response to the event where the agent acquires the goal to achieve ψ_a, she should adopt the plan to execute the primitive action a_3 followed by a_4. For simplicity, we assume that both of these rules have a $true$ context condition. Now, suppose that the agent acquires the goal to achieve ϕ_1 through some external event (such as, via a $request$ action), and that $S_{\mathcal{E}}$ chooses to process this event next. Since the first rule's trigger condition matches (unifies) with this event, the context condition trivially follows, and there is only one applicable rule (i.e. $S_{\mathcal{O}}$ returns this plan), she will adopt this plan as her intention. Thus, a new intention $[+!\phi_1 : true \leftarrow !\psi_a; a_1; a_2]$ will be added to her intention base (lets call this intention i_1). Similarly, each time she acquires an intention due to an external event, a new intention will be added to the intention-base. Now, suppose that $S_{\mathcal{I}}$ chooses to execute i_1. Since executing the first action of i_1 involves executing the achievement goal ψ_a, it will generate the event that $+!\psi_a$, and this event will be added to the event queue. In the next cycle, suppose that $S_{\mathcal{E}}$ chooses to process the event

$+!\psi_a$. In response to this event and after unifying the trigger condition and verifying the context condition, the agent will adopt the intention that $[+!\psi_a : true \leftarrow a_3; a_4]$, since there is only one applicable rule. But, since this event was generated due to the execution of another intention, namely p_1, it will be pushed on top of the stack for p_1 rather than being added as a new intention. Recall that, when deciding on which intention to execute next, only the plans that are on top of the intention stack are considered. This ensures that the agent will execute $a_3; a_4$ before executing $a_1; a_2$.

An AgentSpeak(L) agent is specified by a tuple $\langle E, B, P, I, A, S_{\mathcal{E}}, S_{\mathcal{O}}, S_{\mathcal{I}} \rangle$, where E is a set of possible events, B is a set of possible base beliefs (defined using a ground set of atoms), P is a set of possible plans, I is a set of possible intentions, A is a set of actions (denoting the possible set of actions that the agent has performed so far), and $S_{\mathcal{E}}, S_{\mathcal{O}}$, and $S_{\mathcal{I}}$ are the three selection functions. An AgentSpeak(L) agent can have a number of executions defined in terms of the configurations reachable from the initial configuration. A BDI configuration is a tuple of $\langle E_i, B_i, I_i, A_i, i \rangle$, where $E_i \subseteq E, B_i \subseteq B, I_i \subseteq I, A_i \subseteq A$, and i is the label of the configuration. Note that, an agent's plans are considered to be static, and thus are not included in a configuration. The semantics of the AgentSpeak(L) programming language is defined using a labeled BDI transition system that specifies how agents can evolve from one configuration to another. A BDI transition system is a pair $\langle \Gamma, \vdash \rangle$, where Γ is a set of BDI configurations, and $\vdash$ is a binary transition relation $\Gamma \times \Gamma$ defined using a set of proof (transition) rules. A BDI derivation/execution is defined to be a (possibly infinite) sequence of BDI configurations $\gamma_0, \gamma_1, \cdots, \gamma_i, \cdots$ such that for all i, $\gamma_i \in \Gamma$, γ_0 is the initial configuration, and for any consecutive pair of configurations (γ_j, γ_{j+1}), $\gamma_j \vdash \gamma_{j+1}$. For the AgentSpeak(L) programming logic, the notion of 'refutation' is defined in terms of an intention: it starts when the agent adopts an intention, and ends when her intention stack becomes empty. Rao argued that this programming logic can be used to formally prove certain properties about an agent's behavior, such as safety and liveness of the agent system. Also, there is an one-to-one correspondence between the AgentSpeak(L) interpreter and AgentSpeak(L) transition rules.

Since Rao's original proposal [122], other researchers have proposed various extensions and implementations of the AgentSpeak(L) language. In [40], d'Inverno and Luck use the **Z** specification language to formally specify a complete abstract interpreter for AgentSpeak(L) similar to that given to dMARS [39]. In [107], the operational semantics of AgentSpeak(L) is specified using a more standard Plotkin-style structural approach [118]. The three major implementations of AgentSpeak(L) include JACK [74], Jason [12, 9], and Jadex [18, 120]. In the following, we briefly discuss these, and also point out the extensions provided by these implementations.

The Java Agent Compiler and Kernel (JACK) Intelligent Agents™ framework [74] is a commercial agent-oriented programming tool developed by Agent-Oriented Software Pty. Ltd. Unlike Jason and Jadex, JACK does not provide a logic-based language to specify agents' beliefs and intentions; rather, it uses an extension of Java to implement some features of the underlying logic, such as logical variables, events, beliefs, and plans. One of the design goals of JACK was to provide developers with a robust, stable, light-weight product that can be used to develop components of larger environments, such as legacy software systems. To this end, JACK provides three extensions of Java. It extends the Java syntax to include BDI-related keywords, declaration of attributes, and statements. It provides a compiler that compiles these BDI syntactic additions into pure Java

classes that can be used by other ordinary Java code. Finally, it incorporates a set of kernel classes that provide the required runtime support to the generated code.

From a functionality point of view, JACK incorporates six components, namely, agent, capability, belief-set, view, event, and plan. The agent construct defines the behavior of an agent by including the capabilities an agent has, the types of events she responds to, and the plans she uses to achieve her goals. The capability concept structures the reasoning capabilities of agents into clusters. Capabilities are built up from events, beliefs, plans, Java code, etc. They simplify agent design by allowing code reuse and encapsulate agent functionality. This allows the agent architect to build up a library of capabilities over time, and create an agent promptly by simply plugging in the required capabilities. JACK uses a generic relational model to represent the agents' beliefs. A belief-set consists of relations of the following form: $relationName(key_1, key_2, \cdots, data_1, data_2, \cdots)$. Each relation can be identified by the relation name and any number of keys. The data fields are used to encode the attributes of a relation. Elements of a belief-set can be retrieved using unification as in Prolog. The view construct is central to JACK's data modeling capability. It is built up from multiple belief-sets or arbitrary Java data structures, and allows general purpose queries to be made about an underlying data model. Events and plans are similar to AgentSpeak(L).

On top of these architecture independent facilities, JACK provides a set of plug-in components that address the requirements for specific agent architectures. Currently it supports two such plug-ins, namely a PRS/dMARS-based BDI interpreter, and a plug-in for building teams of agents, called *SimpleTeam*. The underlying concept behind SimpleTeam is that it allows the programmer to specify a high-level view of the coordinated behavior of a team, and then map this high-level view to the individual activities of the participating agents. JACK also include a graphical agent development environment, a debugger, and an object modeling toolkit to support object transport and interaction with existing applications in Java and C++. JACK has been used to develop commercial applications, such as decision support and defense simulation for analysis applications.

Recently, Bordini and Hübner developed a Java-based open source practical interpreter for AgentSpeak(L) called Jason [12, 9], that incorporates an extension of the AgentSpeak(L) semantics to support speech-act based inter-agent communication [108]. To allow for both closed-world and open-world belief-bases, Jason also allows the use of strong negation in beliefs and plans of agents. Jason also has provision for handling plan failure. This is done by generating a "deletion of goal" event when some action fails (or when some sub-goal fails as a result of the absence of an applicable plan for achieving that sub-goal), and then handling that goal deletion event by searching for an applicable plan in the rule library, eventually executing one of these plans. However, for this to work properly, the user must define appropriate responses to various plan failures. In case no such plans are defined, the Jason interpreter just drops the intention altogether. In Jason, it is possible to design plan failure handling rules in a way so that plan failure is propagated up the intention stack. To do this, Jason provides some special actions called *internal* actions. Internal actions only change an agent's mental states and have no effect on the world. These actions can be used both in the context and the body of plans. For instance, if the designer under certain conditions wants to propagate a plan failure up the intention stack, he/she can write a plan failure handling rule whose context condition encodes these conditions, and whose body has an internal action that removes the appropriate intentions.

In [7], Bordini et al. aim to provide a more practical programming language by specifying various selection functions of AgentSpeak(L). In particular, they provide specifications of relations between plans and quantitative criteria (such as the quality, duration, and cost of plans, and the deadlines specified for them) for their execution, and then use efficient decision-theoretic task scheduling to automatically guide the choices made by an agent's intention selection function. The design of the Jason APL allows atomic formulae and plans to have *annotations* [179, 108] which can be used by various user-defined selection functions.[4] For instance, annotations within the belief base can be used to register the source of the associated information, and can later be utilized by the (user-defined) belief update function. Annotations in action expressions can be used to implement sophisticated applicable plan selection and efficient decision theoretic intention selection functions. Another interesting feature of Jason is that it can be easily configured to run on various hosts. This is done using an agent communication infrastructure called SACI [75].

Another implemented agent programming framework is the Jadex software framework [18, 120]. Jadex is implemented as an agent reasoning layer that sits on top of the middleware agent infrastructure JADE [6]. The reasoning engine of Jadex is similar to that of AgentSpeak(L). Jadex utilizes both declarative and procedural approaches to define various components of an agent. It uses Java to procedurally define plan bodies (i.e. actions), and the XML language to declaratively define all other mental attitudes. While Jadex provides a semantics for declarative goals (as discussed in the next section), the current implementation does not utilize these goals. The Jadex toolkit comes with a graphical debugger and various other tools to help the application developer. Jadex has been used in various applications such as simulation, scheduling, and mobile computing.

Bordini and Moreira [13] use the transition semantics in [107] to prove various BDI properties of AgentSpeak(L) agents, including the intention-belief inconsistency principle. There has also been some work on automatic verification of AgentSpeak(L)-like programs. In [14, 10, 11], Bordini et al. introduce a toolkit called CASP which can be used to translate a simplified version of AgentSpeak(L) into the input language of existing model checkers for linear temporal logic, such as SPIN [73] and JPF2 [180].

An Abstract Agent Programming Language (3APL)

Another major PRS-based agent programming language that can be found in the literature is An Abstract Agent Programming Language (3APL) [70]. Like AgentSpeak(L), 3APL utilizes a procedural notion of goals/intentions, and specifies a static set of rules, now called *Practical Reasoning* rules or PR-rules, which operate on these goals. However, 3APL differs from AgentSpeak(L) in various ways and we discuss these differences below.

3APL incorporates the notion of *basic actions*, which are used to specify agents' basic capabilities. These actions are viewed as application dependent mental state transformers in that these change agents' beliefs. The specification of belief updates associated with a basic action is formally represented using a partial function $\mathcal{T}$ that returns the result of updating a belief base by performing an action. Note that $\mathcal{T}$ is a partial function, since the action may not be executable in

[4] However, these sophisticated selection functions are not yet provided with the current distribution of Jason.

some belief states. In contrast, recall that in AgentSpeak(L) one uses rules to update beliefs by generating a primitive addition/deletion event which triggers some update rules (treated like any other plan).

A 3APL agent consists of a belief-base, a goal-base, and a set of rules. While AgentSpeak(L) agents respond to events, 3APL agents respond to goals in their evolving goal-base. The concept of event is missing from 3APL. 3APL rules are triggered by conditions on the goals and belief-bases, rather than events. If an agent has reacted to some new goal or belief, she should memorize it so that the relevant rule won't fire twice. In [69], it has been shown that 3APL can bi-simulate AgentSpeak(L).[5] In response to a goal in the goal-base, a 3APL agent searches for a rule whose trigger condition can be unified with this goal, and whose context-condition follows from her beliefs. The agent then replaces the goal in the goal-base with the body of one such rule. Thus, the agent's goal-base in 3APL evolves over time and works like the intention-base in AgentSpeak(L).

3APL provides a richer set of rules than AgentSpeak(L). Also, the plan-body of a rule can now be constructed using various imperative programming constructs, rather than only using sequences as in AgentSpeak(L). PR-rules can be classified into four types, namely, failure rules, reactive rules, plan rules, and optimization rules. The roles of these rules are as suggested by their names. A typical failure or optimization rule is of the form $\pi_h \leftarrow \phi \mid \pi_b$, where π (possibly with subscripts) denotes a plan/program. This says that if π_h is part of the agent's plan, and she believes that ϕ, then this plan should be replaced by π_b. Note that failure rules with empty bodies can be used to drop a goal (plan). Reactive rules are rules with an empty head (i.e. of the form $\leftarrow \phi \mid \pi$), and states that whenever the agent believes that ϕ, it should adopt the plan/goal that π. These rules are used to create new goals. Finally, a plan rule is of the form $G(\overrightarrow{t}) \leftarrow \phi \mid \pi$, and states that when the agent believes that ϕ, one way of achieving the (atomic) goal $G(\overrightarrow{t})$ is π. In addition to facilitating planning for simple achievement goals, these rules in some sense provides a mechanism for revision and monitoring of goals.

Thus, a 3APL agent is a tuple $\langle \Pi, \sigma, \Gamma \rangle$, where Π is a possible goal-base, σ is a possible belief-base, and Γ is a possible PR-base. The operational semantics of 3APL is provided using a transition semantics. 3APL provides two sets of transition rules, one for specifying the execution of individual plans, and another for specifying the execution of an agent. Plan-level execution rules define what it means to execute a single goal, and include rules for executing basic actions, test goals, sequential goals, non-deterministic choice, and application of PR-rules. Agent-level execution rules, which are defined in terms of these plan-level execution rules, specify what it means for an agent to execute multiple goals in parallel. The overall semantics of 3APL is defined in terms of *computations*. A computation is a finite or infinite sequence of mental states such that the first mental state in this sequence is the initial mental state of the agent, and the successive mental states can be derived using the agent-level transitions.

Another novel feature of 3APL is that it separates the semantic specifications for the agent language, and its control structure, by introducing a distinction between an object-level and meta-level semantics. The control structure at the meta-level specifies which goals should be executed and

[5]The underlying idea for this involves the generation of an event-queue from the goal-base, and the creation of an intention-base from the goal-base and rule-base. However, this technique does not cover the deletion of goal events, and the addition and deletion of belief events (i.e. it only handles the addition of goal events).

which rules should be applied next. To this end, 3APL introduces a meta-language that includes some meta-actions and a meta-level transition system. To determine which goals (rules) should be executed (applied, respectively) next, the meta-language assumes that there is a fixed user-defined ordering on goals (rules, respectively). The transitions of the meta-actions are derived in terms of the object-level transitions. For instance, if there is a rule in the object-level transition system that says that a goal g is executable, then a meta-level transition rule selects g for execution provided that g is maximal w.r.t. the ordering on the set of all goals. The overall control structure of 3APL is a specialization of the update-act cycle. In the *planning/application phase*, the ordering on rules and goals is used to determine the strongest applicable reactive, plan, or optimization rule, and if there is such a rule, it is then applied to the agent's plan. After this, in the *filtering phase*, the controller uses the ordering on goals to choose a goal, searches for failure rules applicable on that goal, and applies *all* such rules. Finally, in the *execution phase*, it executes a single step of the chosen goal, provided that the first action of the chosen goal is executable. While AgentSpeak(L) has a similar control mechanism provided via the three selection functions $S_{\mathcal{E}}, S_{\mathcal{O}},$ and $S_{\mathcal{I}}$, the major difference is that (the original proposal of) AgentSpeak(L) [122] does not handle plan failure, and thus the filtering phase of 3APL is omitted from AgentSpeak(L).

5 Declarative Goals in Agent Programs

As mentioned, an important concept in the context of agent programming is that of declarative goals. We start our discussion on declarative goals by pointing out the differences in expressiveness between declarative and procedural goals, and the advantages of incorporating these goals in APLs.

5.1 Advantages of Declarative Goals

Declarative goals in agent programs are necessary for a variety of reasons. The major difference between declarative and procedural goals lies in the way they express an agent's degree of commitment towards a goal. Being committed to a procedural goal amounts to nothing more than being committed to a plan, i.e. to execute a (possibly infinite) sequence of actions that the procedural goal can be decomposed to. On the other hand, being committed to a declarative goal is much more expressive, and it amounts to being committed to one of all possible plans that achieve the goal. Thus, the difference between them can be viewed as commitment towards some means to an end vs. commitment towards an end. For a static environment, where the agent programmer has a complete model of the world and knows about all possible exogenous actions in advance, it may be possible for him/her to specify an extensive set of hierarchical rules that covers all the ways to achieve some goal (like a policy). However, this may require much effort; also this becomes even harder when the agent designer only has partial knowledge about the domain, and cannot predict all possible interruptions (generally a fixed utility function is assumed). Also, from a technical point of view, procedural goals have limited expressiveness. Procedural goals cannot be combined using logical operators. For instance, even if the agent programmer specifies two procedures for achieving two separate goals P and Q, he/she needs to write a third procedure for the conjunctive goal $(P \wedge Q)$.

Motivations for declarative goals in agent programs can be roughly classified into two categories, namely theoretical and practical motivations. From a theoretical point of view, it has been argued that declarative goals are required in order to bridge the gap between agent theories and APLs [71, 185, 171]. The reason for this is that in agent theories, goals are declarative concepts, and thus the incorporation of these goals is viewed as a necessary prerequisite for bridging this gap. From a practical perspective, various advantages of declarative goals have been pointed out in the literature. In the following, we discuss these advantages.

One reason for using declarative goals is to decouple plan execution and goal achievement. A declarative (achievement) goal represents a state that is to be reached. Declarative goals can be used to decide whether a plan was successful in achieving the associated goal or not. The successful execution of a plan does not necessarily indicate the successful achievement of a goal. Also, failure to execute a plan does not mean that the goal can't be achieved, since there might be another way of achieving the goal, one that is not described by the procedural goal. For instance, consider the following example of a hungry cat (from [185]). Initially, the cat knows that some food has been left on the table, and has the goal of reaching the food. If we are using a procedural representation of goals, one way to define this goal is using the cat's plan that $leapOnChair(chair); fromChairJumpToTable(chair, table)$, i.e. it should first jump on a chair that is close to the table, and then jump from the chair to the table, where the food is located. Suppose, with this goal in mind, the cat leaps on to the chair. At this point, a nearby human, realizing the cat's intention, moves the chair further away from the table. Thus, since the second action is no longer executable, the cat's plan has failed, and since we are using a procedural definition of goal, the cat's goal has also failed. Note that, by using only procedural goals, *the reason for performing the plan* is lost. On the other hand, if we were using a declarative definition of goals, we can use $NextTo(Cat, Food)$ as the goal. In that case, even though the cat's plan fails, it can try to plan again using this goal. Consider another example in the same domain: suppose that the cat was successful in jumping on to the table. Note that, this does not necessarily mean that it was successful in reaching food, since somebody might have removed all the food from the table. These examples illustrate that the success and failure of a plan do not tell us much about the success or failure of the associated goal. Once again, this is especially true in dynamic environments where unexpected events may occur, and it is impossible for the designer to predict all possible interruptions caused by exogenous actions.

Declarative goals can also be used to detect fortuitous achievement of goals. For instance, suppose that some food was left on the chair. So after leaping onto the chair, the cat can detect this, and since its goal to reach the food has been achieved, it can drop the plan to jump to the table. Now, consider why an APL that does not incorporate declarative goals, such as 3APL, cannot detect this. Recall that in 3APL, the agent programmer specifies rules that can be triggered due to the presence of procedural goals in the goal-base. Then the procedural goal is decomposed to/replaced with the rule-body, and eventually executed to achieve the goal. Suppose that the programmer only has partial knowledge about the cat's environment, and wrote the following rule that can be used to achieve the cat's goal:

$$goNextToFood(Cat) \leftarrow true \mid leapOnChair(Chair); fromChairJumpToTable(Chair, Table).$$

Assume that the programmer does not know that unusual situations such as one where food is left

on the chair can occur. In response to the procedural goal $goNextToFood(Cat)$ in the goal-base, the cat will adopt the plan $leapOnChair(Chair); fromChairJumpToTable(Chair, Table)$. After executing the first action, the cat is on the chair. However, although food is at its current location, the 3APL-cat not knowing that the state that needs to be reached (i.e. $NextTo(Cat, Food)$) has been reached, will not drop the goal of jumping to the table. Thus, if goals are defined procedurally, the cat still has the plan to jump on the table even though its goal has been satisfied. Note that, while a 3APL PR-rule of the form $(\delta \leftarrow \phi \mid nil)$, which says that if the agent believes that ϕ and has the plan that δ, then she should give up this plan, can be used to detect this, it involves the use of a declarative goal ϕ in the context condition. Also, in order to do this, the agent programmer needs to specify an extensive set of such rules. The reason for this is that an exogenous action could occur at any stage during the execution of the plan, and thus the dropping of the current plan due to early achievement of the associated goal should be considered for all possible configurations of the plan. For instance, even for a simple plan $a_1; a_2; a_3$ that includes three primitive actions in sequence, the programmer needs to specify the three following rules: $(a_1; a_2; a_3 \leftarrow \phi \mid nil)$, $(a_2; a_3 \leftarrow \phi \mid nil)$, and $(a_3 \leftarrow \phi \mid nil)$.

Another important motivation for using declarative goals is communication. While an agent can delegate a procedural goal to another agent, she is required to plan for the goal before she can delegate it. Moreover, the requester may not know how to achieve the goal. Thus the use of declarative goals allows distribution of both computation and knowledge, in the sense that the requesting agent need not plan for all of her goals. Declarative goals are also necessary for reasoned responses to communication. For instance, to determine if an agent should adopt a request to achieve a goal, she must know whether this requested goal conflicts with her own goals. Moreover, even if this requested goal conflicts with one of her plans, she (being a very helpful agent) might still decide to adopt it provided that it does not conflict with the associated declarative goal.

Declarative goals are essential for rational behavior. In addition to allowing an agent to plan from scratch when all her plans in the plan library have failed, declarative goals facilitate reasoning about interferences among goals. An agent may have two conflicting goals in the goal base. To decide which goal to achieve, it can do some reasoning with these declarative goals to find out which of these is more important to the agent.

5.2 Issues in Agent Programming Languages with Declarative Goals

As mentioned earlier, recently there has been some work on APLs with declarative goals. Before going over these frameworks, we briefly discuss the common issues in APLs with declarative goals.

One issue in APLs with declarative goals concerns the type of goals handled by an APL. Most APLs use achievement goals as the only type of goals. Achievement goals refer to a goal that needs to be achieved once. Some APLs also allow the use of other types of goals, e.g. maintenance goals and perform goals (i.e. a goal to execute some actions). Some also allow the use of inconsistent goal bases (e.g. GOAL [71], Dribble [174]). They assume that two inconsistent achievement goals need to be achieved at different times. Thus, although in these frameworks the goal-base can be inconsistent, the adopted declarative goals (i.e. intentions) must be consistent with eachother. Note that, the need for such inconsistent goal-bases arises from the fact that these APLs take

goals to be state formulae, rather than general temporal formulae. For instance, if an agent has two inconsistent achievement goals ϕ and $\neg\phi$, the agent's goal state could be represented by the temporal formula $\Diamond\phi \wedge \Diamond\neg\phi$, i.e. the goal to eventually achieve ϕ and to eventually achieve $\neg\phi$, which is consistent. One exception to this is [135], that defines a goal as a path formula; however it does not deal with the dynamics of declarative goals. Some frameworks model goals with different priorities (e.g. [135]).

Another issue concerns the representation of declarative goals and intentions. In other words, how can one incorporate these goals as a part of the programming language, and what features of these goals are included in the framework. Most APLs keep declarative goals and procedural plans in two separate databases. They then use these goals to trigger some AgentSpeak(L) or 3APL-like rules. Other frameworks treat declarative goals as individual and active components that manage their own state and post events as required (e.g. Jadex [19]). These events are then handled by AgentSpeak(L)-like agents. These frameworks also define the life-cycle of these goals explicitly.

A third issue concerns the consistency of intentions. Two intended goals can be conflicting in various ways, such as:

- A goal can be directly inconsistent with another goal. In that case, all the plans for achieving that goal conflicts with all of the plans to achieve the other goal.
- A goal can be inconsistent with some of the plans to achieve another goal, and not others. In that case, any plan to achieve the first goal is in conflict with some of the plans to achieve the second.
- Two goals may be mutually consistent. However, it may be the case that some of the plans to achieve one goal are inconsistent with some of the plans to achieve the other.

Thus, while selecting a plan to achieve a goal, the agent must check that only consistent plans are selected. This ensures that the agent will not commit to and execute a plan that makes one of her goals impossible to achieve. Unfortunately, in most APLs with declarative goals, there is no requirement that a declarative goal be consistent with a procedural goal, i.e. plan. The fact that these APLs maintain intended goals and plans in two separate databases makes it even harder to ensure this consistency. In fact, to the best of our knowledge, no APL in the literature handles these issues.

A fourth issue involves the representation of means-ends relationship between goals and sub-goals or plans adopted to achieve these goals, i.e. how these frameworks capture the dependency between sub-goals and their parent-goals. Recall that, PRS-based agents adopt plans in response to procedural goals. Similarly, in PRS-based declarative goal oriented APLs, agents adopt plans in response to declarative goals in the goal-base. These plans may in turn involve the achievement of other declarative (sub)goals that may trigger the adoption of other plans. Note that, the only reason the agent intends any of these sub-goals and plans is due to her commitment towards the parent goal. In other words, the agent's commitments towards these sub-goals can be viewed as conditional intentions, (implicitly) conditioned on intending the super-goal. Thus, if in any situation the agent drops the super-goal, she should also drop all these sub-goals. Most APLs do not model this dependency, and thus fail to give up sub-goals or plans when the associated goal is

dropped. The ones that do, share a similar underlying technique: they introduce some construct in the language that captures the fact that a sub-goal was adopted to achieve a goal (and a sub-sub-goal was adopted to achieve the sub-goal, etc.). Then, when the goal is dropped, they use this information to drop all such sub-goals (and the sub-sub-goals etc., if any). For instance, GD-3APL [34] attaches a declarative goal with each intended plan, and this information can be used to abandon plans when necessary. However, no mechanism is provided to use this information to effect the appropriate goal contraction. CAN [185] includes a procedural construct that includes both the associated declarative goal and its failure condition with an adopted plan.

Another issue concerns how these declarative goals are used by these frameworks, i.e. which of the aforementioned advantages of declarative goals are realized. As mentioned earlier, the literature identifies three major uses of declarative goals: selecting plans using these goals, decoupling plan success/failure from goal success/failure, and planning for goals on-the-fly.

In the following, we discuss in more detail various declarative goal-oriented APLs found in the literature.

5.3 Declarative Goal Oriented Languages

GOAL

The programming language Goal Oriented Agent Language (GOAL) [71] can be identified as one of the firsts to attempt to incorporate declarative goals in agent programming languages. Like 3APL, GOAL integrates theory and programming in a single framework by providing both an agent programming framework and a programming logic, the later derived from the operational semantics of the former. Thus statements proven in the logic concern properties of agents specified in the programming language. Unfortunately, GOAL does not take most of the advantages of declarative goals (as discussed in Section 5.1) into account, and only uses declarative goals to select plans.

A GOAL agent keeps a propositional belief-base and a declarative goal-base. To consider the possibility of mutually inconsistent goals (to be achieved at different time steps), the goal-base is allowed to be inconsistent. However, individual goals need to be consistent, and believed propositions cannot be in the goal-base. An agent's goals are then defined to be the formulae that are entailed by an entry in the goal-base, rather than those that are entailed by the entire goal-base. The reason for doing this is that, since the agent can have mutually inconsistent goals, defining goals using the later can trivialize the logic. Thus agents' goals are modeled using a weak logic that does not include the K axiom, and as a result, goals do not distribute over implication, and two goals cannot be conjoined to form another goal. Note that, this formalization of goals is very "syntactic" and can only handle achievement goals.

A central idea in GOAL is that of *conditional actions*. These actions are used to help agents decide what actions to perform next, and thus can be viewed as very simple action selection rules. Intuitively, a conditional action $\phi \rightarrow do(a)$ states that the agent may consider executing the basic action a if the mental state condition ϕ holds. Basic actions are defined similarly as in 3APL, i.e. using a (unspecified) partial function on the belief-base. Also, two special actions $adopt$ and $drop$

are introduced to respectively adopt a new goal, or drop some old goals. The semantics of $adopt$, $drop$, and conditional actions are specified in terms of the semantics of basic actions. GOAL adopts a simple blind commitment strategy [26]. The authors argue that this is just a default strategy, and that conditional actions (with a $drop$ in the consequence) can be used to override this commitment strategy. However, it is not obvious how any other strategies can be adopted. E.g., say one wants to specify a commitment strategy that enables an agent to give up her goals when they become impossible to achieve. Without a temporal component built into the goal semantics, this is clearly impossible to express.

A GOAL agent is thus defined to be a tuple $\langle \Pi, \Sigma_0, \Gamma_0 \rangle$ consisting of a set of conditional actions Π and an initial mental state $\langle \Sigma_0, \Gamma_0 \rangle$, where Σ_0 is the initial belief-base and Γ_0 is the initial goal-base. While AgentSpeak(L) and 3APL agents search for applicable rules and execute intended actions at every step, a GOAL agent searches for an appropriate conditional action. Since a conditional action associates a basic (primitive) action with a mental condition, the deliberation mechanism of GOAL is indeed a very primitive one. The overall operational semantics of GOAL agents is given using *traces*, which are infinite sequences of consecutive mental states interleaved with scheduled conditional actions, where the first state of each of the traces is the agent's initial mental state.

The specification for basic actions provides the basis for the programming logic of GOAL. Actions are specified using Hoare triples of the form $\{\phi\}\ a\ \{\psi\}$, where ϕ and ψ are mental state formulae. Hoare triples for conditional actions are interpreted relative to the set of traces associated with the GOAL agent, and a time-point in these traces. These user-defined Hoare triples are used to specify preconditions, post-conditions, and frame conditions of actions. On top of the Hoare triples for specifying actions, a temporal logic is defined for specifying properties of GOAL agents. One can then express various *liveness* and *safety* properties of an agent A by considering the temporal formulae that are valid with respect to the set of traces S_A associated with A. It can be shown that such properties are equivalent to a set of Hoare triples. Thus the properties can be proven by showing that the Hoare triples are entailed by the specifications of the actions that appear in the program. Thus it is very straightforward to verify the properties of agents in GOAL.

Note that, a rich notion of action structure is missing in the GOAL programming language. All one has is simple condition-action rules. Moreover, the only deliberation and planning mechanism in GOAL is provided via conditional actions that allow the agents to select primitive actions based on their mental states.

Dribble

Unlike GOAL, Dribble [174] incorporates a procedural motivation component (i.e. plans) in the language. In particular, Dribble takes 3APL's [70] mechanism for *creating and modifying plans* during the execution of the agents, and GOAL's facility for *using declarative goals for selecting actions* into account, and combines these in a single framework. Thus, Dribble uses declarative goals to allow agents to select and modify plans when required. Moreover, as in GOAL, Dribble also includes a dynamic logic on top of its operational semantics to specify and verify properties of agents.

In addition to beliefs and declarative goals, the mental state of a Dribble agent also includes a plan component. Only a single plan can be handled at any one time (no concurrency). The plan of an agent can be changed through application of rules as well as execution of executable actions. Dribble defines two types of rules, namely *Goal rules*, and *Practical Reasoning* (PR) rules. A goal rule g is a pair $\varphi \rightarrow \pi$ such that φ is a propositional formula involving beliefs and goals, and π is a plan. Intuitively a goal rule says that the plan π can be adopted if the mental condition φ holds. Goal rules are used to *select* plans for the first time (i.e. when the plan component of the agent is an empty plan). The condition in φ specifies what the plan π is good for. On the other hand, PR rules are similar to rules in 3APL, and can be used to create plans (possibly from abstract plans), to modify plans, and to model reactive behavior (using rules with empty heads).

Programming a Dribble agent amounts to specifying its initial beliefs and goals and writing sets of goal rules and PR rules. Formally, a Dribble agent is a triple $\langle\langle\sigma_0, \gamma_0, E\rangle, \Gamma, \Delta\rangle$, where $\langle\sigma_0, \gamma_0, E\rangle$ is the initial mental state with initial belief-base σ_0, initial goal-base γ_0, and an empty plan E, and where Γ is a set of goal rules and Δ is a set of PR rules. Note that the initial plan of a Dribble agent is the empty plan E. The reason for this is that a Dribble agent should be able to select a plan (using rules) based on its declarative goal specification, and giving the agent a plan at start up is counter-intuitive in this respect. The operational semantics of the Dribble programming language is specified using a transition system. A computation run $CR(s_0)$ is a finite or infinite sequence $s_0, \cdots, s_n$ or $s_0, \cdots$, where $s_i \in \langle\sigma_i, \gamma_i, \pi_i\rangle$ are mental states (where π_i denotes the plan of the agent), and for all i there exists a transition from s_{i-1} to s_i as defined in the transition system for the Dribble agent. The meaning of a Dribble agent $\langle\langle\sigma_0, \gamma_0, E\rangle, \Gamma, \Delta\rangle$ is then defined to be the set of its computation runs $CR(\langle\sigma_0, \gamma_0, E\rangle)$. Thus the first state of the computation runs is the initial mental state of the Dribble agent.

As mentioned, Dribble is an expressive language that improves on GOAL by adding complex plans and rules to manipulate goals. Nevertheless, it has some limitations. Although the authors argue that goal rules and PR rules together can be used as a regression planning mechanism, this is misleading, since no lookahead is incorporated. Also, Dribble does not support exogenous actions; e.g., suppose that the agent has δ as a plan, and some exogenous action happens, which makes the preconditions of δ false forever. While a PR rule with an empty body can be used drop this plan, without a temporal component built into the language, it is impossible to detect that the plan has become forever impossible to execute. Note that, such a mechanism is important if one wants to ensure that the agents do not intend unachievable goals. Moreover, Dribble only allows sequential plan adoption and execution. In other words, agents cannot concurrently commit to two different plans. Finally, one major problem in Dribble is that it uses distinct databases for two types of intentions, i.e. declarative goals and procedural plans, and there is no mechanism for ensuring consistency between these two. Put otherwise, Dribble semantics allows agents with an inconsistent intention-base, e.g. an agent can have a declarative goal ϕ and a plan that makes ϕ unachievable. While other programming languages address some of the problems mentioned earlier, to the best of our knowledge, no BDI agent programming language with both declarative and procedural goals offer a solution to this problem.

Goal Directed 3APL (GD-3APL)

In [34], Dastani et al. present an extension of 3APL [70], called Goal Directed 3APL (GD-3APL), that incorporates both declarative and procedural goals into a single framework. GD-3APL agents are similar to Dribble agents, in that they have beliefs, goals, plans, and rules. The overall semantics is also very similar. The major difference is that GD-3APL uses a more expressive first order language, and that it defines an additional rule type to allow the agents to reason about and modify their declarative goals. GD-3APL provides three types of rules: *Goal Revision* (GR) rules, *Plan Selection* (PS) rules, and *Plan Revision* (PR) rules. GR rules can be used to revise goals, and variants of GR-rules can be used to generate, extend, or drop goals. In some sense, these rules allow agents to reason about their declarative goals. PS rules are like Dribble's Goal rules and PR rules are similar to practical reasoning rules in Dribble and 3APL.

Another advantage of GD-3APL is that an agent's plans/procedural intentions are modeled using a $\langle$plan-body, goal$\rangle$ pair, where the second element is added to record the goal for which the plan was selected. So if the goal gets dropped for some reason or revised by a GR rule before the agent has finished executing this plan, this information is used to also drop the plan. This facilitates decoupling of plan success from goal success. However, exactly how this information can be utilized is left open in the framework. Finally, GD-3APL agents can concurrently handle several goals, by committing to and executing multiple plans, unlike Dribble agents. Note that, GD-3APL does not provide a logic to verify properties of agent programs. Also, an offline lookahead mechanism for planning is still missing.

A BDI Extension of the Golog-Family

One problem with all of the above frameworks is that although they may support declarative goals, they do not support planning in the sense that there is no lookahead mechanism built into these frameworks. Also, most of these frameworks are not grounded on a formal theory of action, and thus only allow limited reasoning. The agent programming language proposed by Sardiña and Shapiro [135] (let us call it S&S) combines two existing approaches to agent theory (viz. the work in [144]) and to agent programming (namely, IndiGolog) to provide an expressive BDI-agent programming language that supports planning/lookahead. S&S is built on top of a situation calculus-based action theory.

In S&S, an agent program consists of a high-level procedural specification of the agent's behavior (i.e. a single non-deterministic program), a declarative specification of the agent's mental states, and an underlying action theory. The interpreter's job is to search for a *rational* execution of the given program (i.e., one that satisfies the agent's goals, as discussed below). An agent's mental state consists of her beliefs and her goals. S&S incorporates a KD45 model of knowledge and a KD model of goals, both specified in terms of accessibility relations over possible worlds. The model of goals has a temporal component associated with it, and thus it can handle both achievement and maintenance goals. Also, S&S supports prioritized goals through prioritized accessibility relations and all goals are not assumed to be equally important. $(\phi_1 > \phi_2 > \cdots > \phi_n)_s$ is used to represent that the agent in situation s has n prioritized goals, where ϕ_i denotes all the goals of the agent

at level i, ϕ_1 being the highest priority goal. Moreover, S&S's language is rich enough to allow queries of whether the agent is able to achieve certain goals in a given situation.

To help the agents decide which plans are preferable, S&S defines an ordering on plans/strategies, which are modeled using action selection functions (as discussed in Section 2.4). It then defines a rational course of action to be a strategy that the agent *knows-how* (i.e. is able) to execute, and she knows is one of the most preferred strategies w.r.t. her prioritized declarative goals.

As discussed above, most reactive plan execution languages that incorporate declarative goals include a pre-compiled plan library. The agents use their declarative goals as triggers to select plans from this library, and hierarchically decompose and execute these plans. Unlike these APLs, S&S uses an IndiGolog-style controller. Recall that, in IndiGolog, the programmer's job is to model the domain using the appropriate axioms, and specify the agent's behavior using a high-level non-deterministic program. Given a starting situation, the IndiGolog interpreter tries to (incrementally) find an execution of this non-deterministic program. S&S uses a similar control strategy. However, the S&S interpreter also needs to take the agent's declarative goals into account.

To this end, the rational-search operator $\Delta_{rat}(\delta : X_G)$ was introduced in S&S. The idea of this construct is that, given a non-deterministic IndiGolog program δ and a set of prioritized goals $X_G = \phi_1 > \phi_2 > \cdots > \phi_n$, the $\Delta_{rat}(\delta : X_G)$ operator will produce a simple and ready to execute terminating deterministic plan (i.e. sequence of actions or conditional plans) δ', whose execution respects both the given program δ, in the sense that δ' is an instantiation of δ, and the set of declarative goals, in that it achieves as many highest priority goals as possible (i.e. it is a rational course of action w.r.t. the given set of declarative goals). S&S assumes that the given program has the highest priority, and thus any declarative goals that conflict with this program will not be achieved. This operator provides a mechanism for combining a procedural specification of behavior and a set of prioritized declarative goals, and is meant to be used by the agent programmer to specify when lookahead is necessary.

To the best of our knowledge, S&S is one of the only two BDI agent programming frameworks[6] that offers deliberation with lookahead, and thus supports planning. Thus, S&S combines a procedural representation of behavior and prioritized declarative goals in an expressive language that is able to model ability and know-how, the temporal aspects of goals and actions, and the relative importance of goals. However, this expressiveness of S&S comes at the cost of complexity – determining whether a plan is rational or not involves searching the plan space defined by the given high-level non-deterministic program, and comparing all the strategies that can be induced by this program. This is clearly problematic in a dynamic environment. In fact it is unknown whether there exists a practical procedure to implement this mechanism. Also, it can be argued that agents in S&S are not so goal-directed, since their overall behavior is controlled by the given program.

CAN and CAN-PLAN

As mentioned earlier, another agent programming language that supports lookahead/planning is CAN-PLAN [134], which is an extension of the Conceptual Agent Notation language (CAN)

[6] Another such language, as discussed below, is CAN-PLAN.

[185]. In contrast to the situation calculus based S&S, CAN-PLAN is a PRS-based language.

The underlying basic infrastructure of CAN is similar to that of AgentSpeak(L). Agents in CAN have a first-order belief-base, a set of plans, and a first-order intention-base, and thus an agent configuration is modeled as a tuple $\langle B, A, \Gamma \rangle$, where B is a possible belief-base, A denotes the actions performed by the agent so far, and Γ represents a possible intention-base. Plan-bodies or programs in CAN can be constructed from primitive actions, operations to add/delete beliefs, tests for conditions, events or achievement goals $!e$, sequences, parallelism, the operator $\triangleleft \{\cdot : \cdot\} \triangleright$ which is used to represent a set of guarded alternatives as discussed below, the choice operator dual of sequencing $P_1 \rhd P_2$ which executes P_1 and then executes P_2 only if P_1 fails, and the operator $Goal(\phi_s, P, \phi_f)$ (discussed later). CAN uses a set of transition rules to specify the evolution of an agent. Agents in this language respond to events. In response to an event e, a CAN agent uses the plan-library Π with rules of the form $(e' : \phi_i \leftarrow P_i)$ to collect all context condition-plan-body pairs $\phi_i : P_i$ whose event e' can be unified with e, places these pairs inside the special construct $\triangleleft \{\} \triangleright$ to form the plan $I_1 = \triangleleft \phi_1 : P_1, \cdots \phi_n : P_n \triangleright$,[7] and inserts I_1 into the intention-base. Another transition rule is then used to (non-deterministically) choose from one of the plan-bodies in I_1 whose context condition holds. Suppose that the context condition ϕ_1 holds, and the agent chooses to try P_1 first. In that case, I_1 will be replaced with the new intention $P_1 \rhd I_2$, where $I_2 = I_1 - \{\phi_1 : P_1\}$, i.e. with the plan that P_1 should be tried first, and if the execution of P_1 fails for some reason, the intention I_2 (which is similar to I_1 but now does not include the pair $\phi_1 : P_1$) will be attempted. Thus, unlike AgentSpeak(L) and similarly to Jason and 3APL, CAN provides a mechanism for handling failure of plans. However, this is very different from 3APL failure rules.

In addition to this basic infrastructure, CAN provides a mechanism for representing both declarative and procedural goals in a uniform manner. For this purpose, it uses the procedural construct $Goal(\psi_s, P, \psi_f)$, which can be read as: the agent should achieve the declarative goal ψ_s using the set of procedures P (which is of the form I_1 discussed above), failing if ψ_f becomes true. CAN provides a set of transition rules for $Goal(\psi_s, P, \psi_f)$ defined in terms of the above rules. The execution of a $Goal(\psi_s, P, \psi_f)$ construct is specified such that at every step, it only updates the associated P (by executing a step of P), giving up only when the goal ψ_s is achieved, or when it is no longer possible (i.e. when ψ_f holds). Using these rules, CAN guarantees that the execution of $Goal(\psi_s, P, \psi_f)$ will obey some of the properties of declarative goals discussed in Section 5.1. For example, the explicitly specified success condition can be used by the semantic specification to detect early/fortuitous achievement of goals (i.e. achievement of goals before the associated plan has been fully executed) and drop the associated goal and plan. Also, it can be used decouple successful execution of plans from successful achievement of goals. This is done by checking whether the success condition ϕ_s holds after the execution of a plan from P; if the plan has been successfully executed, but the goal has not been achieved yet, the CAN agent will try another plan from P. If the goal remains false after all plans in P have been executed, the CAN agent will retry the plans in P. This mechanism is provided by replacing the intention $Goal(\psi_s, P, \psi_f)$ with $Goal(\psi_s, P \rhd P, \psi_f)$ at the beginning of the execution, and whenever P is not of the form $P_1 \rhd P_2$ (i.e. after it has already tried P but failed to achieve the associated goal). Similarly, the failure

[7] Note that we simplified the notation a bit by getting rid of the variable bindings.

condition is specified to decouple plan failure from goal failure, and to remove any committed-to plans when the associated goal has failed (or becomes impossible). This special construct is meant to appear in the plan body part of rules specified by the agent programmer. Thus, this mechanism of CAN can be used for both failure handling and monitoring of declarative goals.

CAN-PLAN [134] extends CAN by including a lookahead mechanism to support offline planning. To perform offline planning, one needs an action theory. To this end, agents in CAN-PLAN are equipped with a simple STRIPS-like action description library Λ that contains rules of the form $a : \psi_a \leftarrow \Phi_a^-; \Phi_a^+$, one for each action a in the domain. Here ψ_a corresponds to the preconditions of a and Φ_a^+ and Φ_a^- denotes the add and delete list of atoms, respectively. CAN-PLAN incorporates the additional $Plan(P)$ operator in the plan language. This operator searches for a complete hierarchical decomposition of P before executing a single step in a similar way to an HTN planner. It is very similar to the IndiGolog Σ search operator, in that $Plan(P)$ can evolve to $Plan(P')$, provided that P can evolve to P' and can reach a final configuration in a finite number of steps. In CAN-PLAN, the agent programmer can mix the $Goal()$ and $Plan()$ operators in various ways to produce different types of failure handling and lookahead. For example, consider the construct $Goal(\phi_s, Plan(Goal(\phi_s, P, \phi_f)), \phi_f)$;

- The external $Goal()$ operator ensures that the agent will use the program $P^* = Plan(Goal(\phi_s, P, \phi_f))$ towards the eventual satisfaction of the goal ϕ_s. The agent is committed to ϕ_s, in that P^* is reinstantiated and retried until ϕ_s holds. Also, It is not necessary to completely execute the plan returned by the planner (i.e. P^*), e.g. if ϕ_s is satisfied before P^* has been fully executed. Finally, the goal ϕ_s is dropped when failure condition ϕ_f becomes true.

- The $Plan()$ operator guarantees that the program P^* has a terminating execution. Note that, an exogenous action might render this program non-executable; however, as mentioned above, in that case the external $Goal()$ operator will call the planner again.

- The internal $Goal()$ operator ensures that the agent will use the program P towards the eventual satisfaction of the goal ϕ_s. Also, at plan-time, P is solved up to the point where the goal is met.

It can be shown that for a restricted class of CAN-PLAN agents,[8] the $Plan()$ operator indeed corresponds to a HTN-planner in the sense that for an agent, there is an execution of $Plan(P)$ in CAN-PLAN, if and only if there is a solution to the corresponding planning problem in a HTN-planner. Also, any execution of $Plan(P)$ corresponds to a HTN-plan solution.

Thus, CAN-PLAN provides a mechanism for on demand lookahead planning to the agent programmer. While the $Plan()$ operator itself does not consider potential interaction with exogenous actions, in some sense $Goal()$ and $Plan()$ can be mixed to handle external interferences. For instance, $Goal(\phi_s, Plan(P), \phi_f)$ will re-plan if the initial plan obtained fails due to an external interference. Nevertheless, CAN-PLAN's lookahead feature is local in the sense that it does not take into account other concurrent intentions. In other words, the result of planning may include actions

[8]This constraint restricts the belief-base language of an CAN-PLAN agent to that of an HTN planner.

that are in conflict with other goals of the agent. Also, while CAN-PLAN uses an action theory for deliberation, it does not utilize this for updating the agent's state, which is a bit inconsistent.

Jadex

Following [33, 167, 89], Braubach et al. [19] propose to treat declarative goals as first class objects in the Jadex framework. Declarative goals in Jadex are individual entities that manage their own state (in contrast to being managed by the agent), and post appropriate events as necessary. Recall that Jadex agents are PRS based agents that have a built-in plan library, and that respond to events. Thus, to handle these declarative goals, all an agent has to do is to listen to and respond to these events posted by the goal objects by adopting, executing, or dropping the appropriate plan. Note that, the current implementation of Jadex [18] does not utilize these declarative goals, and thus it only provides a specification of an extended version of the language.

Braubach et al. studied various goal oriented agent programming languages, architectures, and methodologies, and identified four different types of goals: achievement goals, maintenance goals, perform goals, and query goals. Here, a perform goal specifies some activities to be done, and hence the success of the perform goal depends only on the fact that the activity was performed. They argued that perform goals are different from achievement or maintenance goals, since they do not require any state to be achieved or maintained. Query goals serve the purpose of information acquisition. If there is enough information in the agent's knowledge-base to answer a query goal, it succeeds with that answer; otherwise it becomes the achievement goal of collecting enough information to answer that query.

By analyzing these goals, Braubach et al. identified various states that a goal can be in its lifecycle. The basic three states of a generic goal are *new, adopted,* and *finished*. An adopted goal can in turn be in various sub-states, such as *option, active,* and *suspended*. Adopting a goal makes it desirable to achieve it and thus it can be seen as an option that the agent can possibly pursue when the actual circumstances allow. To actively pursue a goal, the agent's deliberation mechanism must activate the goal to initiate goal processing. Active goals can be later deactivated by the deliberation mechanism, and saved as an option. For instance, an active goal needs to be deactivated when it conflicts with another higher priority goal. On the other hand, an active or option goal can be suspended when its context becomes invalid. For example, a robot that has a goal of guarding some property during the night can suspend this goal in the day. Unlike options, suspended goals are not fed to the deliberator when it is deciding on what goal to actively pursue next. When the context becomes valid again, the suspended goal is added as an option.

For each of the four goal types, Jadex specifies a refinement of the active state by considering various attributes and the specific life cycle of these goals. For instance, an achievement goal has three sub states of the active state, namely *in process*, *succeeded*, and *failed*. It also consists of two conditions, a *target condition*, and a *failure condition*. The target condition specifies the world state that the achievement goal wants to bring about, and the failure condition specifies the conditions under which the goal should be dropped and considered to have failed. An active achievement goal will first check the target condition for fulfillment of the goal, and if the condition is met, the goal can be moved to the succeeded state and eventually dropped. If both the target and the failure

conditions are false, the active achievement goal can post a goal addition event to the event queue. In response to this event, the agent will eventually adopt an applicable plan, and execute this plan to achieve the goal. At any stage of this execution, if the target or failure conditions are met, the goal will move to the succeeded or failed state, and post a drop event. To handle this event, the agent needs to drop this plan (along with all sub-goals adopted for this goal). Braubach et al.'s proposal also specifies how the other three types of goals are processed (see [19] for details).

Thus, like CAN, Jadex allows full decoupling of goals and plans by monitoring declarative goals. In other words, this framework provides a way to decouple both goal-plan failure and success. Moreover, as in CAN, it provides a mechanism for detecting fortuitous or early achievement of goals. Unfortunately, Jadex does not provide a formal semantics for these goals and their dynamics. While it allows the use of incompatible goals, it does not deal with the relationships between these incompatible goals. For instance, it does not require a plan adopted to process a goal to be compatible with another adopted goal.

6 Open Problems and Research Directions

In this document, we reviewed research concerning logical formulation of declarative goals in agent programming languages. Many of the existing agent programming languages do not incorporate declarative goals. Moreover, the ones that do, often do not utilize many of the advantages of declarative goals, as discussed above. A few existing frameworks that model planning either are intractable, or only consider one goal at a time. In other words, while planning, they ignore other concurrent goals of the agent, and as a consequence, the output of planning may not be consistent with other adopted goals. As we have seen in Section 5.1, there is no doubt that declarative goals are more expressive than procedural ones. However, an agent programming framework that incorporates most of the advantages of declarative goals has yet to be developed. In this section, we suggest some possible extensions to the existing frameworks, which might advance this objective:

- Developing a uniform formalization of intended goals and plans in an expressive first order language like the situation calculus – adopted declarative goals should be consistent with adopted plans. An agent's intentions should be side-effect free, introspectable, etc. (as discussed in Section 2), and expressive enough to capture conditional and prioritized intentions, as well as temporal intentions (e.g. maintenance goals).

- Modeling dynamics of intention – in addition to modeling intentions, clearly the issue of intention/goal dynamics needs to be handled. It should satisfy the properties of intentions given by agent theorists, such as in [26]. For instance, it should always maintain the consistency of the intention-base. Also, intentions should be achievable by the agent, thus already achieved and unachievable intentions should be dropped. It should handle general forms of contraction and revision of intentions (i.e. similar to the ones that can be done in belief-revision frameworks [61]).

- Managing the means-ends relationship between goals and sub-goals – sophisticated mechanisms are required to capture the dependencies between declarative goals and the sub-goals

and plans adopted to achieve those goals. For instance, sub-goals and plans (and possibly sub-sub-goals, etc.) adopted to bring about a goal should be dropped when the associated goal fails or is achieved.

- Formalizing more realistic lookahead/planning – planning should account for other concurrent goals, as discussed in Section 5.3. This is required for both maintaining the consistency of the intention-base (in the sense that generated plans should not be contradictory to adopted goals and plans), and ensuring that the agent does not intentionally do anything that makes one of her intentions impossible to achieve.

- Achieving a balance between deliberation and execution to cope with dynamic worlds – APLs should allow both reactive plan execution and deliberation. While reactive execution of plans speeds up the agent's operations considerably, deliberation is also necessary to handle unanticipated situations. Moreover, a reasonable model for determining how much lookahead is necessary, formalizing whether employing deliberation has good pay off in a given situation, and capturing how quickly the agent has to act to achieve a goal etc. needs to be developed. Note that, these issues are related to the resource-boundedness of the agent (as discussed at the beginning of Section 3).

- Developing representations of (incomplete) knowledge, and nested beliefs and goals, i.e. an agent's beliefs about others' beliefs and goals that can be reasoned with efficiently – this is required for dealing with incompletely known environments when multiple agents are involved.

Additional issues may arise when developing a unified agent programming framework that deals with many of the above issues. For example, there is a direct tradeoff between expressiveness and tractability in these languages, and thus an efficient APL is more likely to be not so expressive, and vice versa.

In particular, S&S's framework discussed in Section 5.3 is a unique framework (that is built on top of a fully specified formal theory of action, unlike most other APLs) that can be used as a starting point. The following is a list of possible extensions to this particular framework:

1. One extension involves modifying the execution style, and the representation of declarative goals and procedural goals or plans. Recall that, in this framework, the programmer specifies, among other things, a single non-deterministic program and an ordered set of declarative goals. The interpreter's job is to search for an execution of the program. If the programmer requires lookahead, he/she can utilize the on-demand rational search operator. In that case, the interpreter will also ensure that the planned program/sub-program satisfies as many of the highest priority goals of the agent as possible. As discussed in Section 5.3, it can be argued that this style of program execution does not really model goal-directed behavior. The reason for this is that in this framework, an agent's goals do not determine what plans she should execute; rather, the program (which is assumed to have the highest priority) is in control. It would be thus interesting to modify this framework to model a more goal-driven program execution. To this end, one could start with an execution scheme similar to the

PRS-based or HTN planning APLs, where user specified rules can be used to trigger plans from a pre-compiled plan-library. Thus, instead of an ordered list of goals and a single program, we would like to generalize this framework by incorporating an ordered set of goals and a set of plans, and developing a mechanism for merging these goals with the adopted plans so that the consistency of the intention-base is maintained.

Another reason for incorporating rules in this framework is to allow arbitrary modification of intended plans. As we have seen in Dribble and GD-3APL, an agent's (plan revision) rules can arbitrarily modify a plan by utilizing the context of the plan (i.e. using pattern matching). In other words, these rules can be used to write very expressive 'self-modifying programs'. It is not clear how this can be done in IndiGolog.

2. The framework can be extended by incorporating a formal model of dynamics of intention. As mentioned above, the dynamics of intention should preserve various useful properties of intention, should be flexible enough to handle arbitrary revisions and contractions, and model the dependency between goals and sub-goals.

3. It would be interesting to develop an on-demand lookahead/planning mechanism in the modified framework with multiple goals and a set of plans. Planning/search should be not only implementable, but also be reasonably efficient. So, one question is if there is a mechanism for the user to specify the agent in a way that allows considerable pruning of the search space to ensure that planning is efficient. Also, recall that, determining rational plans is computationally intractable in S&S's framework. So limited forms of rationality can be considered as a means of increasing efficiency. In particular, some meta-level/syntactic mechanism could be developed to model the cost of deliberation, and could be used to check whether deliberation is worth the effort.

4. The framework can be extended to include communicative actions, and to model changes in agents' goals as a result of communication.

In the future, we look forward to investigate some of these issues.

References

[1] J. Ambros-Ingerson and S. Steel. Integrated Planning, Execution and Monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pp. 83–88, St. Paul, MN (1988)

[2] R. Arkin. *Behavior-Based Robotics*. MIT Press (1998)

[3] J. L. Austin. *How to Do Things with Words*. Oxford University Press, Oxford, England (1962)

[4] M. Baldoni, L. Giordano, A. Martelli, and V. Patil. Modeling Agents in a Logic Action Language. In *Proceeding of the Workshop on Rational Agents (FAPR-00)*, London, UK (2000)

[5] M. Barbuceanu, T. Gray, and S. Mankovski. Role of Obligations in Multiagent Coordination. In *Applied Artificial Intelligence*, 13(1), pp. 11–38 (1999)

[6] F. Bellifemine, F. Bergenti, G. Caire, and A. Poggi. JADE – A Java Agent Development Framework. In R. H. Bordini et al. (eds.), [9], Chapter 5, pp. 125–148 (2005)

[7] R. H. Bordini, A. L. C. Bazzan, R. O. Vicari, and V. R. Lesser. AgentSpeak(XL): Efficient Intention Selection in BDI Agents via Decision-Theoretic Task Scheduling. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, pp. 1294–1302 (2002)

[8] R. H. Bordini, L. Braubach, M. Dastani, A. El F. Seghrouchni, J. J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci. A Survey of Programming Languages and Platforms for Multi-Agent Systems. In *Informatica*, 30, pp. 33–44 (2006)

[9] R. H. Bordini, M. Dastani, J. Dix, and A. El F. Seghrouchni (eds.), *Multi-Agent Programming: Languages, Platforms and Applications*, Multiagent Systems, Artificial Societies, and Simulated Organizations Series, Vol. 15, Springer-Verlag (2005)

[10] R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model Checking AgentSpeak. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-03)*, pp. 409–416 (2003)

[11] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifiable Multi-Agent Programs. In *Proceedings of the First International Workshop on Programming Multi-Agent Systems: Languages, Frameworks, Techniques, and Tools (ProMAS-03)*, pp. 72–89 (2003)

[12] R. H. Bordini and J. F. Hübner. *Jason : A Java-based Interpreter for an Extended Version of AgentSpeak*. http://jason.sourceforge.net (2006)

[13] R. H. Bordini and A. F. Moreira. Proving BDI Properties of Agent-Oriented Programming Languages – The Asymmetry Thesis Principles in AgentSpeak(L). In *Annals of Mathematics and Artificial Intelligence*, 42(1-3), pp. 197–226. Special Issue on Computational Logic in Multi-Agent Systems (2004)

[14] R. H. Bordini, W. Visser, M. Fisher, C. Pardavila, and M. Wooldridge. Model Checking Multi-Agent Programs with CASP. In W. A. Hunt Jr. and F. Somenzi (eds.), *Proceedings of the Fifteenth Conference on Computer-Aided Verification (CAV-03)*, pp. 110–113, LNCS-2725, Springer-Verlag (2003)

[15] C. Boutilier. Toward a logic for qualitative decision theory. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR-92)*, pp. 75–86 (1992)

[16] M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA (1987)

[17] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and Resource-Bounded Practical Reasoning. In *Computational Intelligence*, 4, pp. 349–355 (1988)

[18] L. Braubach, A. Pokahr, and W. Lamersdorf. Jadex: A Short Overview. In *Proceedings of the Net.ObjectDays-04 Conference*, pp. 195–207 (2004)

[19] L. Braubach, A. Pokahr, W. Lamersdorf, and D. Moldt. Goal Representation for BDI Agent Systems. In R. H. Bordini and M. Dastani and J. Dix and A. El F. Seghrouchni (eds.), *Proceedings of the Second International Workshop on Programming Multiagent Systems, Languages, and Tools (ProMAS-04)*, pp. 9–20, LNAI-3346, Springer-Verlag (2004)

[20] R. A. Brooks. A Robust Layered Control System for a Mobile Robot. In *IEEE Journal of Robotics and Automation*, 2(1), pp. 14–23 (1986)

[21] R. A. Brooks. Elephants don't Play Chess. In P. Maes (ed.), *Designing Autonomous Agents*, pp. 3–15, The MIT Press (1990)

[22] R. A. Brooks. Intelligence without Reason. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pp. 569-595, Sydney, Australia (1990)

[23] R. A. Brooks. Intelligence without Representation. In *Artificial Intelligence*, 47, pp. 139-159 (1991)

[24] J. Carmo and A. J. I. Jones. Deontic Logic and Contrary-to-Duties. In D. Gabbay and F. Guentner (eds.), *Handbook of Philosophical Logic*, pp. 209–285 (2000)

[25] B. F. Chellas. *Modal Logic: An Introduction*. Cambridge University Press (1980)

[26] P. R. Cohen and H. J. Levesque. Intention is Choice with Commitment. In *Artificial Intelligence*, 42(2–3), pp. 213–361 (1990)

[27] P. R. Cohen and H. J. Levesque. Persistence, Intention and Commitment. In P. R. Cohen, J. Morgan, and M. E. Pollack (eds.), *Intentions in Communication*, pp. 33–69, MIT Press, Cambridge, Mass. (1990)

[28] P. R. Cohen and H. J. Levesque. Rational Interaction as the Basis for Communication. In P. R. Cohen, J. Morgan, and M. E. Pollack (eds.), *Intentions in Communication*, pp. 221–255, MIT Press, Cambridge, Mass. (1990)

[29] P. R. Cohen and H. J. Levesque. Teamwork. In *Nous*, 25(4), pp. 487–512 (1991)

[30] P. R. Cohen, H. J. Levesque, and I. Smith. On Team Formation. In J. Hintikka and R. Tuomela (eds.), *Contemporary Action Theory, Synthese* (1997)

[31] P. R. Cohen and C. R. Perrault. Elements of a Plan Based Theory of Speech Acts. In *Cognitive Science*, 3, pp. 177–212 (1979)

[32] S. Costantini and A. Tocchio. A Logic Programming Language for Multi-Agent Systems. In S. Flesca, S. Greco, N. Leone, and G. Ianni (eds.), *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA-02)*, pp. 1–13, LNAI-2424, Springer Verlag (2004)

[33] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-Directed Requirements Acquisition. In *Science of Computer Programming*, 20(1–2), pp. 3–50 (1993)

[34] M. Dastani, M. B. van Riemsdijk, F. Dignum, and J.-J. Ch. Meyer. A Programming Language for Cognitive Agents: Goal Directed 3APL. In *Proceedings of the First International Workshop on Programming Multi-Agent Systems (ProMAS-03)*, pp. 111–130, LNAI-3067 (2004)

[35] E. Davis. Knowledge Preconditions for Plans. In *Journal of Logic and Computation*, 4(5), pp. 721–766 (1994)

[36] G. De Giacomo, Y. Lespérance, and H. J. Levesque. ConGolog, A Concurrent Programming Language Based on the Situation Calculus. In *Artificial Intelligence*, 121(1–2), pp. 109–169 (2000)

[37] G. De Giacomo and H. J. Levesque. An Incremental Interpreter for High-Level Programs with Sensing. In H. J. Levesque and F. Pirri (eds.), *Logical Foundation for Cognitive Agents: Contr. in Honor of Ray Reiter*, pp. 86–102, Springer (1999)

[38] G. Delzanno and M. Martelli. Proofs as Computations in Linear Logic. In *Theoretical Computer Science*, 258(1–2), pp. 269–297 (2001)

[39] M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge. A Formal Specification of dMARS. In M. P. Singh, A. S. Rao, and M. Wooldridge (eds.), *Intelligent Agents IV–Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL-97)*, pp. 155–176, LNAI-1365, Springer-Verlag (1997)

[40] M. d'Inverno and M. Luck. Engineering AgentSpeak(L): A Formal Computational Model. In *Journal of Logic and Computation*, 8(3), pp. 1–27 (1998)

[41] J. Doyle, Y. Shoham, and M. P. Wellman. A Logic for Relative Desire. In Z. W. Ras and M. Zemankova (eds.), *Proceedings of the Sixth International Symposium on Methodologies for Intelligent Systems (ISMIS-91)*, pp. 16–31, Springer Verlag (1991)

[42] T. Eiter and V. S. Subrahmanian. Heterogeneous Active Agents, II: Algorithms and Complexity. In *Artificial Intelligence* 108 (1–2), pp. 257–307 (1999)

[43] T. Eiter, V. S. Subrahmanian, and G. Pick. Heterogeneous Active Agents, I: Semantics. In *Artificial Intelligence* 108 (1–2), pp. 179-255 (1999)

[44] E. Emerson and J. Srinivasan. Branching Time Temporal Logic. In K. de Bakker, W.-P. de Roever, and G. Rozenberg (eds.), *Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, pp. 123–172, Springer-Verlag (1989)

[45] A.K.S.I. External Interfaces Working Group. *Specifications of the KQML Agent Communication Language. Working Paper* (1993)

[46] R. Fagin and J. Y. Halpern. Belief, Awareness, and Limited Reasoning. In *Artificial Intelligence*, 34, pp. 39–76 (1988)

[47] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. The MIT Press (1995)

[48] I. A. Ferguson. Towards an Architecture for Adaptive, Rational, Mobile Agents. In E. Werner and Y. Demazeau (eds.), *Decentralized AI 3–Proceedings of the Third European Workshop on Modeling Autonomous Agents and Multi-Agent Worlds (MAAMAW-91)*, pp. 249–262 (1992)

[49] Foundations for Intelligent Physical Agents. *FIPA Request Interaction Protocol Specification*. http://www.fipa.org (1997–2002)

[50] Foundations for Intelligent Physical Agents. *FIPA Query Interaction Protocol Specification*. http://www.fipa.org (1997–2002)

[51] Foundations for Intelligent Physical Agents. *FIPA Contract Net Interaction Protocol Specification*. http://www.fipa.org (1997–2002)

[52] Foundations for Intelligent Physical Agents. *FIPA English Auction Interaction Protocol Specification*. http://www.fipa.org (1997–2002)

[53] Foundations for Intelligent Physical Agents. *FIPA Communicative Act Library Specification*. http://www.fipa.org (1997–2002)

[54] M. Fisher. Concurrent METATEM – A Language for Modeling Reactive Systems. In *Parallel Architectures and Languages, Europe (PARLE)*, pp. 185–196, LNCS-694, Munich, Germany (1993)

[55] M. Fisher. A Survey of Concurrent METATEM – The Language and Its Applications. In D. M. Gabbay and H. J. Ohlbach (eds.), *Temporal Logic - Proceedings of the First International Conference*, pp. 480–505, LNAI-827, Springer-Verlag (1994)

[56] M. Fisher and H. Barringer. Concurrent METATEM Processes – A Language for Distributed AI. In *Proceedings of the European Simulation Multiconference (ESM-91)*, Copenhagen, Denmark (1991)

[57] M. Fisher and C. Ghidini. The ABC of Rational Agent Modeling. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, pp. 849–856, Bologna, Italy (2002)

[58] M. Fisher, C. Ghidini, and B. Hirsch. Organising Logic-Based Agents. In *Proceedings of Formal Approaches to Agent-Based Systems (FAABS-02)*, pp. 15–27, LNCS-2699, Springer-Verlag (2003)

[59] M. Fisher, C. Ghidini, and B. Hirsch. Programming Groups of Rational Agents. In *Proceedings of the 4th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA-IV)*, pp. 16–33, Fort Lauderdale, FL, USA (2004)

[60] N. Fornara and M. Colombetti. Operational Specification of a Commitment-Based Agent Communication Language. In*Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-02)*, pp. 536–542 (2002)

[61] P. Gardenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press, Cambridge, Massachusetts (1988)

[62] M. P. Georgeff. Situated Reasoning and Rational Behavior. In *Pacific Rim International Conference on Artificial Intelligence* (1990)

[63] M. P. Georgeff and F. F. Ingrand. Decision-Making in an Enbedded Reasoning System. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (IJCAI-89), pp. 972–978, Detroit, MI (1989)

[64] C. Ghidini and F. Giunchiglia. Local Models Semantics, or Contextual Reasoning = Locality + Compatibility. In *Artificial Intelligence*, 127(4), pp. 221–259 (2001)

[65] B. J. Grosz and S. Kraus. Collaborative Plans for Complex Group Actions. In *Artificial Intelligence*, 86(2), pp. 269–357 (1996)

[66] B. J. Grosz and C. L. Sidner. Plans for Discourse. In P. Cohen, J. Morgan, and M. Pollack (eds.), *Intentions in Communication*, pp. 417–444, MIT Press, Cambridge, MA (1990)

[67] J. Y. Halpern and Y. Moses. Knowledge and Common Knowledge in a Distributed Environment. In *Symposium on Principles of Distributed Computing*, pp. 50–61 (1984)

[68] A. Herzig and D. Longin. A Logic of Intention with Cooperation Principles and with Assertive Speech Acts as Communication Primitives. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, pp. 920–927, Bologna, Italy (2002)

[69] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. A Formal Embedding of AgentSpeak(L) in 3APL. In *Selected Papers from the 11th Australian Joint Conference on Artificial Intelligence on Advanced Topics in Artificial Intelligence*, pp. 155–166, LNAI-1502, Springer-Verlag (1998)

[70] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent Programming in 3APL. In *International Journal of Autonomous Agents and Multi-Agent Systems*, 2(4), pp. 357–401 (1999)

[71] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent Programming with Declarative Goals. In *Proceedings of the Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL)*, Intelligent Agents VII, pp. 228–243, LNCS-1986, Springer-Verlag (2000)

[72] J. Hintikka. *Knowledge and Belief.* Cornell UP, Ithaca, N.Y. (1962)

[73] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall (1991)

[74] N. Howden, R. Rönnquist, A. Hodgson, and Andrew Lucas. JACK Intelligent Agents™– Summary of an Agent Infrastructure. In *Proceedings of the Fifth International Conference on Autonomous Agents* (2001)

[75] J. F. Hübner. *Um Modelo de Reorganização de Sistemas Multiagentes*. Ph.D. thesis, Universidade de São Paulo, Escola Politécnica (2003)

[76] N. R. Jennings. Commitments and Conventions: The Foundation of Coordination in Multi-Agent Systems. In *The Knowledge Engineering Review*, 8(3), pp. 223–250 (1993)

[77] N. R. Jennings. Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems using Joint Intentions. In *Artificial Intelligence*, 75(2), pp. 195–240 (1995)

[78] S. M. Khan and Y. Lespérance. ECASL: A Model of Rational Agency for Communicating Agents. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-05)*, pp. 762–769, Utrecht, The Netherlands (2005)

[79] S. M. Khan. *A Situation Calculus Account of Multi-Agent Planning, Speech-Acts, and Communication*. M.Sc. Thesis, Dept. of Computer Science and Engineering, York University, Toronto, ON, Canada (to appear)

[80] D. Kinny. *The Distributed Multi-Agent Reasoning System Architecture and Language Specification*. Technical Report, Australian Artificial Intelligence Institute, Melbourne, Australia (1993)

[81] K. Konolige and M. E. Pollack. A Representationalist Theory of Intention. In *Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pp. 390–395, Chambéry, France (1993)

[82] S. Kripke. Semantical Analysis of Modal Logic I. Normal Propositional Calculi. In *Zeitschrift fur math. Logik und Grundlagen der Mathematik*, 9, pp. 67–96 (1963)

[83] S. Kripke. Semantical Considerations on Modal Logic. In *Acta Philosophica Fennica*, 16, pp. 83–94 (1963)

[84] S. Kripke. Semantical Analysis of Modal Logic II. Non-Normal Propositional Calculi. In J. W. Addison, L. Henkin, and A. Tarski (eds.), *The Theory of Models*, pp. 206–220 (1965)

[85] S. Kumar, M. J. Huber, D. McGee, P. R. Cohen, and H. J. Levesque. Semantics of Agent Communication Languages for Group Interaction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI-00)*, pp. 42–47 (2000)

[86] J. E. Laird, A. Newell, and P. S. Rosenbloom. SOAR: An Architecture for General Intelligence. In *Artificial Intelligence*, 33, pp. 1–64 (1987)

[87] J. A. Leite. *Evolving Knowledge Bases*, Frontiers in Artificial Intelligence and Applications-81, IOS Press (2003)

[88] J. A. Leite, J. J. Alferes, and L. M. Pereira. MINERVA – A Dynamic Logic Programming Agent Architecture. In J.-J. Meyer and M. Tambe (eds.), *Proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages (ATAL)*, Intelligent Agents VIII, pp. 141–157, LNAI-2333, Springer-Verlag (2002)

[89] E. Letier and A. van Lamsweerde. Deriving Operational Software Specifications from System Goals. In *Proceedings of the 10th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 119–128, ACM Press (2002)

[90] Y. Lespérance. On the Epistemic Feasibility of Plans in Multiagent Systems Specifications. In J.-J. C. Meyer and M. Tambe (eds.), *Proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages (ATAL)*, Intelligent Agents VIII, pp. 69–85, LNAI-2333, Springer-Verlag (2001)

[91] Y. Lespérance, H. J. Levesque, F. Lin, and R. Scherl. Ability and Knowing How in the Situation Calculus. In *Studia Logica*, 66(1), pp. 165–186 (2000)

[92] Y. Lespérance and H.-K. Ng. Integrating Planning into Reactive High-Level Robot Programs. In *Proceedings of the Second International Cognitive Robotics Workshop*, pp. 49–54, Berlin, Germany (2000)

[93] H. J. Levesque. What is Planning in the Presence of Sensing? In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 1139–1146, Portland, OR (1996)

[94] H. J. Levesque and T. Barrie. Joint Ability (unpublished)

[95] H. J. Levesque, P. R. Cohen, and J. T. Nunes. On Acting Together. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, pp. 94–99, San Mateo, California, Morgan Kaufmann Publishers, Inc. (1990)

[96] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. Golog: A Logic Programming Language for Dynamic Domains. In *Journal of Logic Programming*, 31, pp. 59–84 (1997)

[97] V. Louis. *Conception et Mise en Oeuvre de Modèles Formels de Calcul de Plans d'Action Complexes par un Agent Rationnel Dialoguant*. Ph.D. thesis, Université de Caen, Caen, France (2002)

[98] J. Malec. A Unified Approach to Intelligent Agency. In M. Wooldridge and N. R. Jennings (eds.), *Intelligent Agents: Theories, Architectures, and Languages*, pp. 233–244, LNAI-890, Springer-Verlag, Germany (1995)

[99] V. Mascardi, M. Martelli, and L. Sterling. Logic-Based Specification Languages for Intelligent Software Agents. In *Theory and Practice of Logic Programming*, 4(4), pp. 429–494, Cambridge University Press, New York, NY, USA (2004)

[100] M. J. Matarić. Behavior-Based Robotics as a Tool for Synthesis of Artificial Behavior and Analysis of Natural Behavior. In *Trends in Cognitive Science*, 2(3), pp. 82–87 (1998)

[101] F. McCabe and K. Clark. APRIL: Agent Process Interaction Language. In M. Wooldridge and N. R. Jennings (eds.), *Intelligent Agents: Theories, Architectures, and Languages*, pp. 324–340, LNAI-890, Springer-Verlag, Germany (1995)

[102] J. McCarthy and P. J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In *Machine Intelligence*, 4, pp. 463–502 (1969)

[103] J.-J.Ch. Meyer, W. van der Hoek, and B. van Linder. A Logical Approach to the Dynamics of Commitments. In *Artificial Intelligence*, 113(1–2), pp. 1–40 (1999)

[104] S. Minton, C. Knoblock, D. Kuokka, Y. Gil, R. Joseph, and J. Carbonell. *PRODIGY 2.0: The Manual and Tutorial*. Technical Report CMU-CS-89-146, Carnegie Mellon University (1989)

[105] R. C. Moore. A Formal Theory of Knowledge and Action. In J. R. Hobbs and R. C. Moore (eds.), *Formal Theories of the Commonsense World*, pp. 319–358, Ablex (1985)

[106] R. C. Moore. A Formal Theory of Knowledge and Action. In J. F. Allen, J. Hendler, and A. Tate (eds.), *Readings in Planning*, pp. 480–519, Morgan Kaufmann Publishers, San Mateo, CA (1990)

[107] À. F. Moreira and R. H. Bordini. An Operational Semantics for a BDI Agent-Oriented Programming Language. In J.-J. Ch. Meyer and M. J. Wooldridge (eds.), *Proceedings of the Workshop on Logics for Agent-Based Systems (LABS-02)*, pp. 45–59 (2002)

[108] À. F. Moreira, R. Vieira, and R. H. Bordini. Extending the Operational Semantics of a BDI Agent-Oriented Programming Language for Introducing Speech-Act based Communication. In *Proceedings of the Workshop on Declarative Agent Languages and Technologies (DALT-03)*, pp. 135–154, LNAI-2990, Springer-Verlag (2003)

[109] Y. Moses and M. Tennenholtz. Artificial Social Systems. In *Computers and AI*, 14(6), pp. 533–562 (1995)

[110] J. P. Müller, M. Pischel, and M. Thiel. Modeling Reactive Behavior in Vertically Layered Agent Architectures. In M. Wooldridge and N. R. Jennings (eds.), *Intelligent Agents: Theories, Architectures, and Languages*, pp. 261–276, LNAI-890, Springer-Verlag, Germany (1995)

[111] A. Newell and H. A. Simon. Computer Science as Empirical Enquiry. In *Communications of the ACM*, 19, pp. 113–126 (1976)

[112] A. Omicini and E. Denti. From Tuple Spaces to Tuple Centres. In *Science of Computer Programming*, 41(3), pp. 277–294 (2001)

[113] L. Padgham and M. Winikoff. Prometheus: A Methodology for Developing Intelligent Agents. In *Proceedings of the Third International Workshop on Agent-Oriented Software Engineering (AOSE-02)*, pp. 174–185, Bologna, Italy (2002)

[114] M. Pauly. A Logical Framework for Coalitional Effectivity in Dynamic Procedures. In *Bulletin of Economic Research*, 53(4), pp. 305–324 (2002)

[115] M. Pauly. A Modal Logic for Coalitional Power in Games. In *Journal of Logic and Computation*, 12(1), pp. 149-166 (2002)

[116] R. C. Perrault. An Application of Default Logic to Speech Act Theory. In P. R. Cohen, J. Morgan, and M. E. Pollack (eds.), *Intentions in Communication*, pp. 161–185, MIT Press, Cambridge, Mass. (1990)

[117] R. C. Perrault and J. F. Allen. A Plan-Based Analysis of Indirect Speech Acts. In *American Journal of Computational Linguistics*, 6(3–4), pp. 167–182 (1980)

[118] G. Plotkin. *A Structural Approach to Operational Semantics*. Technical Report DAIMI-FN-19, Computer Science Dept., Aarhus University, Denmark (1981)

[119] A. Poggi. DAISY: An Object-Oriented System for Distributed Artificial Intelligence. In M. Wooldridge and N. R. Jennings (eds.), *Intelligent Agents: Theories, Architectures, and Languages*, pp. 341–354, LNAI-890, Springer-Verlag, Germany (1995)

[120] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI Reasoning Engine. In R. H. Bordini et al. (eds.), [9], Chapter 6, pp. 149–174 (2005)

[121] J. L. Pollock. *Cognitive Carpentry: A Blueprint for How to Build a Person*. The MIT Press, Cambridge, MA (1995)

[122] A. S. Rao. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In W. V. Velde and J. W. Perram (eds.), *Agents Breaking Away*, pp. 42–55, LNAI-1038, Springer-Verlag (1996)

[123] A. S. Rao and M. P. Georgeff. Asymmetry Thesis and Side-Effect Problems in Linear Time and Branching Time Intention Logics. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pp. 498–504, Sydney, Australia (1991)

[124] A. S. Rao and M. P. Georgeff. Modeling Rational Agents with a BDI-Architecture. In J. F. Allen, R. Fikes, and E. Sandewall (eds.), *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR-91)*, pp. 473–484, San Mateo, CA, Morgan Kaufmann Publishers (1991)

[125] A. S. Rao and M. P. Georgeff. An Abstract Architecture for Rational Agents. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR-92)*, pp. 439–449, Cambridge, Massachusetts, USA (1992)

[126] R. Reiter. The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression. In V. Lifschitz (ed.), *Artificial Intelligence and Mathematical Theory of Computation: Papers in the Honor of John McCarthy*, pp. 359–380, Academic Press, San Diego, CA, USA (1991)

[127] R. Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press (2001)

[128] S. Rosenschein and L. P. Kaelbling. The Synthesis of Digital Machines with Provable Epistemic Properties. In J. Y. Halpern (ed.), *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning about Knowledge*, pp. 83–98, Morgan Kaufmann Publishers: San Mateo, CA (1986)

[129] M. D. Sadek. A Study in the Logic of Intention. In B. Nebel, C. Rich, and W. R. Swartout (eds.), *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR-92)*, pp. 462–473, Cambridge, MA (1992)

[130] M. D. Sadek. Communication Theory = Rational Principles + Communicative Act Models. In *Proceedings of the AAAI Workshop on Planning for Interagent Communication* (1994)

[131] M. D. Sadek, P. Bretier, and E. Panaget. ARTIMIS: Natural Dialogue Meets Rational Agency. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 1030–1035, Nagoya, Japan (1997)

[132] M. D. Sadek, A. Ferrieux, A. Cozannet, P. Bretier, F. Panaget, and J. Simonin. Effective Human-Computer Cooperative Spoken Dialogue: The AGS Demonstrator. In *Proceedings of the Fourth International Conference on Spoken Language Processing (ICSLP-96)*, pp. 546–549, Philadelphia, PA (1996)

[133] S. Sardiña, G. De Giacomo, Y. Lespérance, and H. J. Levesque. On the Semantics of Deliberation in IndiGolog – From Theory to Implementation. In *Annals of Mathematics and Artificial Intelligence*, 41(2–4), pp. 259–299 (2004)

[134] S. Sardiña, L. de Silva, and L. Padgham. Hierarchical Planning in BDI Agent Programming Languages: A Formal Approach. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-06)*, pp. 1001–1008, Hakodate, Japan (2003)

[135] S. Sardiña and S. Shapiro. Rational Action in Agent Programs with Prioritized Goals. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-03)*, pp. 417–424, Melbourne, Australia (2003)

[136] R. B. Scherl and H. J. Levesque. The Frame Problem and Knowledge-Producing Actions. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 689–695, Washington, DC, AAAI Press/The MIT Press (1993)

[137] R. B. Scherl and H. J. Levesque. Knowledge, Action, and the Frame Problem. In *Artificial Intelligence*, 144(1–2), pp. 1–39 (2003)

[138] J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press (1969)

[139] J. R. Searle. Collective Intentions and Actions. In P. Cohen, J. Morgan, and M. Pollack (eds.), *Intentions in Communication*, pp. 401–415, MIT Press, Cambridge, MA (1990)

[140] A. El F. Seghrouchni and A. Suna. CLAIM: A Computational Language for Autonomous, Intelligent and Mobile Agents. In M. Dastani, J. Dix, and A. El F. Seghrouchni (eds.), *Proceedings of the First International Workshop on Programming Multi-Agent Systems: Languages, Frameworks, Techniques, and Tools (ProMAS-03)*, pp. 90–110, LNCS-3067, Springer Verlag (2004)

[141] S. Shapiro. *Specifying and Verifying Multi-Agent Systems using the Cognitive Agents Specification Language (CASL)*. Ph.D. thesis, University of Toronto, Toronto, ON, Canada (2004)

[142] S. Shapiro. Belief Change with Noisy Sensing and Introspection. In L. Morgenstern and M. Pagnucco (eds.), *Working Notes of the IJCAI-05 Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC-05)*, pp. 84–89 (2005)

[143] S. Shapiro and Y. Lespérance. Modeling Multiagent Systems with the Cognitive Agent Specification Language – A Feature Interaction Resolution Application. In C. Castelfranchi and Y. Lespérance (eds.), *Proceedings of The Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL-00)*, Intelligent Agents-VII, pp. 244–259, LNAI-1986, Boston, USA (2001)

[144] S. Shapiro, Y. Lespérance, and H. J. Levesque. Goals and Rational Action in the Situation Calculus - A Preliminary Report. In *Working Notes of the AAAI Fall Symposium on Rational Agency: Concepts, Theories, Models, and Applications*, pp. 117–122, Cambridge, MA, USA (1995)

[145] S. Shapiro, Y. Lespérance, and H. J. Levesque. Specifying Communicative Multi-Agent Systems with ConGolog. In *Working Notes of the AAAI Fall 1997 Symposium on Communicative Action in Humans and Machines*, pp. 75–82, Cambridge, MA, USA, AAAI Press (1997)

[146] S. Shapiro, Y. Lespérance, and H. J. Levesque. The Cognitive Agents Specification Language and Verification Environment for Multiagent Systems. In C. Castelfranchi and W. Lewis Johnson (eds.), *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-02)*, pp. 19–26, Bologna, Italy, ACM Press (2002)

[147] S. Shapiro, Y. Lespérance, and H. J. Levesque. Goal Change. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pp. 582–588, Edinburgh, Scotland (2005)

[148] S. Shapiro, M. Pagnucco, Y. Lespérance, and H. J. Levesque. Iterated Belief Change in the Situation Calculus. In A. G. Cohn, F. Giunchiglia, and B. Selman (eds.), *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR-00)*, pp. 527–538, San Francisco, CA, Morgan Kaufmann Publishers (2000)

[149] Y. Shoham. AGENT0: A Simple Agent Language and Its Interpreter. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pp. 704–709, AAAI Press/MIT Press, Anaheim, California, USA (1991)

[150] Y. Shoham. Agent-Oriented Programming. In *Artificial Intelligence*, 60(1), pp. 51–92 (1993)

[151] Y. Shoham and M.Tennenholtz. On Social Laws for Artificial Agent Societies: Off-Line Design. In *Artificial Intelligence*, 73(1–2), pp. 231–252 (1995)

[152] M. P. Singh. Group Intentions. In *10th International Workshop on Distributed Artificial Intelligence (IWDAI-90)*, Texas, USA (1990)

[153] M. P. Singh. A Logic of Situated Know-How. In T. Dean and K. McKeown (eds.), *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pp. 343–348, AAAI Press (1991)

[154] M. P. Singh. Group Ability and Structure. In *Decentralized AI 2 – Proceedings of the 2nd European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pp. 127–146, France (1991)

[155] M. P. Singh. A Critical Examination of the Cohen-Levesque Theory of Intention. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92)*, pp. 364–368, Vienna, Austria (1992)

[156] M. P. Singh. A Semantics for Speech Acts. In *Annals of Mathematics and Artificial Intelligence*, 8(1–2), pp. 47–71 (1993)

[157] M. P. Singh. *Intentions for Multiagent Systems*. Technical Report KBNL-086-93, Austin, TX (1993)

[158] M. P. Singh. *Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communications*. LNAI-799, Springer-Verlag, Heidelberg, Germany (1994)

[159] M. P. Singh. A Social Semantics for Agent Communication Languages. In *Proceedings of the IJCAI-99 Workshop on Agent Communication Languages*, Issues in Agent Communication, pp. 31–45, LNAI-1916, Springer Verlag (2000)

[160] M. Tambe. Towards Flexible Teamwork. In *Journal of Artificial Intelligence Research*, 7, pp. 83–124 (1997)

[161] M. Tan and R. Weihmayer. Integrating Agent-Oriented Programming and Planning for Cooperative Problem Solving. In *Proceedings of the AAAI-92 Workshop on Cooperation among Heterogeneous Intelligent Systems* (1992)

[162] M. Thielscher. FLUX: A Logic Programming Method for Reasoning Agents. In *Theory and Practice of Logic Programming*, 5(4–5), pp. 533–565, Cambridge University Press (2005)

[163] S. R. Thomas. *PLACA, An Agent-Oriented Programming Language*. Ph.D. Thesis (technical report STAN-CS-93-1487), Computer Science Department, Stanford University, Stanford, CA, USA (1993)

[164] S. R. Thomas. The PLACA Agent Programming Language. In M. Wooldridge and N. R. Jennings (eds.), *Intelligent Agents: Theories, Architectures, and Languages*, pp. 355–369, LNAI-890, Springer-Verlag, Germany (1995)

[165] W. van der Hoek, B. van Linder, and J.-J. C. Meyer. *A Logic of Capabilities*. Technical Report IR-330, Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands (1995)

[166] W. van der Hoek and M. Wooldridge. Cooperation, Knowledge, and Time: Alternating-Time Temporal Epistemic Logic and its Application. In *Studia Logica*, 75(1), pp. 125–157 (2003)

[167] A. van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *Proceedings the International Joint Conference on Requirements Engineering*, pp. 249–263, IEEE (2001)

[168] B. van Linder, W. van der Hoek, and J.-J. C. Meyer. *Tests as Epistemic Updates–Pursuit of Knowledge*. Technical Report UU-CS-1994-08, Dept. of Computer Science, Utrecht University, Utrecht, The Netherlands (1994)

[169] B. van Linder, W. van der Hoek, and J.-J. Ch. Meyer. Actions That Make You Change Your Mind. In *Knowledge and Belief in Philosophy and AI*, pp. 103–146 (1995)

[170] B. van Linder, W. van der Hoek, and J.-J. Ch. Meyer. Formalising Motivational Attitudes of Agents : On Preferences, Goals, and Commitments. In M. Wooldridge, J. Muller, and M. Tambe (eds.), *Proceedings of the Agent Theories, Architectures, and Languages Workshop (ATAL-95)*, Intelligent Agents-II, pp. 17–32, LNAI-1037, Springer-Verlag, Montréal, Canada (1995)

[171] M. B. van Riemsdijk. *Cognitive Agent Programming : A Semantic Approach*. Ph.D. Thesis, Department of Information and Computing Sciences, Universiteit Utrecht, SIKS Dissertation Series, No. 2006-19 (2006)

[172] M. B. van Riemsdijk, M. Dastani, F. Dignum, and J.-J. Ch. Meyer. Dynamics of Declarative Goals in Agent Programming. In *Proceedings of the 2nd International Workshop on Declarative Agent Languages and Technologies (DALT-04)*, pp. 1–18, LNCS-3476, Springer-Verlag, New York, NY, USA (2004)

[173] M. B. van Riemsdijk, M. Dastani, and J.-J. Ch. Meyer. Semantics of Declarative Goals in Agent Programming. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-05)*, pp. 133–140, Utrecht, The Netherlands (2005)

[174] M. B. van Riemsdijk, W. van der Hoek, and J.-J. Ch. Meyer. Agent Programming in Dribble: from Beliefs to Goals using Plans. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-03)*, pp. 393–400 (2003)

[175] S. Vassos and H. J. Levesque. Progression of Situation Calculus Action Theories with Incomplete Information. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pp. 2029–2034, Hyderabad, India (2007)

[176] S. Vassos, S. Sardiña, and H. J. Levesque. A Feasible Approach to Disjunctive Knowledge in the Situation Calculus. (draft)

[177] M. Verdicchio and M. Colombetti. A Logical Model of Social Commitment for Agent Communication. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-03)*, pp. 528–535, Melbourne, Australia (2003)

[178] S. Vere and T. Bickmore. A Basic Agent. In *Computational Intelligence*, 6, pp. 41–60 (1990)

[179] R. Vieira, A. Moreira, M. Wooldridge, and R. H. Bordini. On the Formal Semantics of Speech-Act Based Communication in an Agent-Oriented Programming Language (unpublished)

[180] W. Visser, K. Havelund, G. Brat, and S. Park. Model Checking Programs. In *Proceedings of the Fifteenth International Conference on Automated Software Engineering (ASE-00)*, pp. 3-12, IEEE Computer Society (2000)

[181] M. P. Wellman and J. Doyle. Preferential Semantics for Goals. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pp. 698–703, Anaheim, California, USA (1991)

[182] E. Werner. Toward a Theory of Communication and Cooperation for Multiagent Planning. In *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge (TARK-88)*, pp. 129–143, Morgan Kaufman (1988)

[183] E. Werner. Cooperating Agents : A Unified Theory of Communication and Social Structure. In *Distributed Artificial Intelligence*, 2, pp. 3–36 (1989)

[184] E. Werner. What Can Agents Do Together? A Semantics for Reasoning about Cooperative Ability. In *Proceedings of the 9th European Conference on Artificial Intelligence (ECAI-90)*, pp. 694–701, Stockholm, Sweden (1990)

[185] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and Procedural Goals in Intelligent Agent Systems. In *Proceedings of the 8th International Conference on Principles and Knowledge Representation and Reasoning (KR-02)*, pp. 470–481, Toulouse, France (2002)

[186] S. Wood. *Planning and Decision Making in Dynamic Domains*, Ellis Horwood Ltd. (1993)

[187] M. Wooldridge. *Reasoning about Rational Agents*. MIT Press (2000)

[188] M. Wooldridge and N. R. Jennings. Formalizing the Cooperative Problem Solving Process. In *Proceedings of the 13th International Workshop on Distributed Artificial Intelligence (IWDAI-94)*, pp. 403–417, WA, USA (1994)

[189] M. Wooldridge and N. R. Jennings. Agent Theories, Architectures, and Languages: A Survey. In *ECAI Workshop on Agent Theories, Architectures, and Languages (ATAL-94)*, pp. 1–39, LNCS-890, Springer-Verlag (1995)

[190] M. Wooldridge and N. R. Jennings. The Cooperative Problem Solving Process. In *Journal of Logic and Computation*, 9(4), pp. 563–592 (1999)

[191] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. In *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3), pp. 285–312 (2000)

[192] M. Wooldridge and A. Rao. *Foundation of Rational Agency*. Kluwer Academic Publishers (1999)

[193] P. Yolum and M. P. Singh. Commitment machines. In *Proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages (ATAL-01)*, pp. 235–247, LNCS-2333, Springer-Verlag (2002)