

Chapter

Policy-based Agent Directability

KAREN L. MYERS and DAVID N. MORLEY

Artificial Intelligence Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025

Keywords: adjustable autonomy, advisable systems, mixed-initiative, agents

Abstract: Many potential applications for agent technology require humans and agents to work together to achieve complex tasks effectively. In contrast, most of the work in the agents community to date has focused on technologies for fully autonomous agent systems. This paper presents a framework for the *directability* of agents, in which a human supervisor can define policies to influence agent activities at execution time. The framework focuses on the concepts of *adjustable autonomy* for agents (i.e., varying the degree to which agents make decisions without human intervention) and *strategy preference* (i.e., recommending how agents should accomplish assigned tasks). These mechanisms enable a human to customize the operations of agents to suit individual preferences and situation dynamics, leading to improved system reliability and increased user confidence over fully automated agent systems. The directability framework has been implemented within a BDI environment, and applied to a multiagent intelligence-gathering domain.

1. INTRODUCTION

The technical and public press are filled these days with visions of a not-too-distant future in which humans rely on software and hardware agents to assist with tasks in environments both physical (e.g., smart homes and offices) and virtual (e.g., the Internet). The notion of *delegation* plays a central role in these visions, with a human off-loading responsibilities to agents that can perform activities in his place. Successful delegation, however, requires more than the mere assignment of tasks. A good manager generally provides directions to subordinates so that tasks are performed to his liking. To ensure effectiveness, the manager will monitor the progress of subordinates, interrupting occasionally to provide advice or resolve problems. Analogously, effective delegation of tasks to intelligent agents will require tools by which a human supervisor can interact with agents and direct their operations.

The agents research community has, for the most part, focused on the mechanics of building autonomous agents and techniques for communication and coordination among agents. In contrast, little attention has been paid to supporting human interactions with agents. Most agent frameworks lie at the extremes of the interaction spectrum, either assuming full automation by the agents with no means for user involvement, or requiring human intervention at each step along the way (i.e., *teleoperation* mode). Recently, however, there has been increased interest in agent systems designed specifically to

support interaction with humans (e.g., (Bonasso, 1999; Chalupsky et al., 2001; Ferguson and Allen, 1998; Schreckenghost et al., 2001)).

We describe a framework, called Taskable Reactive Agent Communities (TRAC), that supports the directability of agents by a human supervisor. Within TRAC, the human assigns tasks to agents along with guidance that imposes boundaries on agent behavior. By adding, deleting, or modifying guidance at his discretion, the human can manage agent activity at a level of involvement that suits his needs. In this way, guidance enables flexible human control of agent behavior.

A key issue in developing technology for agent directability is determining the types of guidance to be provided. This paper focuses on guidance for *adjustable agent autonomy* and *strategy preferences*. Guidance for adjustable autonomy enables a supervisor to vary the degree to which agents can make decisions without human intervention. Guidance for strategy preferences constitutes recommendations on how agents should accomplish assigned tasks. Effective delegation and management by a human supervisor also requires visibility into ongoing agent operations. Although not described in this paper, the TRAC framework includes a capability for *customizable reporting* that enables a supervisor to tailor the amount, type, and frequency of information produced by agents to meet his evolving needs. Details can be found in (Myers and Morley 2001).

The main contributions of this paper are the characterization of guidance for adjustable autonomy and strategy preference, a formal language for representing such guidance, a semantic model that defines satisfaction of guidance by an agent, and techniques for enforcing guidance during agent operation.

We begin with a description of our underlying agent model (Section 2), followed by an informal characterization of guidance for adjustable autonomy and strategy preferences (Section 3). Next, we describe the TIGER system, in which guidance can be used to direct agents as they perform intelligence gathering tasks in the wake of a simulated natural disaster (Section 4). We use TIGER throughout this document to provide examples of agent directability. Following this description, we present our formal representation for guidance (Section 5) and a semantic model for guidance satisfaction (Section 6). Section 7 presents our techniques for guidance enforcement, while Section 8 describes interface tools that support guidance specification and Section 9 discusses related work. Section 10 concludes with directions for future work.

2. AGENT MODEL

We adopt a typical Belief-Desire-Intention (BDI) model of agency in the style of (Rao and Georgeff, 1995). BDI agents are so-called due to the three components of their “mental state”: *beliefs* that an agent has about the state of the world, *desires* to be achieved, and *intentions* corresponding to plans that an agent has adopted to achieve its desires.

2.1 Agent Components

An agent represents the domain using a standard first-order language; *well-formed formulae* (*wffs*) are constructed from variables, quantifiers, connectives, and domain-specific predicate, function, and constant symbols.

The *beliefs* of an agent are represented by a set, *Bel*, of ground atomic facts. Given a set of beliefs, we define the truth of a wff f (denoted by $Bel ? f$) as follows. A ground atomic fact f is interpreted as true with respect to *Bel* iff $f \in Bel$ and false otherwise (i.e., the Closed World Assumption). The truth of a compound formula is derived from the truth of its constituents in the standard way.

The *desires* of an agent are represented by *goals* that denote conditions to be achieved. Syntactically, goals are constructed from goal name symbols and terms. As the BDI executor achieves a goal, it may bind variables in the goal, effectively returning values that result from goal achievement. Agents manipulate their world by performing *actions*. Syntactically, actions are constructed from action name symbols and ground terms.

Each agent has a library of *plans* that describe alternative ways of achieving a goal or responding to a change in the belief state; our plan model is based on the Act representation language (Wilkins and Myers, 1995). Plans are parameterized templates of activities that may require variable instantiations to apply to a particular situation. Each plan has a *precondition*, consisting of a wff stating conditions under which the plan can be used. The *cue* of a plan specifies a stimulus that activates the plan. *Fact-invoked* plans have a wff for the cue and are triggered by the agent's beliefs changing to make that wff true; *goal-invoked* plans have a goal for the cue and are triggered by the posting of a unifying goal expression. The *body* of a plan specifies how to respond to the stimulus; it consists of a directed graph of actions to perform and subgoals to achieve.

An agent's plan library will generally contain a range of plans describing alternative responses to posted goals or events. Certain of these plans may be *operationally equivalent*, meaning that they share the same cue and preconditions but differ in the approach that they embody. To select among these alternatives, an agent may have some form of meta-control policy, such as user guidance.

A *plan instance* is a copy of a plan with some substitution of terms for plan variables. A plan instance represents a possible way of responding to a triggering event (belief change or posted goal). The *relevant* plans for an event consist of the plans in the library whose cue unifies with the event. The *applicable* plan instances for an event consist of instances of the relevant plans created by applying variable substitutions that unify the cue with the event, provided that the plan's precondition is true with respect to the agent's beliefs.

2.2 Agent Execution

A BDI executor selects plan instances to execute in response to changes in its beliefs and goals. An *intended plan* consists of a plan instance that the agent has decided to execute, together with information about the progress of that execution: the node in the body that is currently being executed and variable bindings that have been introduced through the execution. Intended plans exist as part of an *intention*, a hierarchical structure corresponding to an execution thread. The root of each intention consists of an intended plan that resulted from a belief change or from a goal supplied by the user. Down the intention hierarchy, the cue of each intended plan matches a goal in the intended plan above it. The *intention set*, *Int*, of a BDI agent is defined to be the set of all intentions that the agent is executing.

A BDI executor runs a continuous *sense-decide-act* loop. At the beginning of each cycle, the executor updates the agent's beliefs based on sensor information, and posts a belief change event for each modification. In addition, the executor posts a goal event for

each new goal submitted by the user. The executor then selects an intention and identifies the *current node* to be considered (either a goal or an action) in the body of the lowest-level intended plan of that intention. If the current node is an action, the action is attempted and any variable bindings that result from the successful execution of the action are applied to the intended plans of the selected intention. Otherwise, the current node is a goal and the agent posts a corresponding goal event.

For each posted event, the executor collects the applicable plan instances and selects one to be *intended*. If the event is the posting of a goal from an existing intention i , then intending extends i to include the selected plan instance for the goal. For other events (user-specified goal or belief change), a new intention is created for the selected plan instance. When the last goal or action of an intended plan completes successfully, the intended plan is dropped from the intention, and any goal that triggered it is deemed completed.

The selection of an applicable plan instance to intend for an event depends on the *BDI executor state*, which contains the mental state of the agent (i.e., the beliefs, desires, and intentions) along with the selected event. In this document, we focus on plan instance selection for a goal event, (i.e., the ‘current goal’, g^{cur}); plan instance selection for belief change events can be treated similarly. We denote such a BDI executor state by the tuple $S = \langle Bel, Des, Int, g^{cur} \rangle$.

Within this model of BDI execution, agents make three classes of decision:

- D1** *whether to respond to a new goal or event*
- D2** *how to select among multiple applicable plans when expanding a goal*
- D3** *how to select instantiations for plan variables*

3. TRAC FRAMEWORK FOR AGENT DIRECTABILITY

Our directability framework assumes that agents are capable of fully autonomous operation. More concretely, an agent's plan library covers the range of activities required to perform its assigned tasks. This assumption means that agents do not depend on the human supervisor to provide knowledge for task execution. Within this setting, guidance provides customization of agent behavior to suit the preferences of the human supervisor. In many applications, such guidance will enable superior performance, given that few plan libraries will reflect the full experience, breadth of knowledge, and reasoning capabilities that a human supervisor can bring to the decision-making process.

Our model of agent directability focuses on general and situation-specific policies for influencing the activities undertaken by agents in the execution of assigned tasks. In particular, we emphasize the areas of (a) *adjustable levels of agent autonomy*, and (b) *strategy preferences*.

3.1 Adjustable Autonomy

We define the autonomy of an agent to be the extent to which it is allowed to make decisions (specifically, D1–D3) on its own. In situations where activities are routine and decisions straightforward, a human may be content to delegate all problem-solving responsibility to an agent. However, in situations where missteps could have severe consequences, the degree of autonomy of an individual agent should necessarily be controllable by a human.

We are interested in domains where agents will generally need to operate with high degrees of autonomy. For this reason, we assume a *permissive* environment: unless stated otherwise, agents are allowed to operate independent of human interaction. Our approach allows the human to adjust the scope of operations that can be undertaken by an agent on its own terms, focusing on the notions of *permission requirements* for action execution and *consultation requirements* for decision making.

Permission Requirements: Permission requirements declare conditions under which an agent must elicit authorization from the human supervisor before executing actions. For example, the directive “Obtain permission before abandoning survey tasks with Priority > 3” imposes the constraint that an agent request approval from the supervisor to abandon a certain class of tasks.

Consultation Requirements: Consultation requirements designate a class of agent decisions that should be deferred to the human supervisor. These decisions can relate to either the selection of a value for variable instantiation (e.g., “Consult when selecting locations for staging bases”) or the selection of a plan for a goal (e.g., “Consult when choosing a response to a failed survey task”).

Our model of permission and consultation requirements, like earlier work on authority models, provides the means to block performance of certain actions by an agent. However, authority models are generally static (e.g., the *levels of autonomy* in (Bonasso 1999)) and often derived from organizational structures. In contrast, our approach provides a rich language for expressing permission and consultation policies, which can vary throughout a problem-solving session.

3.2 Strategy Preference

Strategy preferences express recommendations on how an agent should accomplish tasks. These preferences could indicate specific plans to employ or restrictions on plans that should not be employed, as well as constraints on how plan variables can be instantiated.

For example, the directive “Try contacting Nongovernmental Organizations for information before sending vehicles to towns on the west coast” expresses a preference for selecting among operationally equivalent plans. On the other hand, the directive “Only use helicopters for survey tasks in sectors that are expected to be inaccessible by truck for more than 1 week” restricts the choice of resource type for instantiating certain plan variables.

4. THE TIGER SYSTEM

We have developed a prototype implementation of our TRAC framework for agent guidance on top of the Procedural Reasoning System (PRS) (Georgeff and Ingrand 1989). The TRAC implementation has been used as the basis for a demonstration system called TIGER (TRAC Intelligence Gathering and Emergency Response), which provides a testbed for exploring our ideas on agent directability. Within TIGER, a human supervisor can delegate tasks to agents while providing guidance to control their runtime behavior.

4.1 TIGER Functionality

TIGER serves as part of a *disaster response team* whose objective is to provide humanitarian relief in the wake of a natural disaster. Other organizations within the team provide logistics (e.g., supplies distribution), operations (e.g., repair of infrastructure), and medical services. These organizations have their own physical assets (trucks and aircraft) available for their use. As would be expected, these organizations need to share information and resources to perform their functions effectively. A human supervisor oversees operations, dynamically tasking organizations to implement the relief process.¹

The primary role for TIGER is to gather information in response to requests from the supervisor or other members of the disaster response team. These requests can result in tasks to acquire information on the current state of infrastructure (roads, bridges) in designated regions, or to collect supply requirements (medical, food, water, shelter) of designated population centers within impacted regions. There can also be requests to be informed of key events (e.g., medical emergencies) as they become known. A secondary role is to respond to certain unexpected events (e.g., participate in evacuations, assist with medical emergencies). Thus, TIGER agents must incorporate reactive capabilities that balance responsiveness with ongoing goal attainment.

The scope and complexity of the intelligence-gathering operations within the disaster relief context preclude step-by-step management of agent operations by a human. However, effective coordination of the available assets requires human supervision. As such, this domain provides an excellent example of an application that will benefit from technology for agent directability.

4.2 Agent Community Organization

Figure 1 displays the organization of agents within TIGER. The system has at its disposal a collection of simulated *physical agents* (trucks and helicopters) that can be used to gather information and respond to emergencies. In addition, there is a set of simulated *communications agents* (other relief organizations, nongovernmental organizations, local officials) that can be consulted to obtain information. TIGER contains a separate controller for each of the physical agents, as well as a communications manager for interacting with the various communications agents. We refer to these controller agents as the *task execution agents* within TIGER, because they instigate and manage the activities required to perform assigned tasks. The *coordinator agent* provides global management of tasks within the community, acting as a mediator between the human supervisor and the task execution agents. It also manages interactions with members of the disaster response team who request information (i.e., its *information clients*).

4.3 Tasking Model

The TIGER coordinator agent maintains a pool of unassigned tasks and a pool of currently unallocated agents. It matches a waiting task with an unallocated agent based on properties of the task, the available agents, and current knowledge about the state of the roads and bridges. Task properties include *location*, *priority* (an integer from 0 to 10), *type* (e.g., survey, rescue), and *status* (e.g., pending, completed, failed). The agent properties include *type* (e.g., helicopter or truck) and *location*.

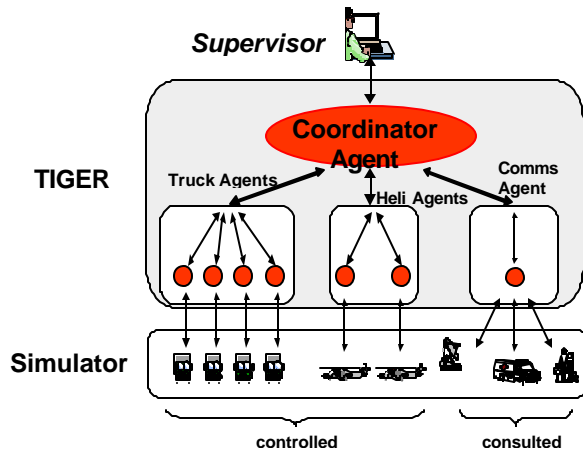


Figure 1. TIGER Architecture

Task management constitutes a major component of an execution agent's decision-making process. An execution agent must determine what to do if, while executing one task, the coordinator agent passes it a second task. It must also decide when to drop tasks that are not progressing well in favor of new tasks with higher potential for success. For simplicity, we limit each task execution agent to at most one active task at any point in time. Agents may also have pending tasks (which they intend to undertake) and preempted tasks (which were begun but put aside for higher-priority tasks). Tasks are assigned to individual agents and do not require coordination with other agents for their completion.

Unexpected events (e.g., a medical emergency) may require immediate response. Events are characterized by the properties *location*, *time*, *severity* (an integer 0 to 10), *number* of people affected, and *type* (e.g., evacuation, medical). The coordinator agent selects an appropriate task execution agent to deal directly with each such event, thus bypassing the task pool.

These characteristics of tasking simplify the decision process for what an execution agent should do when it receives a task request. The agent can choose from among several combinations of actions, including *ignore* the event, *adopt* a new task to respond to the event, *abandon* the current active task, *transfer* the task to another agent, or *postpone* the current task until the new task is completed. The agent's plan library includes options for each of these choices.

5. REPRESENTATION OF AGENT GUIDANCE

Our language for representing agent guidance builds on three components: the underlying *agent domain theory*, a *domain metatheory*, and the connectives of first-order logic. Using these elements, we develop the main concepts for our model of agent guidance. These consist of an *activity specification* for describing abstract classes of action, a *desire specification* for describing abstract classes of goals, and an *agent context* for describing situations in which guidance applies.

5.1 Domain Metatheory

A standard domain theory for an agent consists of four basic types of element: *individuals* corresponding to real or abstract objects in the domain, *relations* that describe characteristics of the world, *goals* that an agent may adopt, and *plans* that describe available means for achieving goals.

The domain metatheory defines semantic properties for domain theory objects. These properties can be used to express preferences among otherwise equivalent options; they also enable description of activity at a level that abstracts from the details of an agent's internal representations. As discussed in (Myers 2000), a metatheory can provide a powerful basis for supporting user communication. The main concepts within our metatheory for agent guidance are *features* and *roles* defined for agent plans and goals (similar to those of (Myers 1996)).

Consider first plans. A *plan feature* designates an intrinsic characteristic of a plan that distinguishes it from other plans that could be applied to the same task. For example, among plans for route determination, there may be one that is `OPTIMAL` but `SLOW` with a second that is `HEURISTIC` but `FAST`; each of these attributes could be modeled as a feature. Although the two plans are operationally equivalent (i.e., same cue and preconditions), their intrinsic characteristics differ significantly. Features provide the means to distinguish among such operationally equivalent alternatives.

A *plan role* describes a capacity in which a domain object is used within a plan; it maps to an individual variable within a plan. For instance, a route determination plan may contain variables `location.1` and `location.2`, with the former corresponding to the `START` and the latter the `DESTINATION`. Roles provide a semantic basis for describing the use of individuals within plans that abstracts from the details of specific variable names.

In analogous fashion, we can define features and roles for goals. For example, a goal of informing another party of task progress may have a `COMMUNICATION` feature and a `RECIPIENT` role. These metatheoretic elements could be used, for instance, to specify the class of goals that involve communicating with the commander (i.e., those goals with the feature `COMMUNICATION`, and role `RECIPIENT` instantiated to `Commander`).

The value of the domain metatheory lies with its provision of a semantically motivated abstraction of the underlying planning domain. This abstraction builds on semantic linkage among domain elements, specifically the sharing of roles and features among plans and goals. A domain metatheory would be developed in conjunction with the definition of the underlying domain theory for an agent. As discussed in (Myers 2000), a domain metatheory should be a natural by-product of a principled approach to domain modeling.

We use the symbols f and r to denote features and roles. The symbols F^P and R^P denote the set of plan features and roles defined for a given agent; similarly, F^G and R^G denote the set of defined goal features and roles.

5.2 Activity and Desire Specifications

An *activity specification* provides an abstract characterization of a class of plan instances. An activity specification is defined in terms of a set of required and prohibited plan features, a set of plan roles, and constraints on how plan roles can be filled.

Definition 1 (Activity Specification) An *activity specification* $A = \langle F^+, F^-, R, \mathbf{f} \rangle$

consists of

- a set of *required features* $F^+ \subseteq F^P$
- a set of *prohibited features* $F^- \subseteq F^P$
- a set of *roles* $R = \{r_1, \dots, r_k\}$ where $R \subseteq R^P$
- a *role-constraint formula* $f[r_1, \dots, r_k]$

The role constraint formula may contain variables, which we represent using the notation `<var-class>.<integer>` (for example, `location.1`). Such variables are interpreted as having existential scope; thus, belief of a role-constraint formula containing variables requires the existence of instantiations for the variables for which the formula is believed.

Example 1 (Activity Specification) The following activity specification describes the class of plan instances with the feature `SURVEY` but not `AIR-BASED`, where the variable that fills the role `DESTINATION` is instantiated to a location in the same sector as the agent's current position.

```
Features+: SURVEY
Features-: AIR-BASED
Roles: DESTINATION
Constraint: (AND (CURRENT-POSITION position.1)
               (= (SECTOR position.1) (SECTOR DESTINATION)))
```

A *desire specification* constitutes the goal-oriented analogue of an activity specification, consisting of a collection of required and prohibited features from F^G , required roles from R^G , and a role-constraint formula. We use the symbol D to represent a generic desire specification.

5.3 Agent Context

Just as individual plans employ preconditions to restrict their applicability, guidance rules require a similar mechanism for delimiting scope. To this end, we introduce the notion of an *agent context*. While plan preconditions are generally limited to beliefs about the world state, our model of agent context focuses on the BDI executor state of an agent (see Section 2.2). In particular, an agent context is characterized in terms of the agent's beliefs, desires, and intentions, as well as the current goal to which it is responding within a given cycle of the executor loop. Beliefs are specified in terms of constraints on the current world state. Desires are specified as desire specifications that describe goals that the agent has adopted, including the goal currently being expanded. Intentions are specified as activity specifications that describe intended plans of the agent.

As discussed in Section 2, our model of agency assumes a hierarchical collection of plans and goals; furthermore, agents are capable of multitasking (i.e., executing multiple intentions in parallel). Within a given phase of the BDI execution cycle, goals for an agent of this type can be scoped in three ways:

- *Current goal*: the goal for which the BDI executor is selecting a plan
- *Local goals*: the current goal, or any of its ancestors
- *Global goals*: any goal of the agent

Distinguishing these different scopes for goals enables guidance to be localized to highly specialized situations. Intended plans can be scoped similarly.

Definition 2 (Agent Context) An agent context is defined by a tuple $C = \langle \Phi, D, A \rangle$, where

- Φ is a set of well-formed formulae (i.e., *beliefs*)
- $D = D^C \cup D^L \cup D^G$ is a set of current, local, and global desire specifications
- $A = A^L \cup A^G$ is a set of local and global activity specifications²

5.4 Permission Requirements

Permission requirements are defined in terms of an *agent context* and a *permission-constrained activity specification*. The agent context defines conditions on the operating state of the agent that restrict the scope of the permission requirement. The permission-constrained activity specification designates a class of plan instances for which permission must be obtained.

Definition 3 (Permission Requirement) A *permission requirement* $\langle C, A \rangle$ consists of an agent context C and an activity specification A .

The interpretation of a permission requirement is that, when an agent's BDI state matches the specified agent context, permission must be obtained from the supervisor in order to execute a plan instance that matches the permission-constrained activity.

Example 2 (Permission Requirement) The statement “Seek permission to abandon survey tasks with priority > 5” could be translated into a permission requirement of the form

```
Agent Context:
  Local Activity Spec:
    Features+: SURVEY
Permission-Constrained Activity Spec:
  Features+: ABANDON
  Roles: CURRENT-TASK
  Constraint: (> (TASK-PRIORITY CURRENT-TASK) 5)
```

5.5 Consultation Requirements

We define two types of consultation requirement: *role-fill* and *plan*. A role-fill consultation requirement consists of an *agent context* and a *consultation role*. The interpretation of a role-fill consultation requirement is that when an agent's BDI executor state matches the agent context, any instantiation decision for a variable corresponding to the consultation role should be passed to the human supervisor. A plan consultation requirement consists of an *agent context* and a *desire specification*; it indicates that when an agent's BDI executor state matches the agent context, the human supervisor should be asked to select a plan to apply for any goal that matches the desire specification.

Definition 4 (Consultation Requirements) A *role-fill consultation requirement* $\langle C,r \rangle$ consists of an agent context C and a role r . A *plan consultation requirement* $\langle C,D \rangle$ consists of an agent context C and a desire specification D .

Example 3 (Consultation Requirements) The guidance “When responding to medical emergencies, consult on the choice of a medical evacuation site” could be translated into a role-fill consultation requirement of the form

```
Agent Context:
  Local Activity Spec:
    Features+: RESPONSE, MEDICAL-EMERGENCY
Consultation Role: MEDEVAC-SITE
```

The guidance “Consult when choosing a response to a failed task if there are other tasks pending” could be translated into a plan consultation requirement of the form

```
Agent Context:
  Belief: (AND (PENDING-TASKS tasks.1) (NOT (NULL tasks.1)))
  Local Activity Spec:
    Features+: SURVEY, FAILURE
Desire Spec:
  Features+: RESPONSE
```

5.6 Strategy Preference

Strategy preference guidance consists of two components: an *agent context* and a *response activity specification*. The activity specification designates a class of recommended plan instances (i.e., choice of plan and variable instantiations for designated roles) for use when an agent's state matches the designated agent context.

Definition 5 (Strategy Preference) A *strategy preference rule* is defined by a pair $\langle CA \rangle$ where C is an agent context and A is an activity specification.

Example 4 (Strategy Preference) The statement “Don't adopt medical emergency tasks involving fewer than 5 people when the priority of the current task exceeds 8” could be represented by the following strategy preference rule:

```
Agent Context:
  Global Activity Spec:
    Features+: EXECUTE
    Roles: CURRENT-TASK
    Constraint: (> (TASK-PRIORITY CURRENT-TASK) 8)
  Current Desire Spec:
    Features+: RESPONSE
    Roles: EVENT
    Constraint: (AND (= (EVENT-TYPE EVENT) MEDICAL-EMERGENCY)
      (< (EVENT-NUMBER-AFFECTED EVENT) 5))
Response Activity Spec:
  Features-: ADOPT
```

This guidance would be relevant in an executor state where the current goal has the feature `RESPONSE`, as well as the role `EVENT` instantiated to a value with type `MEDICAL-EMERGENCY` and fewer than 5 affected people; in addition, there must be a `CURRENT-TASK` being executed with priority greater than 8. The response activity specification indicates not to adopt responsibility for the emergency in such cases.

6. GUIDANCE SEMANTICS

Semantically, guidance acts as a filter on the plan instances that an agent can execute. When a standard BDI agent attempts to find an instance of a plan from its library to apply to a goal, it determines a set of applicable plan instances based on the plan cues and preconditions. The guidance limits this set further in accord with the following conventions.

A guidance rule is deemed *relevant* iff its agent context matches the current BDI executor state of the agent. Each relevant strategy preference rule filters out plan instances that do not match the response activity specification. Each relevant permission requirement rule filters out plan instances that match the permission-constrained activity specification but are refused permission by the supervisor. Each relevant role-fill consultation rule filters out plan instances that have the consultation role but do not bind the corresponding role variable to a value desired by the supervisor. Each relevant plan consultation rule filters out all plan instances other than that selected by the supervisor.

The remainder of this section defines the semantics of guidance more formally. Section 6.1 defines matching for an activity specifications, desire specification, and agent context; these concepts are used in Section 6.2 to define *guidance satisfaction*.

6.1 Matching

Let a be either a plan or a goal. The function $Features(a)$ designates the features defined for a , while $Roles(a)$ designates the roles. The function $RoleVal(a,r)$ designates the term that instantiates the role r in a (if one exists).³ We use the notation $\mathbf{f}[x_1:v_1, \dots, x_n:v_n]$ to represent a well-formed formula \mathbf{f} in which each occurrence of the variable x_i is replaced by the value v_i .

Definition 6 (Activity Specification Match) A plan instance p matches an activity specification $A = \langle F^+, F^-, \{r_1, \dots, r_k\}, \mathbf{f} \rangle$ in BDI executor state $S = \langle Bel, Des, Int, g^{cur} \rangle$ iff:

- $Features(p) \subseteq F^+$
- $F^- \cap Features(p) = \{ \}$
- $\{r_1, \dots, r_k\} \subseteq Roles(p)$
- $Bel \ ? \ \mathbf{f}[r_1:RoleVal(p,r_1), \dots, r_k:RoleVal(p,r_k)]$

Desire specification matches are defined similarly. We use the notation $ActivityMatch(p,A,S)$ and $DesireMatch(g,D,S)$ to denote activity specification and desire specification matches, respectively.

Let $Intention(g)$ be the intention that contains goal g , let $AllPlans(i)$ be the plan instances chosen for execution within intention i , and let $AllGoals(i)$ be the current goals of those plan instances.

Definition 7 (Agent Context Match) A BDI executor state $S = \langle Bel, Des, Int, g^{cur} \rangle$ matches an agent context $C = \langle B, D, A \rangle$ iff there exists some binding of variables in C such that

- For all $f \in B, Bel? f$
- For all $D \in D^C, DesireMatch(g^{cur}, D, S)$
- For all $D \in D^L, \exists g \in AllGoals(Intention(g^{cur})). DesireMatch(g, D, S)$
- For all $D \in D^G, \exists i \in Int, g \in AllGoals(i). DesireMatch(g, D, S)$
- For all $A \in A^L, \exists p \in AllPlans(Intention(g^{cur})). ActivityMatch(p, A, S)$
- For all $A \in A^G, \exists i \in Int, p \in AllPlans(i). ActivityMatch(p, A, S)$

We use the notation $ContextMatch(C, S)$ to denote that an agent context C matches a BDI executor state S .

6.2 Guidance Satisfaction

Building on the above definitions, we define the concept of *satisfaction* of agent guidance. In the following definitions, we distinguish *trivial* from *nontrivial* satisfaction. Trivial satisfaction occurs when the guidance rule does not impact the selection of plan instance for a given executor cycle. This can occur, for example, because the guidance is not relevant to the current executor state.

Definition 8 (Satisfaction: Strategy Selection Rule) A plan instance p *trivially satisfies* a strategy selection rule $R_S = \langle C, A \rangle$ for a BDI executor state S iff $ContextMatch(C, S)$ does not hold; p *nontrivially satisfies* R_S iff $ActivityMatch(p, A, S)$ holds.

The guidance rules for permission and consultation may require explicit feedback from the human supervisor. We model this interaction using the following *oracle fluents*:

- $Permission(p, S)$ specifies whether the human supervisor authorizes an agent with BDI executor state S to adopt plan p
- $ValueChoice(r, V, S)$ specifies the human supervisor's preferred instantiation for role r among values in V , when in BDI executor state S
- $PlanChoice(D, P, S)$ specifies the human supervisor's preferred plan in P for goals that match D , when in BDI executor state S

Definition 9 (Satisfaction: Permission Rule) A plan instance p *trivially satisfies* a permission requirement rule $R_P = \langle C, A \rangle$ for a BDI executor state S iff either of $ContextMatch(C, S)$ or $ActivityMatch(p, A, S)$ does not hold; p *nontrivially satisfies* R_P iff $Permission(p, S) = True$ holds.

Consultation requirements apply when there are choices among role values or applicable plan instances. Thus, the decision as to whether consultation is necessary depends on what other plan instances are applicable. These notions are captured in the following definitions for satisfaction of role-fill and consultation rules.

Definition 10 (Satisfaction: Role-fill Consultation Rule) Let $R_{RC} = \langle C, r \rangle$ be a role-fill consultation rule and $S = \langle Bel, Des, Int, g^{cur} \rangle$ be a BDI executor state. Let P be the set of

applicable plans for g^{cur} . Let $V(r,P)$ be the set of instances to which role r is instantiated in P , that is $V(r,P) = \{RoleVal(p_i,r) \mid p_i \in P \wedge r \in Roles(p_i)\}$. A plan $p \in P$ *trivially satisfies* R_{RC} for S iff either $ContextMatch(C,S)$ does not hold, $r \notin Roles(p)$, or V contains one or fewer elements; a plan p *nontrivially satisfies* R_{RC} for S iff $RoleVal(p,r) = ValueChoice(r,V,S)$.

Definition 11 (Satisfaction: Plan Consultation Rule) Let $R_{PC} = \langle C,D \rangle$ be a plan consultation rule and $S = \langle Bel, Des, Int, g^{cur} \rangle$ a BDI executor state. Let P be the set of applicable plans for g^{cur} . A plan $p \in P$ *trivially satisfies* R_{PC} for S iff either $ContextMatch(C,S)$ does not hold, $DesireMatch(g,D,S)$ does not hold, or P contains one or fewer elements; a plan p *nontrivially satisfies* R_{PC} for S iff $p = PlanChoice(D,P,S)$.

We say that an agent *violates* a piece of guidance during a given executor loop cycle iff the agent selects a plan instance for the current goal that does not satisfy the guidance. In the ideal, an agent executor would never violate user guidance. However, factors beyond the executor's control will generally make it impossible to avoid all violations. In particular, users may provide *conflicting guidance* to an agent that recommends incompatible choices.

Conflicts can arise in different forms. Here, we distinguish between *direct* and *indirect* conflicts. *Direct conflicts* arise when strategy preference guidance yields inconsistent recommendations within a given BDI executor cycle. Such conflicts can be at the level of plan instances (e.g., *Execute P* and *Don't execute P*) or the level of variable bindings (e.g., *Instantiate role R to A* and *Instantiate role R to B*). *Indirect conflicts* arise when guidance recommends multiple plan instances for execution such that, while their execution can be initiated, it is impossible for all of them to complete successfully. For example, the simultaneous execution of two plan instances could lead to deadlock or livelock situations, or downstream resource contention. Such harmful interactions can arise within any multithreaded system, not just systems in which guidance is used to select activities. Because general mechanisms for detecting these interactions do not yet exist, we consider only direct conflicts in this paper.

Given the potential for conflicting guidance, the best that we can expect from an executor is that it satisfy as much guidance as possible with each plan selection decision. The following definition of *maximal guidance compliance* captures this requirement.

Definition 12 (Maximal Guidance Compliance) An executor is called *maximally guidance compliant* iff for a given set of guidance G the executor selects only plan instances that satisfy a maximal subset of G for a given BDI executor state. That is, if the executor selects a plan instance p such that p satisfies G^+ and violates G^- where $G = G^+ \cup G^-$, then there is no applicable plan instance p' that satisfies $G^+ \cup \{R\}$ for any $R \in G^-$.

7. ENFORCEMENT OF GUIDANCE

In this section, we describe a simple extension to the BDI executor from Section 2 that ensures maximal guidance compliance for a set of strategy selection, permission requirement, and consultation requirement guidance rules. This set can vary over time but is assumed to be fixed for a given iteration of the executor loop.

Enforcement of guidance is attained through a simple modification to the executor loop at the point where a plan instance is selected for intending in response to a posted

goal or fact. First, the current BDI executor state for an agent is matched to the agent context of all current guidance to determine the relevant guidance for the current execution cycle. Next, each plan instance is tagged with any relevant guidance that it violates. A strategy preference rule will be violated by plan instances that do not match its response activity specification. For a permission rule, the human supervisor must be queried to determine whether the plan instance is allowed. For a consultation rule, the user must be queried for any plan or role instantiation choices.

If there are plan instances with no guidance violations, then any of them can be selected for application. If every plan instance has at least one violation, then the executor selects a piece of guidance and discards the violations associated with it. Any plan instance without violations would then be considered for application. This process repeats until at least one such plan instance results. The agent then applies the selected plan instance to the current goal.

Different selection strategies can be adopted for deciding the order in which to drop guidance rules. One compelling strategy would prefer permission and consultation requirements over strategy selection rules, since the former incorporate situation-specific information regarding user preferences. In addition, one could define *weights* that reflect relative strength of preference for guidance rules. A policy for combining and comparing the weights associated with the guidance rules that made the conflicting recommendations can then be used to select guidance to ignore, as a way of eliminating the conflict. TIGER incorporates weights in this manner to deal with direct conflicts.

We refer to the above BDI executor algorithm as the *guidance filtering* executor. It is straightforward to establish the following proposition.

Proposition 1 The guidance filtering executor is maximally guidance compliant.

8. GUIDANCE INTERFACE TOOLS

The motivation for our work on agent directability is to enable users to direct and manage agents in dynamic, unpredictable environments. The language presented in Section 5 provides a highly expressive formalism in which to define agent guidance; however, its complexity could overwhelm a typical user. For this reason, we have developed two interactive tools to help users define and manipulate agent guidance within the TIGER system. Figure 2 presents the interfaces for those tools.

The first tool is a *guidance authoring interface* that walks the user through the process of constructing a complex piece of guidance. To enable a simple specification process, the tool does not support the full expressivity of the formal guidance language; however, it supports a broad range of expressions, including the examples described in this paper. An accompanying *guidance library* can be used to store authored guidance. Users can select guidance from the library, as appropriate, for a particular situation.

The second tool is a *permissions specifier* that enables users to activate and deactivate permission requirements interactively for certain classes of action performed on certain types of task. Selections made through this interface are compiled into corresponding permission requirement structures. While this interface limits the scope of permission requirements that can be expressed, it provides a simple, accessible specification mechanism.

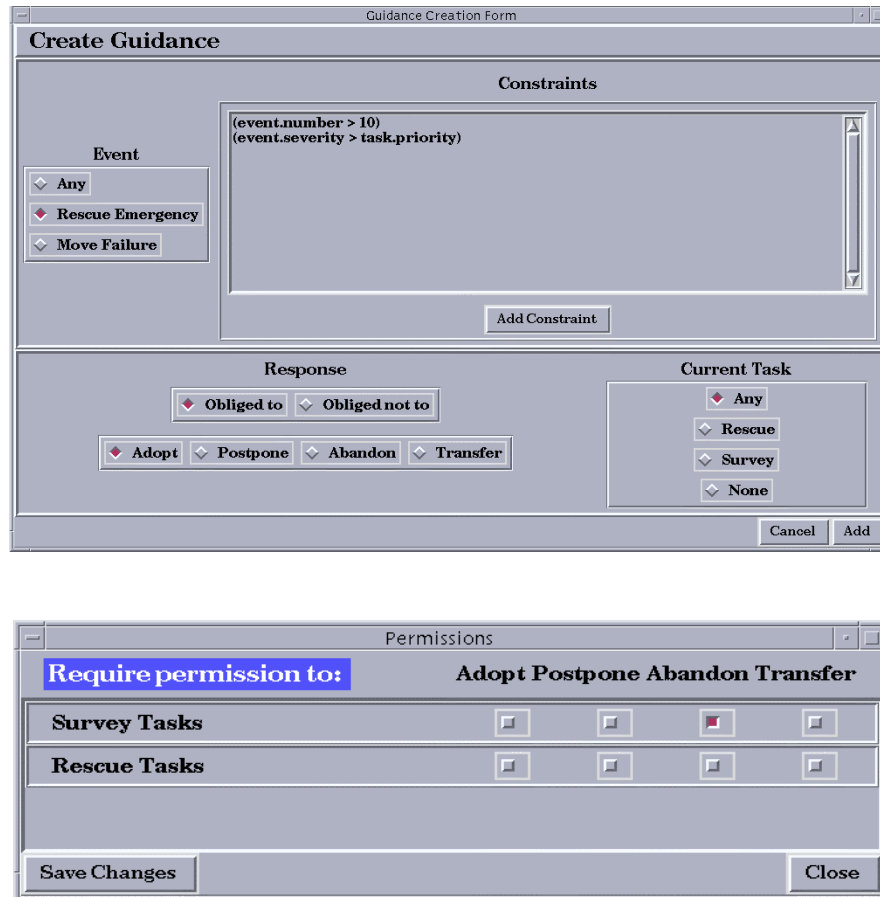


Figure 2. Guidance Authoring Tool (top) and Permissions Specifier (bottom)

9. RELATED WORK

Recognition of the need for technologies to support human-agent interactions has grown substantially in the past few years. However, few concrete technical approaches for enabling agent directability have been proposed.

Scerri et al. (Scerri, Pynadath and Tambe 2001) apply Markov decision processes (MDPs) to provide a form of adjustable agent autonomy. Their approach involves defining an MDP that describes all possible courses of action. The agent uses expected utility estimates from this model to determine when to consult the supervisor, and adjusts the model parameters based on experience. To avoid learning inappropriate behavior, users can impose constraints on what can be learned. In contrast to our approach of having a human explicitly define a policy for autonomy, an agent within this framework determines an appropriate level on its own.

Schreckenghost et al. (Schreckenghost et al. 2001) apply the concept of adjustable autonomy to the management of space-based life support systems. In their framework, a

human can take over both the selection of tasks to perform and the execution of those tasks. In contrast to our use of explicit policies, the level of autonomy is specified by directly altering a “level of autonomy” setting (*manual* vs. *autonomous*); this setting can be contextualized to an individual task, a subsystem, or all tasks.

Our strategy preference guidance selects among previously defined alternative plans; it does not expand the behavioral capabilities of the agent. In contrast, the work on policy-based control for distributed systems management supports runtime definition of new behaviors (e.g., (Moffett and Sloman 1993)). Policy languages in this area focus on the concepts of *authority* and *obligation* to perform actions.

10. CONCLUSION

Our framework for human directability of agents enables a user to define policies for adjustable agent autonomy and strategy preference. Through these mechanisms, a human supervisor can customize the operation of agents to suit his individual preferences and the dynamics of complex execution environments. In this way, system reliability and user confidence can be increased substantially over fully autonomous agent systems. The power of these ideas has been demonstrated within the TIGER system, which supports a human intelligence officer in managing a community of agents engaged in tasks for information gathering and emergency response.

Many outstanding issues in this area remain to be addressed; we briefly describe three topics for future work.

Detecting and Resolving Guidance Conflicts As discussed above, TIGER recognizes only a limited class of guidance-related conflicts (namely, direct conflicts among guidance). Indirect conflicts among guidance, and conflicts between guidance and ongoing activities require more powerful detection methods that reason about the downstream effects and requirements of plans. Furthermore, our prioritization scheme for resolving direct conflicts presents a simple approach to conflicting guidance; it would be interesting to incorporate more advanced conflict resolution policies, such as those of (Dignum et al., 2000) and (Lupu and Sloman, 1999).

Community Guidance The forms of agent directability described in this paper focus on influencing the behavior of an individual agent. Human supervisors will also want to express control at the *community* level, to encourage or discourage various forms of collective behavior. The guidance “Keep 2 trucks within 15 miles of headquarters” provides an example. Enforcement of this type of guidance will require mechanisms that support information exchange and coordinated action selection among groups of agents.

Collaborative Control Our model of agent directability provides a form of *supervised autonomy* (Barber and Martin, 1999) in which control over autonomy rests solely with the human supervisor. Some situations may benefit from a more collaborative approach, where both sides share control over initiative. For example, an agent may choose to initiate a dialogue with the human in situations where adherence to guidance would interfere with the pursuit of current goals, rather than blindly following the user’s recommendations.

NOTES

1. The system operates within a testbed that simulates a major hurricane in Central America; the testbed is built on the MAPLE system (<http://www.cs.cmu.edu/~maple/>).
2. Because the motivation for guidance is to influence the choice of plan instance for the current goal, the agent context excludes an activity specification for the current plan.
3. For a plan p , $RoleVal(p,r)$ could be a variable, nonground term, or ground term, depending on the level of instantiation within p . For simplicity, we restrict the structure of plans so that roles are bound to ground terms as part of the testing of applicability of a plan, through unification with the cue and preconditions.

ACKNOWLEDGMENTS

The authors thank Eric Hsu for his contributions in developing the TIGER interface, and Sebastian Thrun and his group at CMU for providing the MAPLE simulator. This work was supported by DARPA under Air Force Research Laboratory Contract F30602-98-C-0160.

REFERENCES

- K.S. Barber, C. E. Martin. Agent autonomy: Specification, measurement, and dynamic adjustment. In *Proceedings of the Autonomy Control Software Workshop, Agents '99*, pp. 8-15, Seattle, WA, 1999.
- P. Bonasso. Issues in providing adjustable autonomy in the 3T architecture. In *Proceedings of the AAAI Spring Symposium on Agents with Adjustable Autonomy*, 1999.
- H. Chalupsky, Y. Gil, C. A. Knoblock, K. Lerman, J. Oh, D. Pynadath, T. A. Russ, M. Tambe. Electric Elves: Applying agent technology to support human organizations. In *Proceedings of the Thirteenth Conference on Innovative Applications of Artificial Intelligence*, 2001.
- F. Dignum, D. Morley, E. A. Sonenberg, L. Cavedon. Towards socially sophisticated BDI agents. In *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS'2000)*, 2000.
- G. Ferguson, J. Allen. TRIPS: Towards a mixed-initiative planning assistant. In *Proceedings of the AIPS Workshop on Interactive and Collaborative Planning*, 1998.
- M. P. Georgeff, F. F. Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.
- E. Lupu, M. Sloman. Conflicts in policy-based distributed systems. *IEEE Transactions on Software Engineering, Special Issue on Inconsistency Management*, 25(6), 1999.
- J. D. Moffett, M. S. Sloman. Policy hierarchies for distributed systems management. *IEEE Journal on Selected Areas in Communications*, 11(9), 1993.
- K. L. Myers. Strategic advice for hierarchical planners. In L. C. Aiello, J. Doyle, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR '96)*. Morgan Kaufmann Publishers, 1996.
- K. L. Myers. Domain metatheories: Enabling user-centric planning. In *Proceedings of the AAAI-2000 Workshop on Representational Issues for Real-World Planning Systems*, 2000.
- K. L. Myers, D. N. Morley. Directing agent communities: An initial framework. In *Proceedings of the IJCAI Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents*, 2001.
- A. S. Rao, M. P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the International Conference on Multi-Agent Systems*, San Francisco, 1995.
- P. Scerri, D. Pynadath, M. Tambe. Adjustable autonomy in real-world multi-agent environments. In *Proceedings of the International Conference on Autonomous Agents*, 2001.
- D. Schreckenghost, J. Malin, C. Thronesbery, G. Watts, L. Fleming. Adjustable control autonomy for anomaly response in space-based life support systems. In *Proceedings of the IJCAI Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents*, 2001.
- D. E. Wilkins, K. L. Myers. A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation*, 5(6), 1995.