Guided Team Selection*

Gil Tidhar
Australian Artificial Intelligence Institute
171 LaTrobe Street
Melbourne, Australia
Phone: (+61)-3-96637922
Fax: (+61)-3-96637937
gil@aaii.oz.au

Anand S. Rao
Australian Artificial Intelligence Institute
171 LaTrobe Street
Melbourne, Australia
Phone: (+61)-3-96637922
Fax: (+61)-3-96637937
anand@aaii.oz.au

Elizabeth A. Sonenberg
Department of Computer Science
University of Melbourne
Melbourne, Australia
Phone: (+61)-3-92879163
eas@cs.mu.oz.au

Abstract

Team selection, the process of selecting a group of agents with complementary skills to achieve a goal, is an important collaborative task in multi-agent systems. Typically, team selection occurs at run-time using a first principles approach, for example after agents have exchanged relevant information about their abilities, loads, or other status. In time-critical domains such approaches may be impractical. Our work assumes that agents have limited resources and are embedded in a continuously changing world. We provide a mechanism whereby system developers can describe "recipes" for team selection in terms of the required abilities of the team, and appropriate run-time constraints. The paper provides definitions and algorithms, and includes comparisons with related work.

Introduction

The Artificial Intelligence and Multi-Agent Systems research communities have given much attention to the problems of classical planning for single and multi-agent systems. Given a particular initial state and a set of actions, classical planning involves generating a combination of such actions that, if executed, will attain a desired state. In the multi-agent setting, classical planning extends to the generation of a plan for multiple agents. Generating a multi-agent plan encompasses the combination of actions, the coordination required between them, and the selection of agents that will execute each of these actions.

In the past several years, the focus of research into single agent planning has shifted from classical planning and plan recognition to reactive or procedural planning and plan recognition (Georgeff & Lansky 1987; Rao 1994). A similar shift is occurring in the focus of research into multi-agent planning (Grosz & Kraus 1993; Kinny *et al.* 1994). Two main assumptions distinguish reactive planning from classical planning: (a) the environment in which the agents are situated is continuously changing; and (b) the agents have limited resources available to them for achieving goals. In the case of multi-agent planning there is an additional assumption: (c) the nature of the environment and the various agent's abilities are such that the agents are required to cooperate in order to achieve their goals. Such assumptions required the development of specialized techniques for guiding the agents in their individual and collaborative activities.

One technique involved a review of the notion of a plan. In one approach to reactive planning, a plan as a "recipe" for achieving particular desired states (Georgeff & Lansky 1987). Recipes are abstract combinations of actions and sub-goals and are provided to the agents in advance. They are used to guide and constrain the resource-bounded agents in their decision-making and coordination processes. Thus they reduce the time required for searching through a possible solution space and the communication required for performing cooperative activity.

Much of the attention of multi-agent reactive planning has been given to the problems of generating a combination of actions and coordination activity required for multi-agent behavior. The problem of selecting appropriate agents (from all the agents in the multi-agent system) to perform an activity has been mainly done using classical search techniques. Although some of these approaches attempt to reduce the complexity of the problem by either fixing the sets of groups that can be considered (Decker & Lesser 1992), providing some limitations on the possible agents that should be considered (Tidhar & Rosenschein 1992), or performing some of the selection at development-time (Tidhar *et al.* 1992)[1], none have attempted to use recipes to guide the selection process.

We refer to the use of recipes to guide and constrain the selection process as *Guided Team Selection* and to the recipes as *allocations*. Guided team selection is applicable to domains where there are multiple agents with limited resources that must cooperate to achieve their goals in an environment that is contin-

* This work was partially supported by the Cooperative Research Centre for Intelligent Decision Systems, Melbourne, Australia.

[1] We refer to the time during the construction of the multi-agent system as *development-time* and the time during execution as *run-time*.

uously changing. Guided team selection also makes a number of additional assumptions: (a) the construction of the multi-agent system allows for development-time processing; and (b) all agents and their abilities are known at development-time.

In this paper, we focus on the use of allocations to guide the team selection process. In particular we present development-time algorithms that use allocations to generate a set of teams of agents for each goal that should be considered at run-time. We also provide a description of how the selection process proceeds during run-time. The formulation and analysis of these algorithms and their comparison with other relevant work provide an insight into the benefits, limitations, and complexities of our approach.

The guided team selection approach allows the developer to determine the level of guidance. Such guidance can range from being fully guided, resulting in an approach similar to that taken by Decker and Lesser (Decker & Lesser 1992), to being fully unguided, resulting in an approach similar to that taken by Smith and Davis in the Contract-Net Protocol (Smith 1980).

Although, reactive planning initially attracted some criticism, it has been successfully applied in its single and multi-agent forms to a number of complex problem domains. These problem domains include modelling of air-combat pilots (Tidhar, Selvestrel, & Heinze 1995), air-traffic management (Ljungberg & Lucas 1992), and business process management. In such problem domains it was possible to describe the required behavior of agents as plans that were a direct formulation of the standard operating procedures as identified by domain experts. These plans were then used by the agents to react to the changing world and cooperatively achieve their individual and team goals within the boundaries of their limited resources.

It was identified that in such problem domains there is also a need for the agents to be able to form and dismantle teams in response to the changing environment and the introduction of new goals. Although such activity is done in a structured and well defined way, it has to be done under the constraints imposed by limited resources. These requirements have been the main motivation for the work described here. Furthermore, we believe that this work is applicable to similar classes of problems that have previously been addressed by reactive planning.

Roles and Allocations

We will be introducing agent and team roles as abstract specifications or "types" to enable the developer to provide useful information to limit the search space when the system is seeking actual agents/teams to fill various roles. Before defining roles and allocations, let us first define the notion of agents and teams adopted here. An *actual agent* is defined in terms of sets of actions, beliefs, goals, plans, and intentions (Rao & Georgeff 1991). The multi-agent system is a collection of actual agents. In this paper we assume that the set of actual agents in the multi-agent system is known at development-time. An *actual team* is defined to be an ordered set of agents or actual teams. We refer to the elements of an actual team as *actual sub-teams*. An agent is thus regarded as the trivial case of an actual team.

One can view the process followed by an actual agent when determining how to achieve a goal as a means-end analysis with two distinct steps: (a) selecting a group of actual agents or teams that will attempt to achieve the goal; and (b) selecting the combination of actions to be taken by these actual agents in order to achieve the goal. The combination of actions is typically referred to as a *plan*. In this paper the plans are multi-agent joint plans as described elsewhere (Kinny *et al.* 1994). Such plans include the knowledge required for the coordination of the group of actual agents when they attempt to achieve their joint goal.

Role Specification

A plan specifies the means by which a goal is achieved by an actual agent or team when certain preconditions are satisfied. Such means are specified as a labelled, directed, AND-OR graph. The labels specify: (a) a primitive action that is to be performed by an actual agent; or (b) a sub-goal to be achieved by an actual agent or team (Kinny *et al.* 1994). Each actual agent has a set of such plans referred to as the *plan library* of the agent.

An *agent role* is a specification of an abstract agent that possesses particular abilities or skills, i.e., goals and actions, that are characteristic of the defined role. For example, an electrician can be characterized by the ability to achieve specific goals, such as, restore power, disconnect power, etc. Hence the specification of an electrician can be viewed as the specification of an abstract agent. An actual agent in a multi-agent system can now be viewed as filling one or more agent roles.[2]

We define a *team role* to be an unordered set of agent roles and other team roles with specific goals characteristic of the defined role of the team. For example, a manufacturing organisation can be viewed as a team role, with the CEO as an agent role and the marketing, production, and finance departments as other team roles. These team roles can be further decomposed into smaller team roles until we have only agent roles. The manufacturing organisation as a whole will have certain beliefs about its products, goals about its target production, plans about how to meet those targets,

[2]One can also use normal set operations on agent roles to define other agent roles. Normal set operations are used by performing tuple-wise set operations. One can thus create a hierarchy of agent roles. At the top are roles that can achieve any goal and at the bottom specialized roles that can achieve a limited number of goals.

etc. We do not allow recursive definitions. A team can now be viewed as a specific instance of a *team role*.

A *pure abstract team role* includes a set of goals with a role variable instead of the whole set of sub-roles.

There are two main differences between an agent role and a team role: (a) team roles do not include actions, as we assume that all actions are taken at the individual agent level; and (b) team roles assign responsibilities to other sub-roles, thus introducing a notion of structure. Note that as with agent roles we can now define a hierarchy of team roles. If a group of actual agents in the multi-agent system, t, is of a team role type, ρ, then we say that t is an instance of ρ or that t can fill the role ρ.

The specification of an actual agent or team as being able to fill a role depends on the skills of the actual agent or team. These in turn depend on the plans available to them. As the number of possible actual teams that can be formed is an exponential function of the number of actual agents in the system, it potentially can be a very large number.

It is not assumed that the developer of the system specifies the role filling abilities of all possible actual agents and teams. We simply allow the developer to specify the particular set of actual agents or teams that are able to fill each role. This information is then used during the automatic selection of an actual agent or team for a given goal. Note that if the developer does not provide any additional information, the system will automatically compute the set of teams that can fill a role.

Allocation Specification

The specification of agent or team roles is static and can be reasoned about at development-time. Having this information is no guarantee of forming a team and executing the plans; there may be run-time constraints that prevent the formation of such a team. These run-time constraints are specified as part of the *allocation*. For each goal an allocation specifies, the form of agent or team roles required to achieve the goal and the beliefs concerning the state of the world or the team being considered that need to be determined at run-time. The developer can thus use allocations to specify run-time constraints on the state of the world and the state of the actual agents or teams that will be selected.

Given an allocation, we refer to the goal, the belief, and the role, as the *relevance*, *team context*, and *potential team* of the allocation respectively. Note that a particular goal may be the relevance of multiple allocations. Similarly, the same role may be the potential team of multiple allocations. Given a particular goal the developer can thus specify the roles that should be considered under different conditions.

Example 1

Assume that the developer has specified two roles, a Customer Service Representative (CSR) agent role and a Team Leader (TL) agent role (where TL and CSR are unique role constants). A Team Leader is also required to be a CSR, but with additional abilities, such as authorizing credit up to \$200, instead of \$50 permitted for CSR's. Let α be an actual agent of type CSR and one of its goals includes the goal of getting an authorization for crediting a customer's account with \$100. Since α is a CSR it can not achieve this goal and it will have to select an actual agent or team that is able to do so.

In the absence of any role specifications, all actual agents and teams in the system are potential participants in the collaborative activity. However, if there exists an allocation which specifies that the role required for authorizing this amount is a TL, then the number of possible alternatives is reduced to agents of type TL. The developer can now create a goal for credit authorization and define two allocations, both relevant to this goal but with different team contexts and potential teams: (a) a team context where the amount is up to \$50 and the potential team is a CSR; and (b) a team context where the amount is up to \$200 and the potential team is a TL.

We can define the CSR agent roles to be: CSR = < { assure-service, report-fault, credit-customer}, {start-computer} > and the TL agent role to be:
TL = < {annual-review,produce-work-plan},∅> ∪ CSR = < { annual-review, produce-work-plan, assure-service, report-fault,credit-customer }, { start-computer } >.

Note that the TL is also a CSR but has some additional goals that it can achieve. To operate the exchange during one eight-hour shift (i.e., achieve the goal operate-shift) one actual agent of type TL and four actual agents of type CSR are required. The developer can thus define an allocation, CSA, that has as its potential team a team role including these requirements. Here we define this role explicitly and name it Customer Service Shift (CSS). The CSS role is defined as:
CSS=< {operate-shift}, {TL}, {CSR}, {CSR}, {CSR}, {CSR} >

The CSA allocation will thus be defined as:
CSA= < {operate-shift},(not (holiday $today)),CSS> = <operate-shift, (not (holiday $today)), < { operate-shift}, {{TL}, {CSR}, {CSR}, {CSR}, { CSR}} >>.
Note that since a TL is also a CSR, then according to the definition of CSS, an actual agent that can fill the TL role can potentially fill all the roles in CSS. Such a situation can be prevented by adding relevant constraints to the team context. ■

Team Selection

Let us now describe how the specification of roles and allocations are used to guide the process of team selection. The selection process starts with a given goal to be achieved (see Figure 1, where highlighted arrows indicate guided team selection approach). For the given goal we identify the set of allocations that are relevant to it. Each such allocation describes a role and a set of run-time constraints on the selected team. Given a particular relevant allocation, we can identify the set of actual teams able to fill the specified role. Each such actual team is referred to as a *relevant team* for the given goal. Each relevant team that satisfies the run-time constraints of the relevant allocations is referred to as an *applicable team* for the given goal. If

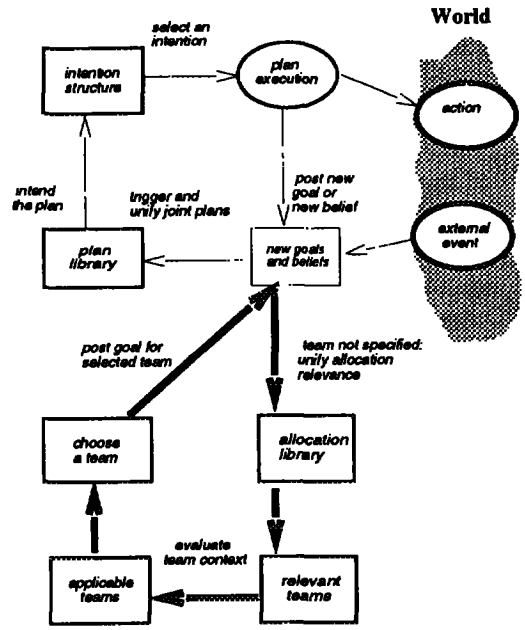

Figure 1: Operational semantics of reactive planning and guided team selection.

the set of applicable teams is empty then the attempt to achieve the goal fails; otherwise, one of the teams in the set is selected, formed, and is delegated the goal to be achieved.

The process of team formation has been addressed in previous work (Tidhar *et al.* 1992) and is beyond the scope of this paper. Note however that if the selected team can not be formed then another team from the set of applicable teams should be selected. If none of the teams in the set of applicable teams can be formed then the attempt to achieve the goal fails.

Relevant Teams

Given a particular goal g and a team t, t is relevant to the goal g if and only if there is an allocation α that is relevant for g and t is capable of filling the team role specified as the potential team of α.

If the developer does not specify the roles that the teams in the system can fill then the system will automatically compute the set of roles that each team can fill. Determining whether a team can fill a particular role will depend on the team's ability to achieve the goals specified in the role. This in turn depends on the availability of allocations relevant for that goal and the ability of the actual sub-teams to fill the roles as specified in the potential team of each allocation. Given a goal γ, the function $allocs(\gamma)$ returns the set of allocations with a relevance that unifies with γ. The set of teams that are relevant to an allocation α is computed using the algorithm **alloc-relevant-teams** and is denoted AR_α.

In the following algorithm, the set S_α is a cross-product of the sets of teams that can fill the roles specified in the potential team of the allocation (denoted as $potential(\alpha)$). It is computed using the algorithm **role-filling-teams**. Since the teams that are considered are required to achieve a goal we require that the team possesses at least one plan for achieving this goal. Hence, in the following algorithm the set AR_α is the subset of teams in S_α that also possess a plan for achieving the relevant goal of the allocation α (denoted as $relevance(\alpha)$). The function $plans(\tau)$ returns the set of plans that a team τ possesses and the function $purpose(p)$ returns the relevant goal of plan p.

alloc-relevant-teams(α)
S_α = **role-filling-team**($potential(\alpha)$)
$AR_\alpha = \{\tau \in S_\alpha | \exists p \in plans(\tau) :$
$\quad Unify(purpose(p), relevance(\alpha))\}$
return(AR_α)

Given a team role ρ, the algorithm **role-filling-teams** computes the set of teams that can fill the role ρ (denoted by RF_ρ). This algorithm is based on the set AR_α that is computed using the algorithm **alloc-relevant-teams**. The function $actperf(a)$ returns the set of actual agents (also defined as singleton teams) that can perform a; the function $expreq(\rho)$ returns the set of requirements for role ρ (i.e., the goals and/or actions); and the function $teams(\rho)$ returns the set of developer defined actual agents or teams that can fill this role.

role-filling-teams(ρ)
if ρ is an actual team **then**
  return(ρ)
if ρ is a pure abstract team role **then**
  $\sigma = first(expreq(\rho))$
  if σ is an action **then** $RF_\rho = actperf(\sigma)$
  if σ is a goal **then**
    $RF_\rho = \bigcup_{\alpha \in allocs(\sigma)}$ **alloc-relevant-teams**(α)
otherwise
  if $teams(\rho) \neq \emptyset$ **then**
    return($teams(\rho)$)
  $RF_\rho = \times_{v_i \in subteams(\rho)}$ **role-filling-teams**(v_i)
for $\sigma \in expreq(\rho)$ **do**
  if σ is an action **then**
    $RF_\rho = RF_\rho \cap actperf(\sigma)$
  if σ is a goal **then**
    $RF_\rho = RF_\rho \cap$
      $\bigcup_{\alpha \in allocs(\sigma)}$ **alloc-relevant-teams**(α)
return(RF_ρ)

Note our assumption that all the required knowledge, e.g., actual agents, available allocations, available plans, etc., is known at development-time. Also, it is assumed that all actual agents and all allocations are known to all other actual agents. This assumption can be relaxed by introducing the notions of an *allocation library* and a *known agents library* to each actual

agent. These libraries will hold the list of known allocations and known actual agents respectively. We can then modify the algorithms to consider only allocations and actual agents from each actual agent's libraries.[3]

The algorithms used for this computation are similar to the algorithms **goal-achieving-teams** and **plan-achieving-teams** used in previous work (Kinny *et al.* 1994).

Applicable Teams

The team context of an allocation specifies the state of the world or the state of the potential team that should hold for the team to be considered applicable for the goal. This allows the developer to guide the system in the dynamic aspects of the selection of teams and further restrict the number of teams considered. More formally, we say that a team τ is *applicable* to achieve a particular goal if and only if it is relevant with respect to an allocation α and the team context of α is true. The choice of the team will now be made from the set of applicable teams.

The availability of the information regarding the state of the team depends on the way this information is disseminated between the actual agents. The dissemination of the state of a team to other teams ensures that when an actual agent is required to reason about that state, it has the required information available. This information is made available to the actual agent through its beliefs about the world. The way each predicate is maintained and where the information is stored depends on the implementation of that predicate. One approach may be to retrieve the information from a local knowledge base that holds the model of the world. Another option is to query other actual agents' knowledge bases when the information is required. The method for retrieving the information should be defined for each component of the model of the world. A similar consideration should be given to changes in the state of the team, such as adopting a new goal. The developer should specify what the behavior of the system should be when the state changes. The behaviour will be determined by the state dissemination algorithm that is adopted.

The problem of information dissemination has been well studied by the distributed systems community (Alon, Barak, & Manber 1987; Ramamritham, Stankovich, & Zhao 1989). Distributed approaches include: information distributed on request; local information diffused between agents; and bidding techniques. Other approaches take a more "centralized" view, with the communication managed by a central process. Discussion of these issues is not pertinent to our focus here on team selection, beyond noting that performance will be affected by the actual choice.

[3]Such an approach will thus give a new meaning to the saying "It does not matter what you know but rather who you know".

Example 2

Consider the previous example where the developer imposes the restriction that in the allocation `CSA` the Team Leader cannot fill the role of a `CSR`. Now consider a particular multi-agent system that has been developed using these specifications. In this system there are two actual agents of type `TL`, `tl1` and `tl2`, six actual agents of type `CSR`, `csr1`,...,`csr6`, and no specific information on the teams that can fill the role `CSS`. Also, all actual agents have plans for operating the exchange.

Given the goal `operate-shift` the only relevant allocation is `CSA` and the relevant teams are a combination of all teams that are composed of one of the `TL`s and four `CSR`s. The set of relevant teams, computed by **role-filling-teams** is a cross-product of all the teams that can fill each of the roles in the potential team of `CSA`.

On the other hand, if the developer had specified that `CSS` can only be filled by the teams $teams(\texttt{CSS}) = \{tl1, csr1, csr2, csr3, csr4\}$, $\{tl1, csr1, csr2, csr3, csr5\}$, $\{tl2, csr1, csr2, csr3, csr5\}$, then there is would have been no need to compute this set and the team selection would have been trivial. ■

Analysis

Agent and team roles were introduced as "types" to enable the developer to reduce the part of the means-end tree that the system searches when seeking actual agents/teams. In this section we provide an analysis of the processes described above. Furthermore, we analyse the effects that the typing of actual agents and teams has on the breadth and depth of the means-end tree.

Formulation of Processes

For each goal, the number of relevant teams is a function of: the number of relevant allocations; the number of sub-roles in each allocation; and the number of teams that can fill each role required for each allocation. The teams that can fill each role are either provided by the developer or have to be computed using the **role-filling-teams** algorithm. This computation depends on the number of indirect recursive calls and the number of teams that can perform each action. The number of recursive calls depends on the information provided by the developer about the teams that can fill a particular role and the occurrence of actions in the definition of these roles. Whenever the developer provides more information, the average number of calls is reduced. When the developer provides all the information needed, there will be only one call to the algorithm.

We assume for simplicity that in each allocation each sub-role of the potential team is required to achieve only one goal. We then define the following domain-dependent parameters:

1. the average number of relevant allocations for each goal is l;
2. the average number of sub-teams in the potential team of an allocation (each required to fill a role) is

m;

3. the average number of calls to the **role-filling-teams** algorithm is f;
4. the average number of actual agents that can execute a particular action is n; and
5. the average proportion between: (a) the number of actual teams that are relevant for an allocation and also possess the plan in which this goal appears; and (b) the number of actual teams that are relevant for an allocation, is given by $0 \leq q \leq 1$.

Note that the parameter q is required because we assume that joint plans include knowledge about the coordination required for achieving a joint goal. Using such an approach reduces the communication required for coordination but it also constrains the teams that can achieve a goal to those that possess an appropriate joint plan. Using the **role-filling-teams** and **alloc-relevant-teams** algorithms the average number of relevant teams for a goal σ, denoted $\mathcal{R}_\sigma$, is:

$$average(\mathcal{R}_\sigma) = l^{(\sum_{i=0}^{f-1}(m)^i)} * q^{(\sum_{i=1}^{f}(m)^i)} * n^{(m^f)} \quad (1)$$

Note that this number can decrease if sub-roles are required to achieve more than one goal in an allocation. Since the set of relevant teams is created in a constructive way, we need only add the complexity of: (a) determining the set of relevant allocations for a given goal; and (b) determining for each allocation the roles each actual sub-team is required to fill. Both (a) and (b) can be done once. If there are p allocations defined and the parsing of each allocation is k_1, then the complexity of (b) is $k_1 * p$. If there are q goals referred to in the system and the complexity of the unification function $Unify$ is k_2, then the complexity of (a) is $k_2 * p * q$. If we also make equation (1) hold for the continuous case of f, we get that the average total complexity of calculating the relevant teams for a given goal is:

$$l^{(\int_0^{f-1}(m)^i\, di)} * q^{(\int_1^{f}(m)^i\, di)} * n^{(m^f)} + p*(k_1 + k_2 * q) \quad (2)$$

A detailed analysis of the algorithms described above can be found in a longer version of this paper (Tidhar 1995). The complexity of calculating the set of applicable teams depends on the complexity of the run-time constraints specified in the allocation and the complexity of the state dissemination algorithms. The analysis of these algorithms is beyond the scope of this paper.

Effects on Means-End Tree

Given a particular goal, the part of the means-end tree that is related to the process of team selection has at its leaves the set of applicable teams. The computation involved in exploring this tree depends on the typing of actual agents and teams done by the developer. This dependency is reflected in the number of recursive calls to the **role-filling-teams** algorithm (i.e., f). It is also dependent on the specialization of the actual agents and teams reflected in the number of teams that can execute a particular action (i.e., n).

Let us now describe a few examples that will demonstrate the effects of reducing the depth and breadth of the means-end tree on the number of relevant teams. If, for example, the roles are defined only in terms of actions, then the algorithm is called only once and the depth of the tree is 1, i.e., $f = 1$. On the other hand, if for each role the developer provides the set of teams that can fill this role, then again the algorithm is called only once. In this case, we get that the number of relevant teams is:

$$average(\mathcal{R}_\sigma) = l * (q)^m * (n)^m \quad (3)$$

One can now use the above formulation (Equation 1) for the continuous case and compare the effect of the various parameters on the number of relevant teams. It can also be used to provide estimates of the complexity of the process and the expected response time of the system. For example, let us compare two possible scenarios: (1) the number of recursive calls is fixed to 2 (i.e., $f = 2$), the number of teams that can execute a particular action is in the range [1–2.5] (i.e., $n \in$[1–2.5]) and all other parameters are fixed; and (2) the number of teams that can execute a particular action is fixed to 2 (i.e., $n = 2$), the number of recursive calls is in the range [1–2.5] (i.e., $f \in$[1–2.5]) and all other parameters are fixed with the same values as before.

When analyzing these two scenarios, we get that the number of recursive calls is dominant in the process and should be minimized as much as possible. That is, although difficult to do, the developer should focus efforts on attempting to provide as much information as possible on the actual agents and teams that can fill each role. The details of this comparison can be found elsewhere (Tidhar 1995).

Comparison with Relevant Work

Team selection can be done by actual agents exchanging, at run-time, full information about their skills, goals, plans, or beliefs. Probably the best known selection method in Multi-Agent Systems is the Contract-Net Protocol (Smith 1980) (CNP). Given a goal to be achieved, it is first decomposed into sub-goals and the protocol then uses a bidding-like mechanism to select the actual agents that will attempt to achieve the given sub-goals. Such a selection process either involves an exponential number of possible actual agent combinations, where all processing is done at run-time.

Note that in the CNP if a sub-goal has to be further decomposed, then the selection process will have to be repeated for that sub-goal. Therefore we can view the parameters from the formulation in the previous section, l, m, f, and n, as meaning: the number of decompositions of each goal; the number of sub-goals each goal is decomposed into; the number of times a

goal has to be decomposed before it is decomposed into primitive actions; and the average number of agents that can execute a particular action respectively.[4]

Thus, there are at least four major differences between the two approaches: (1) the actual values of the parameters; (2) the time in which the relevant teams are determined; (3) the additional overhead required; and (4) the flexibility of the approach.

With respect to the values of the parameters, we argue that in the Guided Team Selection (GTS) approach, such values would be less than the values used in CNP. The reason being that in GTS, the developer has to specify the relevant allocations for each goal. In CNP, each goal is decomposed into sub-goals. When specifying the allocations, the developer restricts the possible means that can be employed to achieve the goal to the most likely ones. It seems then that in general this number will be substantially less than the number of possible decompositions of a goal into sub-goals.

With respect to the time in which the relevant teams are determined, GTS does this at development-time while in CNP, this is done at run-time. This gives GTS an obvious advantage when considering time-critical domains. Note however that like in CNP, the GTS approach makes the final selection run-time. This allows the selecting agents to take into account dynamic consideration such as workload or the cost of achieving the goal.

With respect to the additional overhead required, CNP determines the set of relevant teams is done through the use of communication. The use of communication is typically very expensive, both in time and actual cost of usage. With respect to flexibility, CNP is more flexible in some respects than GTS. In CNP there is no substantial cost in adding or removing actual agents unexpectedly during run-time. This flexibility does not exist in GTS, since it is assumed that the actual agents are known at development-time.

Unlike CNP, in the approach adopted by Decker and Lesser (Decker & Lesser 1992) agents are homogeneous and teams are fixed in advance. Given a particular goal/task, team selection is reduced to the problem of load-balancing, i.e., dynamically assigning the goal/task to one of a small number of known teams. Obviously, this approach avoids the complexity of team selection. Unfortunately it also imposes a limitation on the ability of the actual agents to respond to the changing world by dynamically forming new teams or re-grouping.

Another variation of the CNP is the work of Shehory and Kraus (Shehory & Kraus 1993). Influenced by Game Theory, the solution they provide is a combination of the CNP and a Game Theoretic approach. In their work, they assume that: actual agents are capable of achieving the goals without assistance from other actual agents; cooperation emerges as a result of the actual agents attempting to maximize some individual utility function (i.e., it costs less to cooperate); and there is no central design process.

In previous work (Tidhar *et al.* 1992) we presented a solution to the team selection problem that is performed at development-time. Like with the CNP, it is fully unguided and resulted in an exponential number of teams. An analysis of these solutions can be found elsewhere (Tidhar 1995).

An unguided team selection results in either a blow-out in the number of interactions required to select the members of a team or a blow-out in the number of teams considered. Obviously this is unacceptable in time-critical domains where the actual agents have limited resources. As mentioned above, the GTS approach provides a generic way for the developer to provide additional knowledge to the system, so that this number can be substantially reduced. It also allows the developer to determine the level of guidance (or restriction) that is used in the team selection process.

Conclusions

The process of selecting the means for achieving a goal in a multi-agent system can be decomposed into two main steps: the first is the selection of a team that will attempt to achieve the goal; and the second is the selection of a (joint) plan of action. The choice of teams that can potentially achieve a goal depends on the skills of each team. In a totally unrestricted system, the number of such teams may be very large and hence such an approach is unsuitable for time-critical domains.

In this paper we have adopted an approach similar to that taken by reactive planning. We have defined a mechanism for team selection that allows the developer of the system to guide the process by specifying team selection "recipes". These recipes take the form of allocations. For a given goal, an allocation specifies the types of actual agents and teams that could be selected and appropriate run-time constraints.

These specifications are then used by algorithms that generate the appropriate sets of relevant teams at development-time. This computation reduces the run-time complexity of team selection and is therefore more suitable for time critical domains.

After selecting a team, one must form the team and delegate the goal to it. This raises issues such as the synchronization of the mind-set of team members (Tidhar *et al.* 1992). Similarly after selecting a plan, the team has to execute it. Executing joint plans is an area that has received much attention. Such work includes the work of Durfee and Lesser (Durfee & Lesser 1991), Grosz and Kraus (Grosz & Kraus 1993), Kinny et al. (Kinny *et al.* 1994) and many others.

[4] The parameter q is not relevant for CNP, since coordination is centralized (i.e., done by the *announcer*) and thus achievement of goals depends on communication not on common knowledge of a joint plan. We can thus say that $q = 1$.

The theoretical work described in this paper has been implemented as two separate demonstrator systems using different technologies (Gabric *et al.* 1996). The first demonstrator has used the dMARS^{TM} agent-oriented system (Aus 1996) as a basis for adding a team selection process. The second demonstrator has used the functional language, Gofer (Jones 1994), for implementing the agent architecture and the team selection process. These implementations have confirmed that the major benefit from the approach described here is obtained when the developer specifies the actual agents and teams that can fill some of the roles and when the actual agents are fairly specialized, i.e., there is not much overlap between their expertize.

Goals adopted by an actual agent or team would typically lead to the adoption of sub-goals. Further optimization of the selection process can be achieved by allowing the developer to specify, where possible, the types of teams that should be considered for achieving the sub-goals. This information, together with the allocations, can then be used in the selection process. We leave this as future work.

If we allow the developer to specify the types of teams that should be considered for sub-goals and if we enforce that sub-goals can only be achieved by sub-teams of the team achieving the goal, then we can provide some mechanisms for checking the plans that can be used by different teams. Again this is left as future work.

Acknowledgments

We would like to thank Tim Gabric and Simon Ch'ng for their valuable comments and contributions to the implementation of the ideas described in this paper.

References

Alon, N.; Barak, A.; and Manber, U. 1987. On disseminating information reliably without broadcasting. In *Proceedings of the Seventh International Conference on Distributed Computing*, 74–81.

Australian Artificial Intelligence Institute, Melbourne, Australia. 1996. *The dMARS V1.6.11 System Overview.*

Decker, K., and Lesser, V. 1992. Generalizing the partial global planning algorithm. *International Journal on Intelligent Cooperative Information Systems* 1(2):319–346.

Durfee, E. H., and Lesser, V. R. 1991. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics* 21(5):1167–1183.

Gabric, T.; Ch'ng, S.; Tidhar, G.; and Sonenberg, E. 1996. Implementation of the guided team selection approach. Technical Report 96/21, The University of Melbourne, Department of Computer Science.

Georgeff, M. P., and Lansky, A. L. 1987. Reactive Reasoning and Planning. In *Proceedings of AAAI-87*, volume 2, 677-682.

Grosz, B., and Kraus, S. 1993. Collaborative plans for group activities. In *Proceedings of IJCAI-93*, 367-373.

Jones, M. P. 1994. Gofer functional programming environment. Available via anonymous `ftp` from `ftp://ftp.cs.nott.ac.uk/pub/nott-fp/languages/gofer`.

Kinny, D.; Ljungberg, M.; Rao, A.; Sonenberg, E.; Tidhar, G.; and Werner, E. 1994. Planned team activity. In Castelfranchi, C., and Werner, E., eds., *Artificial Social Systems, LNCS Vol. 830.* Springer Verlag. 227–256.

Ljungberg, M., and Lucas, A. 1992. The oasis air-traffic management system. In *Proceedings of PRICAI-92.*

Ramamritham, K.; Stankovich, J. A.; and Zhao, W. 1989. Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Transactions on Computers* C-38(8):1110–1123.

Rao, A. S., and Georgeff, M. P. 1991. Modeling rational agents within a BDI-architecture. In *Proceedings of KR&R-91.*

Rao, A. S. 1994. Means-end plan recognition: Towards a theory of reactive recognition. In *Proceedings of KR&R-94.*

Shehory, O., and Kraus, S. 1993. Coalition formation among autonomous agents: Strategies and complexities. In *Proceedings of the Fifth Workshop on Modelling Autonomous Agents in a Multi-Agent World.*

Smith, R. G. 1980. The contract net protocol: High level communication and control in a distributed problem solver. *IEEE Transactions on Computers* C-29(12):1104–1113.

Tidhar, G., and Rosenschein, J. S. 1992. A contract net with consultants: An alternative architecture and experimental results. In *Proceedings of ECAI-92*, 219–223.

Tidhar, G.; Rao, A.; Ljungberg, M.; Kinny, D.; and Sonenberg, E. 1992. Skills and capabilities in real-time team formation. Technical Report 27, Australian Artificial Intelligence Institute, Melbourne, Australia.

Tidhar, G.; Selvestrel, M.; and Heinze, C. 1995. Modelling teams and team tactics in whole air mission modelling. In Forsyth, G. F., and Ali, M., eds., *Proceedings of the Eighth IEA/AIE Conference*, 373-381. Melbourne, Australia: Gordon & Breach Publishers.

Tidhar, G. 1995. Team oriented programming: Choosing teams. Technical Report 64, Australian Artificial Intelligence Institute, Melbourne, Australia.