A Flexible Framework for SharedPlans

Minh Hoai Nguyen and Wayne Wobcke

School of Computer Science and Engineering
University of New South Wales
Sydney NSW 2052, Australia
{minhn,wobcke}@cse.unsw.edu.au

Abstract. SharedPlans is an agent teamwork model that provides a formalization of the conditions under which a group of agents has a collaborative plan. This paper describes a general framework for implementing SharedPlans theory that addresses the computational issues of team formation, group plan elaboration and plan execution, involving coordination, communication and monitoring. The framework includes a team-oriented programming language for specifying recipes for SharedPlans, and an extension to a BDI architecture with several metaplans for interpreting the plan language. We indicate how the formal requirements for the establishment of SharedPlans are fulfilled within the framework.

1 Introduction

Agent teamwork models have been shown to be suitable in a range of applications, such as simulating air combat in a military training environment, Tidhar, Heinze and Selvestrel [19], and investigating Robot-Agent-Person teams in rescue domains, Scerri *et al.* [16]. Teamwork is characterized by a high degree of communication, collaboration and cooperation. A team of agents not only work together to achieve common goals, but maintain an ongoing commitment to the team, helping one another when necessary, keeping others informed of relevant information, etc. In addition, agents working in teams must maintain mutual beliefs about the world (beliefs not only about the world but about other agents' beliefs), joint goals (goals the team has collectively agreed to adopt), and joint intentions (commitments not only to the agent's own actions but to those of the team). Underpinning this team behaviour must be mechanisms to support team formation, communication between agents, synchronization and monitoring of the execution of joint plans, and the fulfilment of obligations to inform the team of relevant facts or notify the team when it is appropriate to abandon a team plan. Thus teamwork applications are complex, both theoretically and computationally.

There is a substantial "gap" between theory and practice for computational models of teamwork. For example, the theory of joint intentions developed by Cohen and Levesque [4] presents a formalization of joint persistent goals, goals such that all team members have that goal, and in addition, a commitment that on dropping the goal, to notify the other team members that the goal is no longer mutually held. This obligation to inform others on abandoning the team goal captures only one basic computational aspect of teamwork. The more complex theory of SharedPlans developed by Grosz and Kraus [7] shows how structured team plans can be modelled. SharedPlans theory aims

to formalize the conditions on the mental attitudes an agent must have to engage in collaborative activities, similar to the approach of Bratman [2] in which cooperative activity results from an interlocking set of intentions held by multiple agents. SharedPlans theory captures some complex constraints on the beliefs of agents participating in a team, but does not address computational issues surrounding the formation and execution of SharedPlans.

In contrast, existing computational approaches to teamwork, e.g. Tidhar [18], Kinny *et al.* [11], Tambe [17], Pynadath *et al.* [13] and the JACKTeams model [10], are focused on the need for efficient architectures, languages and platforms for developing applications, and are not well-motivated theoretically. This makes team-based applications built using those approaches difficult to understand. Hence we believe there is a need for a more theoretically sound implementation of a teamwork model that also addresses computational concerns.

In this paper, we present a general, flexible, approach to implementing the SharedPlans theory of Grosz and Kraus [7, 8] using as a basis a PRS-type architecture, Rao and Georgeff [14]. The approach, implemented using JACK Intelligent Agents[TM], allows programmers to specify team plans that are executed using the standard BDI interpreter. The approach addresses the computational concerns of team-oriented programming, such as how agents in a team agree on the form and structure of the team, how they synchronize their actions with one another, how and when they communicate with one another, how they monitor the execution of their joint plans and activities, and how they agree to abandon infeasible joint activities – in a manner consistent with SharedPlans theory. Our framework thus goes much further than the work of Grosz and Kraus [8], who describe an implementation of SharedPlans in a "Truckworld" environment, and that of Grosz and Hunsberger [5], who propose a SharedPlans extension (unimplemented) to the IRMA architecture of Bratman, Israel and Pollack [3], as both these approaches present general-purpose algorithms, but which are domain independent only for the formation of SharedPlans. The main contribution of our approach is a team-oriented programming language for specifying team plans, and domain-independent mechanisms, inherited from JACK, for SharedPlan execution and monitoring.

It should be noted that there are also several other implementations of SharedPlans. These include an electronic commerce system, Hadad and Kraus [9], a collaborative interface for distance learning (DIAL), Ortiz and Grosz [12], and a multi-agent system for collaboration of heterogeneous groups of people and computer systems (GigAgent), described in Grosz, Hunsberger and Kraus [6]. SharedPlans theory is also used as the basis of the Collagen dialogue system, Rich and Sidner [15]. However, all these systems are special-purpose implementations of the theory for specific problems and do not provide a general implementation of SharedPlans, whereas our framework provides an architecture for implementing SharedPlans independent of any specific application.

The remainder of this paper is organized as follows. Section 2 contains a summary of the main definitions of SharedPlans theory. In Section 3, we present MIST, our framework for implementing SharedPlans, describing the language for team-oriented programming and the internal architecture of MIST agents. In Section 4, we indicate how the basic definitions of SharedPlans theory are satisfied in MIST. Finally, we compare MIST to other general team-oriented programming platforms.

2 SharedPlans Theory

SharedPlans theory is a formalization of the mental attitudes of agents engaging in group activities. In SharedPlans theory, a group of agents have a collaborative plan when they each hold certain beliefs, desires and intentions. Thus the formalization attempts to define some complex concepts, such as full SharedPlans and partial SharedPlans, based on these basic mental attitudes. The formalization is given in first-order logic enhanced with several primitive predicates, modal operators, meta-predicates and action functions. Some axioms also govern the commitments and behaviour of agents. This section provides a brief summary of the main definitions of SharedPlans theory.

SharedPlans theory distinguishes between two kinds of intentions: intentions to perform an action (IntTo) and intentions that a proposition holds (IntThat). An agent intending to do an action must commit to doing that action, and must hold appropriate beliefs about its ability to perform the action, Grosz, Hunsberger and Kraus [6]. IntThat is used to represent an agent's expectation that some proposition will hold or some actions will be performed (possibly by other agents). An agent intending that a proposition holds must be committed to doing what it can to help make the proposition hold. However, unlike IntTo, with IntThat, it is not necessary for the agent to do or to be able to do anything.

A group of agents are said to have a SharedPlan for doing an action if they mutually believe that all members of the group are committed to having the action done. In addition, there exists a recipe such that the group mutually believe the need to perform all subactions in the recipe. Furthermore, the group must mutually believe that every subaction is catered for by a capable agent or subgroup of agents.

A more formal definition of a full SharedPlan, adapted from Grosz, Hunsberger and Kraus [6], is as follows. Let $FSP(Gr, \alpha, R_\alpha)$ denote that a group Gr has a full SharedPlan to do action α using recipe R_α. $FSP(Gr, \alpha, R_\alpha)$ holds if and only if the following conditions are satisfied:

1. Gr has a mutual belief that each member of Gr intend that Gr do α.
2. Gr has a mutual belief that Gr has a full recipe R_α for doing α.
3. For each subaction β in R_α, (i) there is an agent A_β in Gr having an individual plan or a subgroup Gr' of Gr having a full SharedPlan to do β, (ii) Gr has a mutual belief that A_β/Gr' has an individual plan/full SharedPlan to do β and is able to do β – note that agents who are not members of Gr' are not required to know the recipe involved, and (iii) Gr has a mutual belief that each agent in Gr intends that A_β/Gr' be able to do β.

Note that the theory is typically understood as providing conditions on the attribution of SharedPlans to a group of agents *at the time of plan formation*. It is unclear what SharedPlans a group of agents has during execution (e.g. whether they continue to have the whole SharedPlan or only the part remaining to be executed). This is because the notion of individual intention used in SharedPlans theory is not precisely defined (one might go further, in that if the theory of Bratman [1] is followed, an agent would no longer intend to do an action already completed; in such a case, the SharedPlans held by the agents are constantly changing as execution proceeds).

SharedPlans theory also provides a definition of a partial SharedPlan. Partial Shared-Plans are plans in which the recipes for the actions might be incomplete or in which some subactions have not been assigned to any agent or any subgroup of agents. In the case of a partial SharedPlan, the group must have a full plan for elaboration of the partial plan into a full plan.

A more formal definition of a partial SharedPlan (with the amendments to the full SharedPlan definition highlighted) is as follows. Let *PSP*(Gr, α, R_α) denote that a group Gr has a partial SharedPlan to do action α using recipe R_α. *PSP*(Gr, α, R_α) holds if and only if the following conditions are satisfied:

1. Gr has a mutual belief that each member of Gr intend that Gr do α.
2. Gr has a mutual belief that Gr has a full recipe R_α for doing α *or that Gr has a partial recipe R_α that may be extended into a full recipe and a full SharedPlan for selecting such an extended recipe*.
3. For each subaction β in R_α, *either* (i) there is an agent A_β in Gr having an individual plan or a subgroup Gr' of Gr having a partial SharedPlan to do β, (ii) Gr has a mutual belief that A_β/Gr' has an individual plan/partial SharedPlan to do β and is able to do β, and (iii) Gr has a mutual belief that each agent in Gr intends that A_β/Gr' be able to do β, *or (iv) Gr has a mutual belief that there is some agent in Gr or subgroup Gr'' of Gr that can do β and that there is a full SharedPlan to select such an agent/subgroup*.

The formalism of Grosz and Kraus [8] makes clearer some subtle points in the definition, e.g. with "Gr' is able to do β" each agent in Gr' must know a recipe that the group can use to do β (but the notion of group ability is not analysed any further), hence with the corresponding mutual belief of Gr, each agent of Gr must believe, for some candidate subgroup Gr', that that subgroup can do β, but Gr need not know (at the time of forming the partial SharedPlan) which subgroup Gr' will be selected. Now since SharedPlans theory is unclear about the SharedPlans held during plan execution, it is not clear whether, at the time that Gr' is selected, all agents in Gr are required to know the identity of Gr'. However, we believe that this condition would need to be satisfied in any practical implementation.

3   MIST: Minimal Infrastructure for SharedPlans Theory

This section describes MIST, our general implementation of SharedPlans theory using the JACK agents platform. First, MIST provides a specification language for recipes; recipes, once adopted by a group of agents that form the relevant beliefs and intentions then become SharedPlans held by the group. Second, MIST extends the JACK architecture with particular JACK plans (from now on called *processes* to avoid confusion with SharedPlans or domain plans) that embody the mechanisms for a group of agents to form teams, settle on a team plan, then execute a team plan (each agent synchronizing with and communicating relevant information to other agents in the team, while monitoring events in the environment that affect the success or failure of the team plan). MIST provides a generic platform for SharedPlans theory in that recipes are specified in a general "team-oriented programming" language independent of the JACK plans used

to form and execute SharedPlans (these JACK plans act more like meta-plans in taking SharedPlans as arguments). In MIST, the JACK platform is used for the implementation of the individual agents, for event processing and inter-agent communication, and to implement the MIST infrastructure processes for team plan formation and execution. MIST agents also use the JACK plan library for representing individual agent plans.

3.1 Team-Oriented Programming in MIST

Each agent system contains a set of agents and their capabilities (actions that can be executed by an agent). The main part of the agent is the recipe library. Each recipe is a hierarchical plan, containing a list of subactions and a list of actions it supports (can be used to fulfil). In addition, the recipe includes the interdependencies between subactions. As in the approach of Kinny *et al.* [11], each recipe also contains information about the roles in the recipe. Finally, each recipe contains a definition of its success and failure conditions. Each recipe is of the following form.

```
recipe => supports-action₁, ..., supports-actionₙ      (1)
SUBACTIONS : subaction₁, ..., subactionₘ               (2)
SUCCEEDS_WHEN : Dependency-Expression                  (3)
role :: subaction₁, ..., subactionₗ                    (4)
subaction := Dependency-Expression                     (5)
```

The first three lines are followed by any number of lines of the format (4) or (5). A line in format (4) gives a role name followed by the subactions carried out by agents in filling that role. This use of roles provides a convenient way to express constraints that some subactions must be performed by the same agent. A line in format (5) describes the start condition for a subaction. The formula means that the subaction on the left hand side should start when the conditions in the right hand side are met. The dependency expressions on the right hand side can be any boolean combinations of atomic conditions of the form `Condition+Time`, where an optional `+Time` is a numeric offset and `Condition` is of the form `Event` (meaning successful termination of an action or some condition in the world) or `Event@FAILURE` (meaning termination of an action execution with failure). MIST uses communication between agents, where possible, to synchronize execution using these conditions, so as to minimize the amount of monitoring required of the individual agents (this communication is also minimized).

As an example, consider a scenario involving a team of three scouting helicopters and an infantry platoon. The mission is to get the majority of the infantry platoon to the battlefield; it is considered successful even if some scouting helicopters or individual soldiers are shot down. The team can use a `ScoutMoveRec` recipe, defined as follows, with `Scouting2` as an alternative plan to be used when `Scouting` fails.

```
ScoutMoveRec => MoveToBattleField
SUBACTIONS : Scouting, Scouting2, Move, BuildBridge, PumpFuel
SUCCEEDS_WHEN : Move
MainTroopRole :: Move, BuildBridge
ScoutingRole :: PumpFuel, Scouting
Move := BuildBridge+5 AND (Scouting OR Scouting2)
Scouting := PumpFuel AND Sunrise
Scouting2 := Scouting@FAILURE
```

3.2 MIST Agent Architecture

The basis of MIST is an extension to JACK providing processes for team formation, group plan elaboration and SharedPlan execution. There are four types of processes: group-related elaborator processes (GREPs), group-related intention processes (GRIPs), single-agent processes (SAPs) and permanent monitor processes (PMPs). GREPs and GRIPs are responsible for coordinating group activities based on SharedPlans theory, and are similar to the processes used in Grosz and Kraus [8] (in turn similar to the algorithms of Kinny *et al.* [11]), while SAPs and PMPs are responsible for executing domain specific actions. Figure 1 depicts the internal architecture of a MIST agent in terms of the messages sent and received by the agent's processes.
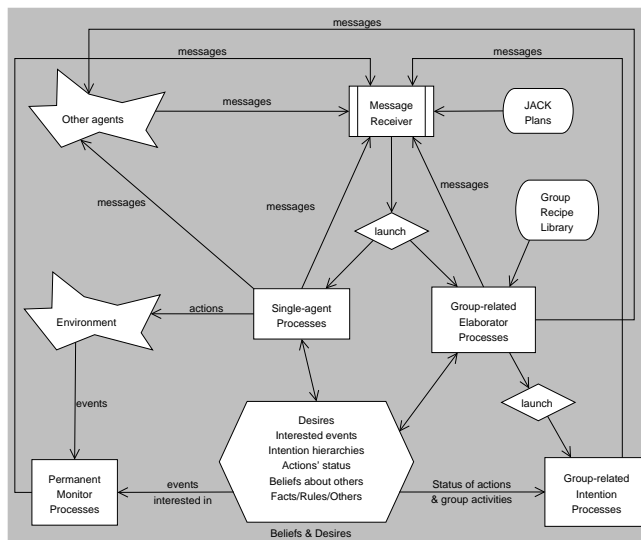

Fig. 1. MIST Internal Agent Architecture

Team Formation: When a message is sent to an agent requesting it to execute a group action, the agent invokes a GREP, which computes a list of all groups possibly able to perform the action (based only on a predefined list of agent capabilities). From this list, one group and a group leader are randomly chosen, and the GREP sends a message to every agent in the chosen group asking them to execute the group action. From this time onwards, the group leader is responsible for coordinating the group activities. If some agents refuse to participate or fail to respond before the timeout, the leader broadcasts termination messages. If the leader receives messages from every agent confirming their commitment to the action, it broadcasts this information to every member and the team is considered formed. Intuitively, the group now mutually believes that it is working towards an intention to do the group action, though it is yet to commit to that intention until a recipe for doing the group action is determined.

Group Plan Elaboration: GREPs are used for forming and elaborating group plans. After a team is formed to execute a group action, GREPs are used to identify a recipe for the action. The team leader sends messages to each team member requesting them to propose a recipe for the group action. The plan to execute the action fails if no replies are received, otherwise the leader selects one of the proposed recipes and requests each team member to commit to the chosen recipe. The recipe will only be adopted if all agents respond with such a commitment, and in this case, the team leader informs each team member of their commitment to the selected team plan.

The team leader now initiates a role assignment phase. The leader requests bids from each team member concerning the roles in the recipe they are prepared to fill, then computes a role assignment consistent with the bids. This does not guarantee the SharedPlan is executable, so next the team leader proposes the role assignment to each team member, requesting confirmation of the assigned role(s). The agent confirms only if it believes the whole role assignment is feasible. Again, the group action fails if there is no agreement on the role assignment. But if agreement is reached, intuitively the team now has a mutual belief that they have an intention to do the group action and a recipe for executing it (conditions (1) and (2) in the definition of a partial SharedPlan).

SharedPlan Execution: After a SharedPlan has been established, agents must fulfil their allocated roles by executing their part of the team plan. For this purpose, GRIPs are launched for each subaction in the plan. Moreover, as there might be interdependencies between these subactions, the group action and environmental conditions, agents need to keep track of these dependencies in order to execute their subactions at the appropriate time. The GRIP that corresponds to the intention to execute α first waits for the start condition of the super-action of α to be satisfied, then waits for the start condition of α to be satisfied, then executes α, and finally notifies all agents that need to know about the success or failure of α. Let us examine these steps in more detail. First, since each GRIP must wait for the start condition of its super-action, rather than having the GRIP itself monitor this condition, in MIST an agent relies on the team leader to notify it when this condition is satisfied. The team leader therefore needs to monitor this condition. Second, agents know the start conditions of their subactions by looking at the containing recipe. Third, actions are executed by launching either a SAP, in the case of an individual plan, or a GREP, in the case of a group action. Finally, the agents needing to know about the status of α can be determined from the recipe and the assignment of roles to agents. An agent only reports the execution result of an action to the responsible agents of dependent actions. The responsible agent of a group action is the team leader, and for an individual action is the agent executing the action. By using this mechanism, agents in the team are able to synchronize their execution of the team plan through communication, minimizing the individual monitoring done by each agent.

Permanent monitoring processes (PMPs) are domain-specific processes invoked only once when the agent is created. PMPs constantly monitor the environment for events the agent is currently interested in (determined from the agent's belief set). If a PMP detects such an event, it sends appropriate messages to the message receiver which is responsible for invoking the relevant processes to handle the event, for example, to execute an action. PMPs, therefore, can be used to initiate the whole process of team formation, group plan elaboration and SharedPlan execution.

4   Satisfying SharedPlans Theory

In this section, we discuss informally how MIST satisfies the requirements of Shared-Plans theory in attributing a partial SharedPlan to a group of MIST agents – focusing on the time at which a team plan is formed by the group as the standard case.

First note that though the basis of SharedPlans theory is mutual belief, the theory provides no indication of how mutual beliefs can be attained in practice. In MIST, agents have beliefs annotated by a set of agent names, so each agent can explicitly represent beliefs about other agents. The basic mechanism for a set of agents to reach mutual belief is communication. Here we make some standard assumptions about the truthfulness of agents and the reliability of the communication channel that guarantee that mutual belief can be attained. More precisely, it is assumed that an agent will only send a message that it believes α if it does believe α, and that any message sent will eventually be received (correctly) by its recipients within a known finite amount of time.

The beliefs required by SharedPlans theory are not explicitly represented in MIST agents, but instead are derived from their intention structures. In particular, a belief in the intention to do α is attributed to a MIST agent if it contains GRIPs to execute its part of a recipe R_α for doing α. The team formation and group elaboration procedures described above result in each agent in the team instantiating appropriate GRIPs, and moreover, each agent also knows that all agents in the team instantiate their appropriate GRIPs. So the mutual beliefs for conditions (1) and (2) of the partial SharedPlans definition obtain. For condition (2), it is also required that there be a mutual belief that the recipe R_α, if partial, can be extended into a full recipe for α, and that there is a full SharedPlan for selecting such an extended recipe. Here again, it is assumed that the GREP for group elaboration provides a mutually known mechanism for extending and selecting a full recipe for α extending R_α. However, note that MIST agents (like JACK agents) do no computation to determine whether R_α can in fact be extended to a full SharedPlan for α (doing so would require predicting the state of the world when R_α needs to be extended), but simply accept this, which is sufficient for condition (2), since SharedPlans theory requires only the *belief* that R_α can be extended to a full recipe.

Consider now condition (3) on partial SharedPlans. For individual actions, clauses (i)–(iii) are satisfied, whereas for group actions, clause (iv) is satisfied. First, for an individual action, the existence of a plan for the subaction follows from the role assignment phase of the group plan elaboration procedure. Each agent is explicitly required to check its assigned subactions against its capabilities, and only commit to the team plan if there is no conflict (however, MIST agents, again as in JACK, do not look for potential conflicts between different team plans). This procedure also establishes clause (ii), mutual belief in the relevant agent having an appropriate recipe to fulfil its role(s) in the team plan, and, in the case of group actions, clause (iv), however, only on the understanding that agents assume that for any group subaction β, a subsequently invoked GREP (the mutually known mechanism) will succeed in elaborating that action into a full SharedPlan for β. Clause (iii) involves IntThat. Here it is unclear what SharedPlans theory formally requires (see the discussion in Grosz and Kraus [8]), but we take it that any agent intends that the agent or group assigned to an action do that action. Appropriate communicative actions are included in the GRIPs for executing the SharedPlan to enable synchronization of action execution and abandonment of the SharedPlan.

Finally, note that although we have focused on the definitions of full and partial SharedPlans, SharedPlans theory also includes "rationality axioms" about the agents, such as that agents do not adopt conflicting intentions, Grosz and Kraus [7]. We have made no attempt to satisfy these axioms, as this would require highly complex reasoning by the agents that would at best undermine the efficiency of the computational model of teamwork, or at worst be impossible to compute, especially as agents are typically operating under a high degree of uncertainty in a dynamically changing environment.

5 Related Work

In this section, we discuss related general computational approaches to modelling teamwork and supporting team-based applications. First, Kinny *et al.* [11] presented a BDI-style approach to representing and executing team plans, which introduced the use of roles needed to be filled by the agents in a team. Synchronization amongst team members occurs through rewriting the team plan to include communicative actions between agents informing them when a synchronization point has been reached.

In the STEAM architecture, Tambe [17], each agent has a copy of a plan in which certain steps are designated as team plans (requiring coordinated execution). For each team plan, the team leader sends a synchronizing message to establish a joint persistent goal. Once messages from each team member have been broadcast confirming the establishment of the joint goal, execution can commence. Having the team plan as a joint persistent goal places obligations on team members to inform the team when that goal is dropped. The work of Pynadath *et al.* [13] extended STEAM to a more general framework for team-oriented programming using TEAMCORE.

Finally, the JACKTeams model released as part of the JACK platform [10] presents an extension of this approach, based on the work of Tidhar [18]. In the JACKTeams framework, (software) team agents are treated on a par with individual agents in having explicit team beliefs, goals and intentions. Team agent beliefs are both derived from, and propagate to, the beliefs of the individual agents in the team, and the team agent mediates the interaction between team members and acts as a central point of control. Thus the agents in the team do not need to be aware of one another, whereas SharedPlans theory requires teams with more autonomous agents which must know one another in order to negotiate and participate in team activities.

6 Conclusion

In this paper, we presented a general framework for implementing the SharedPlans theory of collaborative action, addressing computational issues such as team formation, group plan elaboration, and SharedPlan execution, which involves communication, coordination, synchronization and monitoring. The framework includes a team-oriented programming language for the specification of recipes for SharedPlans, and uses the JACK platform and BDI architecture for implementing the framework and interpreting SharedPlans expressed in the recipe language. Our overall aim has been to develop a generic computational teamwork model that is theoretically well-motivated and more directly related to the supporting theory.

References

1. Bratman, M.E. (1987) *Intention, Plans, and Practical Reason.* Harvard University Press, Cambridge, MA.
2. Bratman, M.E. (1992) 'Shared Cooperative Activity.' *The Philosophical Review*, **101**, 327–341.
3. Bratman, M.E., Israel, D.J. & Pollack, M.E. (1988) 'Plans and Resource-Bounded Practical Reasoning.' *Computational Intelligence*, **4**, 349–355.
4. Cohen, P.R. & Levesque, H.J. (1991) 'Teamwork.' *Noûs*, **25**, 487–512.
5. Grosz, B.J. & Hunsberger, L. (2004) 'The Dynamics of Intention in Collaborative Activity.' Paper presented at the Conference on Collective Intentionalty IV, Siena, Oct, 2004.
6. Grosz, B.J., Hunsberger, L. & Kraus, S. (1999) 'Planning and Acting Together.' *AI Magazine*, **20**(4), 23–34.
7. Grosz, B.J. & Kraus, S. (1996) 'Collaborative Plans for Complex Group Action.' *Artificial Intelligence*, **86**(2), 269–357.
8. Grosz, B.J. & Kraus, S. (1999) 'The Evolution of SharedPlans.' in Wooldridge, M. & Rao, A.S. (Eds) *Foundations of Rational Agency.* Kluwer Academic Publishers, Dordrecht.
9. Hadad, M. & Kraus, S. (1999) 'SharedPlans in Electronic Commerce.' in Klusch, M. (Ed.) *Intelligent Information Agents.* Springer-Verlag, Berlin.
10. Howden, N., Rönnquist, R., Hodgson, A. & Lucas, A. (2001) 'JACK Intelligent Agents™– Summary of an Agent Infrastructure.' Paper presented at the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS, Montreal, May, 2001.
11. Kinny, D.N., Ljungberg, M., Rao, A.S., Sonenberg, E.A., Tidhar, G. & Werner, E. (1994) 'Planned Team Activity.' in Castelfranchi, C. & Werner, E. (Eds) *Artificial Social Systems.* Springer-Verlag, Berlin.
12. Ortiz, C.L. & Grosz, B.J. (2002) 'Interpreting Information Requests in Context: A Collaborative Web Interface for Distance Learning.' *Autonomous Agents and Multi-Agent Systems*, **5**, 429–465.
13. Pynadath, D.V., Tambe, M., Chauvat, N. & Cavedon, L. (1999) 'Toward Team-Oriented Programming.' in Jennings, N.R. & Lespérance, Y. (Eds) *Intelligent Agents VI.* Springer-Verlag, Berlin.
14. Rao, A.S. & Georgeff, M.P. (1992) 'An Abstract Architecture for Rational Agents.' *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, 439–449.
15. Rich, C. & Sidner, C.L. (1998) 'COLLAGEN: A Collaboration Manager for Software Interface Agents.' *User Modeling and User-Adapted Interaction*, **8**, 315–350.
16. Scerri, P., Pynadath, D., Johnson, L., Rosenbloom, P., Si, M., Schurr, N. & Tambe, M. (2003) 'A Prototype Infrastructure for Distributed Robot-Agent-Person Teams.' *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 433–440.
17. Tambe, M. (1997) 'Agent Architectures for Flexible, Practical Teamwork.' *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, 22–28.
18. Tidhar, G. (1993) 'Team-Oriented Programming: Preliminary Report.' Technical Note 41, Australian Artificial Intelligence Institute, Apr, 1993.
19. Tidhar, G., Heinze, C. & Selvestrel, M. (1998) 'Flying Together: Modelling Air Mission Teams.' *Applied Intelligence*, **8**, 195–218.