Towards the implementation of multiagent planning dialogues

Yuqing Tang
PhD Program in Computer Science
Graduate Center,
City University of New York
365 Fifth Avenue
New York, NY 10016, USA
Email: ytang@gc.cuny.edu

Timothy J. Norman
Dept of Computing Science
The University of Aberdeen
Aberdeen, AB24 3UE, UK
Email: t.j.norman@abdn.ac.uk

Simon Parsons
Dept of Computer & Information Science
Brooklyn College,
City University of New York
2900 Bedford Avenue
Brooklyn, NY 11210 USA
Email: parsons@sci.brooklyn.cuny.edu

***Abstract*—To support coalition teams, we are investigating the use of software agents that can help in planning team activities, thus reducing the cognitive burden on team members. Since the timely delivery of information can be crucial to team performance, we are especially interested in creating plans than explicitly include communications between team members. In previous work we developed a formal model of planning and communication that can support the creation of such plans. Here we describe how to implement this formal model, in particular how to map our model into the language of binary decision diagrams (BDDs), a representation of states and actions for which there are efficient open-source planners.**

I. Introduction

This paper is concerned with managing collaboration in a team. In particular, we are interested in teams engaged in military missions, and teams in which members may come from different parts of an international coalition. In such situations, effective coordination can be problematic, with units unable to communicate easily, and handicapped by having been trained to operate under rather different doctrines. Software agents can support effective collaboration in teams, and can overcome some of the problems with coalition forces. In particular, agents can filter messages [13]; coordinate activities [4], ensure timely delivery of crucial data [15]; and enforce the correct protocol for team behavior, [8].

In our work, we are particularly interested in how software agents can help in the planning and execution of missions. Previously, we have described how to manage appropriate communication between agents during plan execution [20], and how to integrate this communication into team plans [21], developing a mechanism for simulanteously constructing plans for a team and the communication necessary to carry out the plan. Here we describe an implementation of this mechanism.

The mechanism centers around a state-space model adapted from non-deterministic planning [5]. *States* are objects that capture some aspect of a system, and *actions* are transitions between states. States and actions together define a *state-space*. When action effects are non-deterministic [5] then what one seeks for any state-space is a *policy*: i.e. a state-action table to specify which actions one should take in a given state. We define a non-deterministic domain to be a tuple $\mathcal{M} = \langle \mathcal{P}, \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$ where:

- $\mathcal{P} = \mathcal{P}_S \cup \mathcal{P}_A$ is a finite set of propositions;
- $\mathcal{S} \subseteq 2^{\mathcal{P}_S}$ is the set of all possible states;
- $\mathcal{A} \subseteq 2^{\mathcal{P}_A}$ is the finite set of actions; and
- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the state-transition relation.

A propositional language $\mathcal{L}$ with quantification extension can be defined by allowing standard connectives $\wedge, \vee, \rightarrow, \neg$ and quantifiers $\exists, \forall$ over the propositional variables. The resulting language is a logic of quantified boolean formulae (QBF) [2]. Now, QBF formulae can be represented using a data structure called a Binary Decision Diagram (BDD) [2], and the advanatge of this representation that is able to efficiently compute the operations that are necessary for planning. This is exactly way that the current state of the art planning systems are implemented [5] — they take QBF descriptions of states goals and actions, translate these into BDDs and compute policies from them.

The key insight in [21] is that since communication is another form of action, we can adapt the kinds of model described in [5] to not only come up with a policies that include physical actions, what we call *world policies*, but with policies for communication, which we call *dialogue policies*. Indeed, we can, as described in [21], come up with policies that merge communication and physical action.

Now, to provide an implementation that can create such policies automatically, we have to provide a means to specify descriptions of world states, or physical actions, or dialogue states and dialogue actions in QBFS that can then be translated into the BDDs that can be fed into planning algorithms. In fact, we go further. Rather than expecting states, goals and actions to be specified as QBFs, we want to provide a more human-readable language, an *ontology* for describing the kinds of team coordination scenario that we are interested in, and have our implementation automatically translate this into the BDD description. In addition to making the system more user-friendly, this additional level of abstraction will make it easier to experiment with different ontologies for describing scenarios, and simpler to connect this work with

the collaborative planning model CPM [14].

II. An ontology

The main contribution of this paper is to present an ontology that can be used to describe a specific domain and the mapping from this ontology into the BDD format that can be used as input to a system, such as [11] which can create plans. We aim to do this in a very general way, intending that as much of the process as possible be automated, and providing algorithms that can do this automation.

A. A Heirarchy of Attributes

We start with the logical attributes that capture properties of a domain. An attribute is formally defined as:

$$\begin{aligned}&ATTRIBUTE ::\equiv ATOMIC\_ATTRIBUTE\\&\quad |COMPOUND\_ATTRIBUTE\\&ATOMIC\_ATTRIBUTE ::\equiv\\&\quad \langle NAME\rangle\ \langle WIDTH\rangle\\&COMPOUND\_ATTRIBUTE ::\equiv\\&\quad \langle NAME\rangle\ ATTRIBUTE\_LIST\\&ATTRIBUTE\_LIST ::\equiv\\&\quad ATTRIBUTE\_LIST\ ATTRIBUTE|ATTRIBUTE\end{aligned}$$

where $\langle NAME\rangle$ is a string to name the attribute, and $\langle WIDTH\rangle$ is an integer to describe the number of information bits are needed to represent the attribute[1].

The set of attributes that describe a domain stand in a compositional heirarchy. The attribute $JOINT\_STATE$, which appears in every specification capturing the state space of the whole team that is being modelled (and which is the input to the planner), will have N sub-attributes $STATE$ of individual team members (assuming we have N team members in the system). An individual's $STATE$ might be further decomposed into $SELF\_LOCATION$ (the agent's location), $ENEMY\_LOCATION$ (the enemy's location) and so on. However, the planning algorithm will only need to understand $JOINT\_STATE$ and will ignore the detailed structure of the $JOINT\_STATE$.

B. Mechanisms to Assign BDD Variables to Attributes

Attributes thus provide a general language for describing a domain. To plan, we need to convert this into a set of BDDs. A BDD is just a representation of all the possible models of a set of propositional variables, so establishing a set of BDDs from a set of attributes comes down to assigning a BDD variable to each variable in the set of attributes.

The basis of any process to do this assignament is what call the BDD variable locator, $ATTR\_LOCATOR$, which keeps track of which BDD variables have already been assigned to which attributes. This has the following function:

- $[name_1][name_2]\ldots[name_h]$: locate an attribute (either compound or atom) (and all its BDD variables)
- $[name_1][name_2]\ldots[name_h][ith]$: locate the ith bit of the attribute $[name_1][name_2]\ldots[name_h]$ (if this is not an atom attribute, the BDD variables will be numbered by pre-order traversing the sub-attributes rooted at $[name_1][name_2]\ldots[name_h]$)
- $[name_1][name_2]\ldots[name_h][ith, jth]$: locate the bits from the ith bit to the jth bit of the attribute $[name_1][name_2]\ldots[name_h]$

Given this we can implement a simple mechanism which assigns BDD variables from x_0 to x_K to an attribute $ATTR$ where K is the width of the $ATTR$. The assignment is done in preorder by traversing the hierarchy of attributes from the root to the bottom and from left to right. Since the size of the resulting BDD is highly related to the ordering in which the BDD variables are assigned to the attributes, we make use of a frequently used assignment mechanism. The mechanism will require additional inputs in the form of a set of pairs:

$$\begin{aligned}&INTERLEAVES =\\&\quad \{\langle ATTR\_LOCATOR,\quad ATTR\_LOCATOR'\rangle\}\end{aligned}$$

which denote attributes for which the BDD variables may be interleaved in the BDD. For example, if $INTERLEAVES$ contains $\langle [js][s_1], [js][s_2]\rangle$, then the BDD variables assigned to agents T_1 and T_2's states will be interleaved. It is also useful to be able to identify a list of attribute pairs or attribute bit pairs that should be assigned the same (sub-)vector of BDD variables, thus reducing the size of the BDD:

$$\begin{aligned}&UNIFIED =\\&\quad \{\langle ATTR\_LOCATOR, ATTR\_LOCATOR'\rangle\}\end{aligned}$$

This is another mechanism used in the planning literature to reduce the information needed by the planning algorithms [7].

Finally, we can also make use of additional information about the domain that allows us to assign the BDD variables in some preset order. We capture this information as:

$$\begin{aligned}&\langle ATTR\_LOCATOR_1,\\&\quad ATTR\_LOCATOR_2, \ldots ATTR\_LOCATOR_k\rangle\end{aligned}$$

These assignment mechanisms are sufficient for our current experiments, but there are others that can be used for more complex problems [10].

C. Operations on attributes

The system implements the following operations on attributes:

- $Clone$: $Clone(attr)$ will clone the system of and attribute and return an attribute with the same structure as $attr$ and temporarily with the same name as $attr$
- $+$: $attr(n) = attr_1(n_1) + attr2(n_2) + \ldots + attrN(n_N)$ means that a compound attribute name n is formed by having a list of attribute $attr_1(n_1), attr_2(n_2), \ldots, attrN(n_N)$.
- $Name$: $Name(attr, newName)$ will name the $attr$ with a new name $newName$. It is usually used in combination with $Clone$. $attr1(n') = Clone(attr2(n))$ is a

[1]The number of bits is necessary because of the subsequent translation into QBFs — each bit requires a propositional variable to encode it.

shortcut of two steps: $attr1 = Clone(attr2(n))$ and $Name(attr1, n')$.

As we will see, these operations are necessary to support the automatic creation of dialogue states and actions for a domain.

D. Relations on Attributes

For this paper, the equality relation $=$ is defined as

$$ATTR\_LOCATOR_1 = ATTR\_LOCATOR_2.$$

The equality relation contains all the equal pairs of values in attributes $ATTR\_LOCATOR_1$ and $ATTR\_LOCATOR_2$ (it is required that the two attributes are of the same width). Other relations have been implemented for BDD representations, but for now the equality relation constant is sufficient.

$$U(ATTR\_LOCATOR_1, ATTR\_LOCATOR_2, \{REL\_ATTR\_LOCATOR_i\}) ::\equiv ((ATTR\_LOCATOR_1 - \{REL\_ATTR\_LOCATOR_i\}) = (ATTR\_LOCATOR_2 - \{REL\_ATTR\_LOCATOR_i\})$$

where $ATTR\_LOCATOR - \{REL\_ATTR\_LOCATOR_i\}$ stands for the set of BDD variables located by $ATTR\_LOCATOR$ but excluding the variables located by $ATTR\_LOCATOR[REL\_ATTR\_LOCATOR_i]$, and "U" stands for unitary. When $\{REL\_ATTR\_LOCATOR_i\}$ is empty, the "U" becomes a synonym for $=$.

III. Constructing and executing policies

Given an ontology that is composed of a set of attributes as above, we can turn to the information that must be encoded in terms of these attributes if we are to create team plans.

A. Joint policy context

For now we consider that plans are created centrally (distributed planning is future work) with the planner operating on descriptions of the *joint state*, that is the state of all team members, and *joint action*, that is the combination of all possible actions by all team members. All the necesary information about the joint state is collected in the set of attributes that describe all the possible sets of joint state js, joint action ja, and next joint state js' (this is the typical way to describe state spaces in this kind of planning) – we call this the *joint policy context*. This can be composed out of the sets of attributes that describe individual agents' state and action information as shown in Procedure 1. The attributes for individual agents, which contains $STATE(s_i)$ and $ACTION(a_i)$ for every agent T_i, will be specific to a given domain, and will be the input to the system we describe.

B. Dialogue context

A key aspect of our work is the integration of communication — which we think of as being in the form of a *dialogue* between team members — into the plans we construct. These dialogues are included in the plan to reduce undertainty that team members face as a result of their incomplete knowledge about the states of other team members. Here we will outline a way to construct the set of attributes that describe a dialogue, what we call the *dialogue context*, automatically out of the joint policy context, and then describe not only how to create a policy for action and a policy for dialogue, but also how to go about executing these policies as well.

Procedure 1 Compose Joint Policy Context

1: **function** $ComposeJointAttributes(WS)$ {
(1) $WS = \{W_i\}$ is a set of individual's world context with $W_i = \{s_i, a_i\}\}$
2: $JSTATE(js) \leftarrow \Sigma_i s_i$
3: $JSTATE(ja) \leftarrow \Sigma_i a_i$
4: $JSTATE(js') \leftarrow Clone(js)$
5: $WorldContext \leftarrow \{js, ja, js'\}$
6: **return** $WorldContext$
7: **end function**

We start by considering how to construct the necessary dialogue system from the joint policy context. There are many ways to do this — what we present is just one of these ways (a fuller analysis of the different ways that we might do this is left for future work). We begin with the joint policy context:

$$W = \{js, ja, js'\}$$

where

- js is the set of joint state attributes, which includes descriptions of individual states s_i ($i = 1, \ldots, N$);
- ja is the set of joint action attributes, which includes descriptions of individual actions a_i ($i = 1, \ldots, N$); and
- js' is the set of next joint state attributes, which includes descriptions of indivdual next states s'_i ($i = 1, \ldots, N$)

In addition, we need to denote the information about one agent that is available to another (since communication takes place exactly when one agent needs information that it doesn't have access to), and what information has been exchanged. The set of attributes that describe the availability of information between agents can be constructed as:

$$AL\_BITS(al_i) = Clone(js) + Clone(ja)$$

This denotes the availability of the bits of the information about a joint state-action pair in a policy. Similarly, $AL\_BITS[al_i][js][s_j]$ is the availability of agent j's state information in the information that agent i's has, and $AL\_BITS[al_i][ja][a_i]$ is the availability of agent j's action information to agent i. The set of attributes that describe the information that is communicated between agents is then:

$$CM\_BITS(cm_{i,j}) = Clone(s_i) + Clone(a_i)$$

This denotes whether bits of information regarding agent i's state and action have been communicated to agent j. Now,

$$CM\_BITS(cm_i) = \Sigma_{j \neq i} cm_{i,j}$$

is the set of communication bits of agent i, and $CM\_BITS[cm_i][cm_{i,j}][s_i][k]$ denotes the communication bit

of the kth bit of agent T_i's state variables, which denotes whether or not this has been communicated to agent T_j.

From these components, we can construct a dialogue system. The description of the dialogue state of agent T_i in the joint state $\{js, ja, js'\}$ can be computed as:

$$DIAL\_STATE(ds_i) = js + ja + al_i + cm_i$$

And we can also compute the set of dialogue actions available to T_i. If agent T_i has the ID:

$$AGENT\_ID(ag\_id) ::\equiv name(ag\_id)\ width(\lceil logN \rceil),$$

the IDs of the state and action variables that describe the state and action of an agent are:

$$VAR\_ID(v\_id_i) ::\equiv \\ name(v\_id_i)\ width(\lceil log(max(width(s_i), width(a_i)))\rceil)$$

and the variable indicating the selection of whether to tell the action or the state:

$$INFO\_DIM(dim) ::\equiv name(dim)\ width(1),$$

(where $dim = 0$ means a state variable is conveyed, and $dim = 1$ means an action variable is conveyed) and the value of a variable is encoded as

$$VALUE(vv) ::\equiv name(vv)\ width(1),$$

then T_i can carry out dialogue actions:

$$DIAL\_ACTION(da_i) ::= \\ name(da_i)\ list(dim, v\_id_i, ag\_id, vv)$$

meaning that agent T_i tells the agent with ID encoded in ag_id the value encoded in vv, where this is the value of T_i's state or action variable v_id_i indicating by dim. For convenience we write:

$$da_i[sv\_id_i] = k \quad ::\equiv \quad (da_i[dim] = 0) \wedge (da_i[v\_id_i] = k)$$

$$da_i[av\_id_i] = k ::\equiv \\ (da_i[dim] = 1) \wedge (da_i[v\_id_i] = (k + width(s_i)).$$

A similar construction for every $\{js, ja, js'\}$ will give the full set of dialogue acts for T_i, meaning that T_i can communicate (or not) the value of all of the variables that encode its state, and all the actions that it can carry out, across all possible states that it might be in. If we do this unrolling of all the dialogue states and actions for every agent, we can establish the joint dialogue state as:

$$JOINT\_DIAL\_STATE(jds) = \Sigma_i ds_i$$

and the the joint dialogue actions as:

$$JOINT\_DIAL\_ACTION(jda) = \Sigma_i da_i$$

Procedure 2 formalises the procedure by which dialogue context can be built from the joint policy context.

Procedure 2 Compose Dialogue Attribute Context

1: **function** $ComposeDialContext(W)$ {
(1) $W \supseteq \{js, ja\}$ is the world attribute context}
2: Compose jds and jda out of js and ja (as defined in Section III-B)
3: $DialContext \leftarrow \{jds, jda\}$
4: **return** $DialContext$
5: **end function**

C. Dialogue State Transitions

Given the dialogue action da, we can specify a corresponding message that can be passed between agents:

$$[tell(i, j, s_i[k], v)] = \\ (da_i[sv\_id_i] = k) \wedge (da_i[ag\_id] = j) \wedge (da_i[vv] = v)$$
$$[tell(i, j, a_i[k], v)] = \\ (da_i[av\_id_i] = k) \wedge (da_i[ag\_id] = j) \wedge (da_i[vv] = v)$$

These can then be used to specify transitions between the dialogue states. Thus $tell(i, j, s_i[k], v)$ creates the transition:

$$R^i_{i,j,k|S,D} = \\ (ds_i[js][s_i][k] = v) \wedge (ds_i[al][js][s_i][k] = TRUE) \\ \wedge (ds_i[cm_i][cm_{i,j}][s_i][k] = FALSE) \\ \wedge [tell(i, j, s_i[k], v)] \\ \wedge (ds'_i[cm_j][s_i][k] = TRUE)] \\ \wedge (U(ds_i, ds'_i, \{[cm_j][s_i][k]\}))$$

This means that if the value of a bit that can diambiguate the global state is available, and hasn't been communicated to agent T_j, then this information should be sent out to T_j, and the communication should be recorded. There is a corresponding transition for agent T_j:

$$R^j_{i,j,k|S,D} = \\ (ds_i[al][s_i][k] = TRUE) \\ \wedge [tell(i, j, s_i[k], v)] \\ \wedge (ds'_j[al][s_i[k] = 1)] \wedge (ds'_j[js][k] = v) \\ \wedge (U(ds_j, ds'_j, \{[js][k]\}))$$

while for all other agents T_l with $l \neq j, i$,

$$R^l_{i,j,k|S,D} = TRUE$$

We also have update rules for when the state is known but it is still unclear which action T_j should take. For T_i this is:

$$R^i_{i,j,k|A,D} = \\ (ds_i[js][a_i[k]] = v) \wedge (ds_i[al][js][a_i][k] = TRUE) \\ \wedge (ds_i[cm_i][cm_{i,j}][a_i][k] = FALSE) \\ \wedge [tell(i, j, a_i[k], v)] \\ \wedge (ds'_i[cm_j][a_i][k] = TRUE)] \\ \wedge (U(ds_i, ds'_i, \{[cm_j][k]\}))$$

Procedure 3 Compose Dialogue State Transitions

1: **function** $ComputeDialTransitions(DialContext)$ {
(1) $DialContext = \{ds, da, ds'\}$ is the dialogue attribute context}
2: Compute $\mathcal{R}_{|D}$ as described in Section III-C
3: **return** $\mathcal{R}_{|D}$
4: **end function**

and for T_j

$$\begin{aligned} R^j_{i,j,k|A,D} = \\ & (ds_i[al][a_i][k] = TRUE) \\ & \wedge[tell(i,j,a_i[k],v)] \\ & \wedge(ds'_j[al][a_i[k] = TRUE)] \wedge (ds_j[ja][k] = v) \\ & \wedge(U(ds_j, ds'_j, \{ja[a_i][k]\})) \end{aligned}$$

while for all agents T_l with $l \neq j, i$

$$R^l_{i,j,k|A,D} = TRUE$$

In total[2], we have the following state transitions for $tell(i,j,a_i[k],v)$

$$R_{i,j,k|A,D} = R^i_{i,j,k|A,D} \wedge R^j_{i,j,k|A,D} \wedge \bigwedge_{l \neq j,i} R^l_{i,j,k|A,D}$$

, and for $tell(i,j,s_i[k],v)$

$$R_{i,j,k|S,D} = R^i_{i,j,k|S,D} \wedge R^j_{i,j,k|S,D} \wedge \bigwedge_{l \neq j,i} R^l_{i,j,k|S,D}$$

.

In total, we can have the following dialogue system:

$$\mathcal{R}_{|D} = \bigwedge_i \bigwedge_j \bigvee_k (R_{i,j,k|S,D} \vee R_{i,j,k|A,D})$$

and Procedure 3 will construct the set of all dialogue state transitions from the dilaogue attributes which themselves were built from the components joint policy context $\{js, ja, js'\}$.

D. Policy Execution

The state-space model described above allows us to describe the world in which an agent finds itself, and the actions it can undertake. We now turn to considering what the output of planning process will be. We call this output a *policy*, and we consider it to simply be a set of state-action pairs,

$$\pi = \{\langle s_i, a_i \rangle\}$$

where $s_i \in \mathcal{S}$ and $a_i \in \mathcal{A}(s)$ with

$$\mathcal{A}(s) = \{a | \exists \langle s, a, s' \rangle \in \mathcal{R}\}$$

that is the set of actions that are applicable in s. A policy π is a *deterministic policy*, if for a given state s, there is no more than one action is specified by π, otherwise it is a *non-deterministic policy*. What we are calling a policy is the state-action table of [5]. It is also related to what the literature on MDPs calls a policy [1], but we allow a policy to only specify actions for a subset of all possible states. The space of all policies is denoted by Π. The set of states in a policy π is $S_\pi = \{s | \langle s, a \rangle \in \pi\}$. Adapting from [5], we have the following definition:

Definition 1: An *execution structure* induced by the policy π from a set of initial states I is a directed graph $\Sigma_\pi(I) = (V_\pi, E_\pi)$ which can be recursively defined as

- if $s \in I$, then $s \in V_\pi$, and
- if $s \in V_\pi$ and there exists a state-action pair $\langle s, a \rangle \in \pi$ such that $\langle s, a, s' \rangle \in \mathcal{R}$, then $s' \in V_\pi$ and $a : \langle s, s' \rangle \in E_\pi$ where the action a is the label of the edge.

Procedure 4 gives a suitable procedure for executing policies. $SenseCurrentState$ will update the current world state of T_i with sensing. $ReceiveCommunication$ will update the dialogue state by the communication it received. $RefineDecision$ is a function to refine the external world action decision by utilizing the information in the dialogue state and the knowledge of the joint policy $\pi_{|W}$ (the reference to the joint policy can be relaxed however):

$$\begin{aligned} RefineDecision(s_{i|W}, a_{i|W}, s_{i|D}) = \\ & \exists_{\vec{x}_{-a_{i|W}}} s_{i|W} \wedge s_{i|D} \\ & \wedge U(a_i, js[a_i]) \wedge U(s_i, da_i[a_i]) \\ & \wedge \bigwedge_{j=1}^{N} (ds_i[al_i][a_j] \rightarrow U(ja[a_j], ds_i[a_j]) \\ & \wedge \bigwedge_{j=1}^{N} (ds_i[al_i][s_j] \rightarrow U(js[s_j], ds_i[s_j])) \wedge \pi_{|W} \end{aligned}$$

where $\vec{x}_{-a_{i|W}}$ is the vector of BDD variables in the formula other than those of $a_{i|W}$. A simple example is that the dialogue state contains full information of the other agents' state-action pairs, so that an individual agent can choose a unique local action to ensure that the necessary joint state transition is completed. $RefineDialState$ is to refine the dialogue state based on its local sensing and decision, and have the information ready for communicating to other agents:

$$\begin{aligned} RefineDialState(s_{i|W}, s_{i|D}) = \\ & \exists_{\vec{x}_{-a_{i|D}}} s_{i|W} \wedge s_{i|D} \\ & \wedge U(a_i, js[a_i]) \wedge U(s_i, da_i[a_i]) \\ & \wedge U(ja[a_i], ds_i[a_i]) \\ & \wedge U(js[s_i], ds_i[s_i])) \wedge \pi_{|W} \end{aligned}$$

where $\vec{x}_{-s_{i|W}}$ is the vector of BDD variables in the formula other than those of $s_{i|W}$.

E. Creating world and dialogue policies

A world policy and associated dialogue policy can be created with an instantiation of Procedure 5, an adaptation of standard non-deterministic planning techniques to include

[2] $R^i_{i,j,k|A,D}$ and $R^j_{i,j,k|A,D}$ are of a simplified version here. In a real implementation, the frame intertial description will be in a more complicated form than $U(ds_i, ds'_i, \ldots)$ because of the need to take into account the effect of simultaneous dialogue actions.

Procedure 4 Execution of a world policy of agent T_i

Input: $\pi_{i|W}$: A local policy for T_i
Input: $\pi_{i|D}$: A local dialogue policy for T_i

loop
  $s_{i|W} \leftarrow SenseCurrentState(s_{i|W})$
  $s_{i|D} \leftarrow ReceiveCommunication(s_{i|D})$
  $a_{i|W} \leftarrow \exists_{\vec{x}_{i|W}}(s_{i|W} \wedge \pi_{i|W})$
  $a_{i|W} \leftarrow RefineDecision(s_{i|W}, a_{i|W}, s_{i|D})$
  if $|a_{i|W}| = 1$ **then**
    Execute $a_{i|w}$
    $s_{i|D} \leftarrow FALSE$ {Reset the dialogue state for a new round of communication}
  else
    $s_{i|D} \leftarrow RefineDialState(s_{i|W}, s_{i|D})$
    $a_{i|D} \leftarrow \exists_{\vec{x}_{i|D}}(s_{i|D} \wedge \pi_{i|D})$
    if $|a_{i|D}| = 1$ **then**
      Execute $a_{i|D}$
    end if
  end if
end loop

Algorithm 5 World policy generation (high level description)

function $ComputeWorldPolicy(W, D, I, G, R)$ {
(1) $W = \{js, ja, js'\}$: The world policy context,
(2) $D = \{ds, da, ds'\}$: The dialogue context,
(3) I: Initial states,
(4) G: Goal states,
(5) R: Joint world state transition }
repeat
  Search the execution structure (forward, backward, or best-search) of R between I and G
  for all joint decision $\langle s_k, a_k \rangle$ searched **do**
    if $\langle s_k, a_k \rangle$ is not feasible **then**
      Continue
    end if
    if $\langle s_k, a_k \rangle$ can be executed by individual agents without coordination **then**
      Add $\langle s_k, a_k \rangle$ to the world policy
    else if $\langle s_k, a_k \rangle$ can be coordinated by a dialogue **then**
      Add $\langle s_k, a_k \rangle$ to the world policy
      Plan a dialogue for $\langle s_k, a_k \rangle$, and add it into the dialogue policy
    end if
  end for
until A successful joint policy and dialogue policy are reached or fully explored the execution structure
end function

dialogue planning. A sound and complete instantiation of this procedure can be found in [21].

With the attributes for world policy and dialogue policy, we will have a overall system showed in Procedure 6. This gives all the steps in the process, from the composition of individual states and actions into joint descriptions, the synthesis of the dialogue model, the composition of the overall policies, and the projection of those policies to the individual agents. In the

Procedure 6 Overall System

Input: $IndividualStateTransitionOntology : \{W_i\}$ with $W_i = \{State(s_i), State(a_i)\}$
Input: $\{I_i\}, \{G_i\}, \{R_i\}, \beta_I, \beta_G, \beta_R$

$W \leftarrow ComposeJointAttributes(\{W_i\})$
$D \leftarrow ComposeDialAttributes(W)$
$I \leftarrow \bigwedge_i I_i \wedge \beta_I$
$G \leftarrow \bigwedge_i G_i \wedge \beta_G$
$R \leftarrow \bigwedge_i R_i \wedge R_I$
$\mathcal{R}_{|D} \leftarrow ComputeDialTransition(D)$
$\langle SA_{|W}, SA_{|D} \rangle \leftarrow ComputeWorldPolicy(W, D, I, G, R)$
$\{\langle SA_{i|W}, SA_{i|D} \rangle\} \leftarrow ProjectToLocalPolicy(SA_{|W}, SA_{|D})$
return $W, D, \{SA_{i|W}\}, \{SA_{i|D}\}$

procedure,

$$ProjectToLocalPolicy(SA_{|W}) = \exists_{-SA_{i|W}} SA_{|W}$$
$$ProjectToLocalPolicy(SA_{|D}) = \exists_{-SA_{i|D}} SA_{|D}$$

where $-SA_{i|W}$ and $-SA_{i|D}$ means the BDD variables other than those of $SA_{i|W}$ and $SA_{i|D}$ respectively.

IV. An NGO Example

To demonstrate the function of the system, we will run the system through an NGO example that is based on the example in [3]. Two agents, one representing an NGO (N) and one representing a peace-keeping force (F), are working in a conflict zone. The agents (and the organizations they represent) work independently and have different agendas. N is based at A in Figure 1. F is based at H. N's goal is to reach D to help the villagers there. F's goal is keeping the peace in general in the area, but it also has to protect N while N is carrying out its work. At any time, with some probability, some disruption may flare up at W. If it happens, only F has the surveillence data to know this is happening, and F must go to W to suppress the disturbance. The routes between different points are shown as arcs in Figure 1. N cannot traverse the routes $(J, W), (W, C), (W, B)$, when there is a disturbance at W, and it is only able to traverse (C, D) and (B, D) without harm when it is accompanied by F. N can traverse the rest of the routes independently and F can traverse any route. The goal of the agents is to have N reach D and to have F put down the conflict in W if it happens.

We first need to specify are the attributes that define the ontology we need to encode information about the scenario.

A. Common attributes

We need to describe locations, the ability to move, and the fact that there are discrete routes between different locations. Thus we have:

$$\begin{aligned} location(location) &::\equiv name(loc)\ width(3) \\ move(move) &::\equiv name(move)\ list(loc, loc') \\ loc(loc) &::\equiv clone(location) \\ loc'(loc') &::\equiv clone(location) \\ Route(Route) &::\equiv name(Route)\ list(loc, loc') \end{aligned}$$

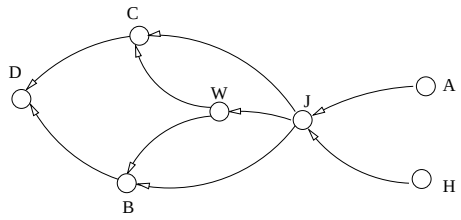

Fig. 1. An NGO team task

where possible location values are the encodings of A, B, C, D, H, J and W. The map can then captured as:

$$\begin{aligned} route &= \{(A,J),(H,J),(J,W),(J,C),\\ &\qquad (J,B),(W,C),(W,B),(C,D),(B,D)\}\\ \beta_{route} &= \bigvee_{(x,y)\in route} (route[loc]=x)\wedge(route[loc]=y) \end{aligned}$$

B. The NGO agent

Next we consider attributes that are specific to the NGO agent:

$$\begin{aligned} STATE(s_N) &::\equiv name(s_N)\ list(loc,health)\\ health &::\equiv name(health)\ width(1)\\ ACTION(a_N) &::\equiv name(a_N)\ list(stay,move)\\ stay &::\equiv name(stay)\ width(1) \end{aligned}$$

We use the proposition $health$ to capture the fact that the NGO will suffer it it tries to move in certain places without the presence of the peace-keeping force. Then, considering s_N, s'_N and a_N, we have:

$$\begin{aligned} R_{move_N} &= (s_N[loc]=a_N[move][loc])\\ &\quad\wedge(s'_N[loc]=a_N[move][loc'])\\ &\quad\wedge(a_N[stay]=FALSE)\\ R_{stay_N} &= (s_N[loc]=s'_N[loc])\wedge(a_N[stay]=TRUE) \end{aligned}$$

and with the unified list of attributes $\langle a_N[loc], route[loc]\rangle$, $\langle a_N[loc'], route[loc']\rangle$ we have:

$$\begin{aligned} &\beta_{route_N}=\beta_{route}\wedge\\ &\quad\neg(\bigvee_{\substack{(x,y)\in\{(J,W),\\(W,C),(W,B)\}}} route[loc]=x\wedge route[loc']=y) \end{aligned}$$

which gives the constraint on the NGO's movement, and

$$R_N = R_{stay_N}\vee(R_{move_N}\wedge\beta_{route_N})$$

which says that the NGO either stays in one place or moves along an allowed route.

C. The peace-keeping force

Nest we turn to the attributes specific to the peace-keeping force:

$$\begin{aligned} STATE(s_F) &::\equiv name(s_F)\ list(loc,conflict)\\ conflict &::\equiv name(conflict)\ width(1)\\ ACTION(a_N) &::\equiv name(a_N)\ list(move) \end{aligned}$$

Now, with the unified list: $\langle a_F[loc], route[loc]\rangle, \langle a_F[loc'], route[loc']\rangle$ we have: $R_F = R_{move_F}$

$$\begin{aligned} R_{move_F} &= (s_F[loc]=a_F[move][loc])\\ &\quad\wedge(s'_F[loc]=a_F[move][loc'])\wedge\beta_{route_F}\\ &\quad\wedge(a_F[move][loc']=W\rightarrow s_F[conflict]=FALSE) \end{aligned}$$

and, since the peace-keeping force can move along any route:

$$\beta_{route_F}=\beta_{route}$$

D. Joint state and actions

From the above, Procedure 1 will generate joint state and joint action descriptions. The joint state will be

$$JSTATE(js)=\Sigma_i s_i.$$

and since $s_i\in\{s_F,s_N\}$, this will become:

$$JSTATE(js)=s_F+s_N$$

Similarly, the joint action will be:

$$JSACTION(ja)=\Sigma_i a_i.$$

and since $a_i\in\{a_F,a_N\}$, this will become:

$$JSACTION(ja)=a_F+a_N$$

Given the joint state and joint actions, the constraint on the health of the NGO becomes:

$$\begin{aligned} \beta_{health} &= [[\neg(ja[a_F][loc]=B\wedge ja[a_F][loc']=D)\\ &\quad\wedge(ja[a_N][stay]=FALSE)\\ &\quad\wedge(ja[a_N][loc]=B\wedge ja[a_N][loc']=D)]\\ &\quad\vee[\neg(ja[a_F][loc]=C\wedge ja[a_F][loc']=D)\\ &\quad\wedge(ja[a_N][stay]=FALSE)\\ &\quad\wedge(ja[a_N][loc]=C\wedge ja[a_N][loc']=D)]]\\ &\quad\wedge js'[s_N][health]=FALSE \end{aligned}$$

In total:

$$R=R_F\wedge F_N\wedge\beta_{health}$$

E. Initial and terminal states

We have the following initial state:

$$\begin{aligned} I_1 &= (js[s_N][loc]=\text{“}A'')\wedge(js[s_N][health]=TRUE)\\ I_2 &= (js[s_F][loc]=\text{“}H'')\wedge((js[s_F][conflict]=TRUE)\\ &\quad\vee(js[s_F][conflict]=FALSE))\\ I &= I_1\wedge I_2 \end{aligned}$$

and the goal is:

$$\begin{aligned} G_1 &= (js[s_N][loc]=\text{“}D'')\wedge(js[s_N][health]=TRUE)\\ G_2 &= (js[s_F][conflict]=FALSE)\\ G &= G_1\wedge G_2 \end{aligned}$$

and this completes all the information we need to apply Procedure 6.

V. Conclusions

This paper takes an important step towards the implementation of the combined dialogue and planning system we proposed in [20], [21]. This system is built around a state-transition model which is specified using a set of attributes related to multiagent planning and dialogue. Different applications can specify their own set of atttributes, and the algorithms described here will produce joint plans and dialogues for those applications without detailed knowledge of the relevant states and actions.

In addition, the ontology we describe here can potentially serve as a bridge to a more comprehensive collaborative planning ontology, such as CPM, and the similarity between our state transition model and those used in decision-theoretic planning suggest that we can use MDP-based approaches to planning along with those described here.

As discussed by [18], teamwork requires requires the establishment of joint intentions and the determination of which goals to achieve, the creation of a plan, the sharing of knowledge about the environment in which the team is operating, and the ability to monitor plan execution. While we do not claim that what we have described in this paper is a comprehensive model of teamwork — it is much less powerful and comprehensive than Teamcore [19], Retsina [17] or GPGP/TÆMS [12], for example — it marks a useful step towards our overall goal of constructing a model of argumentation-based dialogue that can support many of the important aspects of teamwork. In particular, it deals with planning and the sharing of information.

One obvious area of future work is moving from a centralised planning process, which just hands every agent a policy that will help the team achieve its goals, to a decentralised process in which agents can engage in a discussion of the best plan. For that we plan to combine our prior work on argumentation-based planning [22], which assumes a simple, deteministic model of actions, with the work we have described here. Another area of future work, which addresses the main area in which our model falls short of a model of teamwork, is to consider the formation of joint intentions. Here there is a rich vein of work to draw on, for instance [6], [9], and we will seek to incorporate this into our model. Finally, we mean to explore the connections between the model we are using here and those for decision theoretic planning [1] and multiagent reinforcement learning [16].

Acknowledgments

This work was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[2] R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[3] C. Burnett, D. Masato, M. McCallum, T. J. Norman, J. Giampapa, M. J. Kollingbaum, and K. Sycara. Agent support for mission planning under policy constraints. In *Proceedings of the Second Annual Conference of the ITA*, Imperial College, London, 2008.

[4] H. Chalupsky, Y. Gil, C. A. Knoblock, K. Lerman, J. Oh, D. V. Pynadath, T. A. Russ, and M. Tambe. Electric elves: Agent technology for supporting human organizations. *Artificial Intelligence*, 23(2):11–24, 2002.

[5] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2):35–84, 2003.

[6] P. Cohen and H. Levesque. Intention is choice with commitment. *Artficial Intelligence*, 42:213–261, 1990.

[7] S. Edelkamp and M. Helmert. Exhibiting knowledge in planning problems to minimize state encoding length. In *Recent Advances in AI Planning. 5th European Conference on Planning (ECP'99)*, volume 1809 of *Lecture Notes in Artificial Intelligence*, pages 135–147, New York, 1999. Springer-Verlag.

[8] M. Esteva, J. A. Rodriguez, C. Sierra, P. Garcia, and J. L. Arcos. On the formal specification of electronic institutions. In *Agent-mediated Electronic Commerce*, number 1991 in Lecture Notes in Artificial Intelligence, pages 126–147. Springer Verlag, Berlin, Germany, 2001.

[9] B. Grosz and S. Kraus. The evolution of sharedplans. In A. Rao and M. Wooldridge, editors, *Foundations and Theories of Rational Agency*. Kluwer, 2003.

[10] O. Grumberg, S. Livne, and S. Markovitch. Learning to order BDD variables in verification. *Journal of Artificial Intelligence Research (JAIR)*, 18:83–116, 2003.

[11] U. Kuter, D. S. Nau, M. Pistore, and P. Traverso. A hierarchical task-network planner based on symbolic model checking. In S. Biundo, K. L. Myers, and K. Rajan, editors, *ICAPS*, pages 300–309. AAAI, 2005.

[12] V. R. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. E. Neiman, R. M. Podorozhny, M. V. N. Prasad, A. Raja, R. Vincent, P. Xuan, and X. Zhang. Evolution of the GPGP/TÆMS domain-independent coordination framework. *Autonomous Agents and Multi-Agent Systems*, 9(1-2):87–143, 2004.

[13] P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–40, 1994.

[14] D. Mott and J. Hendler. Progress on the collaborative planning model. In *Proceedings of the First Annual Conference of the ITA*, University of Maryland, 2007.

[15] S. D. Ramchurn, B. Deitch, M. K. Thompson, D. C. de Roure, N. R. Jennings, and M. Luck. Minimising intrusiveness in pervasive computing environments using multi-agent negotiation. In *Proceedings of the 1st International Conference on Mobile and Ubiquitous Systems*, Boston, MA, 2004.

[16] P. Stone and M. M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[17] K. Sycara, M. Paolucci, J. Giampapa, and M. van Velsen. The RETSINA multiagent infrastructure. *Journal of Autonomous Agents and Multiagent Systems*, 7(1), 2003.

[18] K. Sycara and G. Sukthankar. Literature review of teamwork. Technical Report CMU-RI-TR-06-50, Carnegie Mellon University, November 2006.

[19] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7, 1997.

[20] Y. Tang, T. Norman, and S. Parsons. Agent-based dialogues to support plan execution by human teams. In *Proceedings of the Second Annual Conference of the ITA*, Imperial College, London, 2008.

[21] Y. Tang, T. Norman, and S. Parsons. A model for integrating dialogue and the execution of joint plans. In *Proceedings of the Eigth International Joint Conference on Autonomous Agents and Multiagent Systems*, Budapest, Hungary, May 10-15 2009.

[22] Y. Tang and S. Parsons. Argumentation-based dialogues for deliberation. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 552–559, New York, NY, USA, 2005. ACM Press.