Monitoring and Synchronization for Teamwork in GPGP

Sherief Abdallah
Computer Science Dept.
Faculty of Computers and Information
Cairo University, Egypt
shario@cs.umass.edu

Nevin Darwish
Computer Engineering Dept.
Faculty of Engineering
Cairo University, Egypt
ndarwish@alpha1-eng.cairo.eun.eg

Osman Hegazy
Information Systems Dept.
Faculty of Computers and Information
Cairo University, Egypt
Ohegazy@idsc.net.eg

ABSTRACT

This paper addresses the problem of coordinating a group of agents involved in a team. To achieve flexible teamwork, agents should synchronize their work and monitor their performance to avoid redundant work. Generalized Partial Global Planning (GPGP) is one of the most common techniques used in coordinating cooperative agents, however, no technique is without limitations. Our work adopts some concepts of STEAM to overcome some of GPGP limitations. In particular, we suggest adding coordination mechanisms to GPGP and extending TAEMS, the model underlying GPGP, to facilitate such mechanisms. The work has successfully been implemented using JAF architecture. The coordination mechanisms are written as Soar rules where we implemented a JAF component that implements the Soar engine. Analysis of a case study is presented along with experimental results to illustrate the power of the proposed work.

Categories and Subject Descriptors

H.4.m [**Information Systems**]: Miscellaneous

General Terms

Algorithms, Design, Reliability, Languages

Keywords

Coordination, Teamwork, GPGP, STEAM, SOAR

1. INTRODUCTION

The need to coordinate different agents in order to reduce redundant work, maximize utilization of shared resources, increase reliability of the system and enhance characteristics of the overall solution is one of the main issues of multi-agent systems research. Cooperative agents have shared goals that they plan to achieve. Generalized Partial Global Planning (GPGP) [3] is one of the techniques that provide coordination among cooperative agents. It consists of modular coordination mechanisms that use TAEMS model (Task Analysis, Environment Modeling, and Simulation) [2] to reason about the environment. The result of coordination is a set of contrainsts that modulate the operation of the agent's local scheduler. However, the GPGP mechanisms do not address special coordination needs in case of agents forming explicit teams. Tambe argued that to obtain flexibility and reliability in teamwork, explicit teamwork modeling is required [11]. His implemented teamwork model, STEAM, implements a hybrid of joint intentions [4] and SharedPlan theories [9].

In this work we suggest adding several coordination mechanisms, which implement some of the concepts introduced in STEAM, to GPGP. Some modifications of some of the GPGP standard mechanisms are also proposed. These extensions cause better synchronization between team members and reduce redundant teamwork. Throughout this paper we will use a simple case study for illustration, the Attack City domain. It is required to attack a city by an army consisting of two teams: air team with N members, and ground team with M members. The attack has two forms: air attack and ground attack. Air attack is undertaken by the air team. The air attack mission consists of three subtasks: fly to city, attack city military points, and finally return back to base. Ground attack is undertaken by the ground team and is divided into two subtasks: move to city, and takeover city. Air attack facilitates the mission of the ground team by eliminating much of possible resistance (i.e. air attack should complete before the start of ground attack to increase ground attack probability of success; however, the ground team might start its attack before air attack completion with more risk).

In the following sections we tackle this case study using both GPGP and STEAM frameworks, illustrating the main limitations of each. Section 2 presents an overview of GPGP and its underlying model, TAEMS followed by some of its limitations. Section 3 describes briefly STEAM framework along with its main limitations. Implemented extensions to GPGP are discussed in section 4. Empirical results are presented in section 5 followed by the conclusion and future work directions.

2. GPGP

GPGP defines a set of modular coordination mechanisms using a domain independent model, TAEMS. These mechanisms can be used in any combination depending on the domain of concern. The next subsections describes briefly TAEMS and GPGP mechanisms along with their limitations.

2.1 TAEMS

TAEMS model is composed of tasks and subtasks organized in a *task structure* hierarchy (also called *task group*). The leaves of the hierarchy are the *methods*, which are executable actions. Each method is characterized by three dimensions: duration, cost, and quality. Duration is the time spent executing the method. Cost represents the penalties of executing the method. Quality is an abstract measure to quantify the success of the method. Uncertainty of the three dimensions is modeled through discrete probability distributions. *Tasks* consist of subtasks. *Quality accumulation functions* (QAFs) define how the qualities of subtasks relate to the super task quality, e.g., q_sum QAF means the quality of the super task is the summation of subtasks' qualities. Tasks are linked by *interrelationships* (also called *non-local effects*) that model interdependencies between tasks [2, 7, 8]. Figure 1 shows a simplified TAEMS model for Attack City domain. Tasks are shown as rounded edge rectangles (e.g. attack city) while methods are shown as rectangles (e.g. fly_1). QAFs are indicated below tasks' rounded rectangles (e.g. Attack_City task has q_max QAF). Two types of non-local effects are shown, *facilitates* (e.g. the dashed arrow from Air_Attack task to Ground_Attack task), and *enables* (e.g the solid arrow from Fly_to_City to Attack_Targets). Enables interrelationship means that affecting task (source of the arrow) must complete successfully before the start of the affected task (destination of the arrow). Facilitates means it is advantageous for the affecting task to complete before the start of the affected task though not necessary.

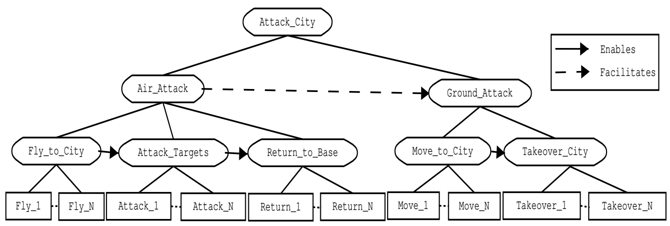

Figure 1: TAEMS model for Attack City domain

2.2 Coordination Mechanisms

Based on TAEMS model five coordination mechanisms were defined. The purpose of these coordination mechanisms is to recognize tasks' interdependencies and create corresponding commitments and constraints, which in turn is passed to the local scheduler to produce globally better schedules [3, 7]. The first mechanism deals with exchanging private views in order to construct a more global view. The second mechanism handles simple redundancy where a task's quality is the maximum of its subtasks' qualities and these subtasks are also methods. Third mechanism tackles results exchange. Fourth and fifth mechanisms create commitments in response to the presence of interrelationships (mainly enables and facilitates).

2.3 Limitations of GPGP

The main limitations of GPGP are as follows:

- Representing the contribution of each member (agent), in a given team task, by a subtask is inconvenient for large teams consisting of hundreds and even thousands of agents (e.g. the number of leaves in figure 1 grows as number of members per team increases).
- TAEMS does not provide an explicit way to express achievement of tasks performed by teams. For instance, for the subtask attack_targets, it might be sufficient to successfully destroy 70% of the targets. The lack of capability of expressing achievement conditions of overall team task might result in team members trying to complete their part in an already achieved task.
- Predefined GPGP mechanisms neither support synchronization of team members nor handle teamwork redundancy. Miss-synchronizing team members may cause severe penalties on teamwork. For instance, if an air agent started flying to city without making sure that other air agents also started flying to city, there is a large threat that the impatient air agent will be destroyed. Redundancy in teamwork is essential and is required all through the execution of the team task, and until the successful completion of the team task. In GPGP, redundancy avoidance occurs before starting the execution. Moreover, this simple mechanism deals with tasks having QAF max, i.e. only one subtask is necessary. In teamwork domains, redundancy is observed in tasks with QAF sum, i.e. each subtask (controlled by a member of the team) contribute somehow to the overall team's task and as more subtasks succeed, the quality of the team task increases.

3. STEAM

STEAM assumes the presence of a common team plan shared by members of the team. The plan is presented as a hierarchy of operators. Figure 2 shows operator hierarchy for the Attack City domain. The key novelty of STEAM is the introduction of team operators. Team operators define operators that should be performed by a team not individuals. These operators require special treatment in that members must synchronize before starting execution and once synchronized a member can not stop executing the operator until ensuring mutual, synchronized termination for all members. This is done through establishing a commitment protocol. A team task is considered terminated if it is believed to be achieved, unachievable, or irrelevant. STEAM attach termination conditions, which when evaluates to true the team task is believed to be terminated. Main limitations of STEAM are as follows:

- Roles relationships, introduced in STEAM, provide limited expressiveness. For instance, partial ordering between fly_to_city and attack_targets subtasks (fly_to_city precedes attack_targets) can only be modeled implicitly in the preconditions of the team operators: fly_to_ city and attack_targets.
- Uncertainty is not modeled explicitly except for communication. This limits the agents' ability to reason about 'risky' alternatives that might, exceptionally, lead to great advantages.
- Reactive nature of this model provides flexibility in responding to unexpected events. However, it is not

straightforward to use this model in reasoning about temporal constraints (deadlines, earliest start time, . . . etc.) in a straightforward way.

- The establish commitments protocol coordination cost (communication and processing) is proportional to n^2 where n is number of agents in a team. This disadvantage is more severe for domains using point to point communication.

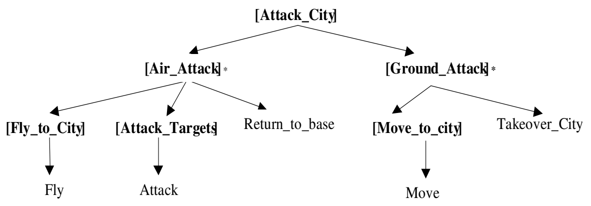

Figure 2: STEAM operator hierarchy for Attack City domain

4. PROPOSED EXTENSIONS TO GPGP

It is noted that GPGP solves most of STEAM limitations. First, the GPGP framework is more expressive in describing interdependencies between tasks than STEAM. Second, TAEMS provides an explicit model of uncertainty (through probability distributions of quality, duration, and cost), which enables agents to reason about risky tasks having high profits, against safe tasks with lower profits. Third, in GPGP agent's actions are scheduled, enabling agents to reason about duration and imposed deadlines. Hence, we have chosen the GPGP framework as the basis of our work. We present some extensions to GPGP mechanisms and its underlying TAEMS model, and then give a brief overview of our implemented agent architecture.

4.1 TAEMS Extensions

The proposed extensions to TAEMS are two: modeling organizational information and defining monitoring conditions.

4.1.1 Modeling Organizational Information

TAEMS specification [8] does not provide a compact way of representing teams and the contribution of each team member in a team task (first limitation). We suggest a new TAEMS object, template. A template is any TAEMS object defined over a whole team not only a single agent. In general, this extension drops the number of objects required to define a task structure to a fixed number of templates. For example, in task *fly_to_city*, each agent A_i contributes by executing his method *Fly_i*. A template *Fly_tmpl* replaces methods *Fly_1*, . . . , and *Fly_N*. As shown in figure 3 the number of nodes at the lowest level in the task hierarchy has dropped from $(3N + 2M)$ to only 5 nodes. Method templates are shown as circles .

4.1.2 Defining Achievement and Unachievability Conditions for Team Tasks

TAEMS specification does not define conditions of achievement and/or unachievability for a task. In standard TAEMS, there is only one (implicit) achievement condition applied

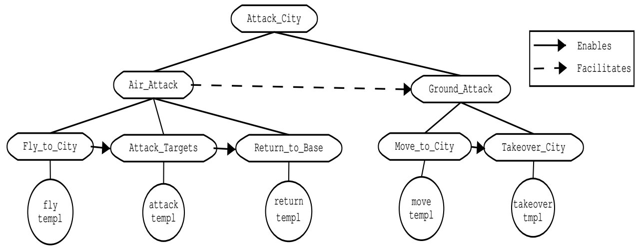

Figure 3: TAEMS model fo Attack City domain with templates introduced

to all tasks: task quality is greater than zero. However, TAEMS still provides us with abstract measures (referred to afterwards as *monitoring terms*) that can be used to specify achievement and/or unachievability conditions of a team task, such as:

- Expected distributions[1] of quality, duration, cost, start time and finish time.
- Uncertainty represented in probability distributions of quality, duration, ..etc.
- Current (actual) values of quality, duration, cost, start time and finish time.

For example, the following condition specifies that a task T is unachievable if average expected quality is less than 100, average expected cost greater than 10 and probability that T misses deadline is greater than 0.7: [q(T) ¡ 100 AND c(T) ¿ 10 AND p(ft(T)¿dl(T)) ¿ 0.7] where q(T) and c(T) are average expected quality and cost (respectively) of task T , ft(T) is expected distribution of finish time of T, dl(T) is deadline constraint over T.

This clear and formal definition of monitoring terms can be contrasted to the approach used in STEAM where monitoring conditions can be of any form. The lack of formal definition degrades the reusability of coordination mechanisms that respond to a specific monitoring condition. For instance, we can define a coordination mechanism that warns team members when the probability that their team task becomes unchaievable goes above 0.6.

4.2 GPGP Extensions

We suggest three extensions to GPGP mechanisms: a new synchronization mechanism, a new monitoring framework, and modifying the "communicate results" existing mechanism. We have implemented the three suggested extensions.

4.2.1 Adding Synchronization mechanisms

We added a new GPGP mechanism to ensure synchronization between team members, which is based on *establish commitments* protocol (ECP). Figure 4 shows part of the protocol as a finite state machine. It is noted that the complete protocol has two roles: the team leader and a normal member of the team. There is a similar mechanism for terminating team tasks. Apparent from figure 4 the leader role is critical (in synchronization and other mechanisms); hence we added a recovery mechanism that detects leader

[1]The expected probaility distribution given the current beliefs of the agent (e.g. schedules, commitments, . . . etc.)

failure (using timeouts) and replaces the failing leader with a functioning agent (according to a static leadership hierarchy).

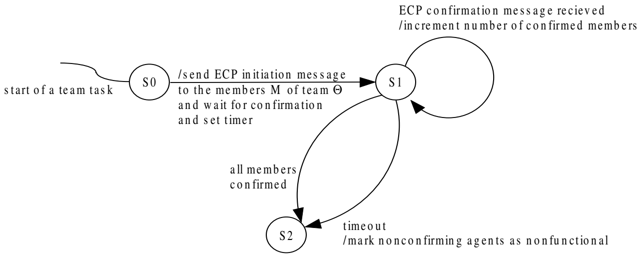

Figure 4: FSM representing leader role in joint commitment protocol

4.2.2 *Adding Monitoring Framework*

We also added a monitoring framework to monitor team task termination. Once an agent believes either its team has achieved the currently executing task or the currently executing task is unachievable, the agent terminates its part of it. Our monitoring framework differs from STEAM's monitoring framework in the use of monitoring terms. It should be noted that we did not implement recovery mechanisms introduced in STEAM. Currently, once an agent recognizes the task is unachievable it terminates it and goes on executing any remaining, achievable task. We plan to add a comprehensive recovery capability in the future.

4.2.3 *Modifying Communicating Results*

In GPGP second mechanism for *communicating results*, each *affecting* agent (committed to execute an affecting task) is responsible for communicating results to each *affected* agent (executing an affected task). For instance, for the attack_targets task each air agent is responsible for communicating the success (or failure) of his mission to:

- Each air agent, due to the enables interrelationship between attack_targets and return_to_base, resulting in N^2 messages.
- Each ground agent, due to the facilitates interrelationship between air_attack and ground_attack, resulting in $M \times N$ messages.

We took advantage of the teams' organization and modified the mechanism such that only the leader communicates results on behalf of his team to affected agents that are not members of the team (members of the team have the same knowledge of the team leader). So in the Attack City case study, each air agent will be responsible for communicating the success (or failure) of its mission to each air agent, due to being member of the air team, still resulting in N^2 messages. However, only the leader of the air team is responsible for communicating the results of the air_attack task, after its termination, to each ground agent, resulting in M messages instead of $3M \times N$ messages.

4.3 Implementation

We used JAF [5] framework as basis for our implementation. We used some of the standard JAF components, e.g. the scheduler component (to schedule the tasks given the new contrainsts), the communication component (to enable agents to communicate together), ... etc. We implemented a coordination component, which provides basic functionality for coordination mechanisms through a common substrate. Coordination mechanisms themselves are implemented as Soar rules [6]. Recent work, namely GPGP2 [12], implemented this substrate as a Finite State Machine engine. Coordination mechanisms were defined in terms of finite machine scripts that is parsed and compiled and linked to the substrate java code. We took a completely different approach. Our coordination substrate encapsulates a Soar [6] compiler and run-time engine. This provides more flexibility as coordination mechanisms can be written in the form of production rules. Furthermore, caching of partial matching results is supported (through RETE network). This particular property is important to facilitate the monitoring of achievement and unachievability conditions, which is continually changing throughout the execution. Figure 5 shows the anatomy of the module. A brief description of each component of the module is given below.

- *Soar Engine.* This is the core of the coordination module. It implements Rete network algorithm [10] and the Soar architecture. Coordination mechanisms are written as Soar production rules that is compiled and executed by our implemented engine. More details about our implementation can be found in [1].
- *Soar/Java Translator.* This component translates data structures from java classes to the corresponding Soar working memory elements (WMEs) and vice versa. It is used whenever a data structure should be transferred to/from the Soar engine.
- *Message Listener.* This component allows coordination rules (mechanisms) to register the message header(s) they are interested in. This is important to avoid translating unimportant messages. When a message with a registered header is received, the message is translated and passed to the Soar Engine.
- *Property Listener.* Through this component, coordination mechanisms can register the names of the properties they are interested in. Whenever a registered property is added, changed, or removed, the event is translated and passed to the Soar Engine.
- *Command.* The command component encapsulates interface with other JAF components. When a Soar rule requires to schedule a task structure, terminate currently executing action, ... etc. it asks the Command components.

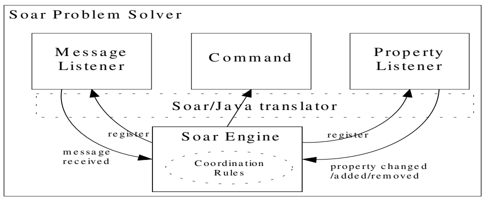

Figure 5: Components of the coordination module

5. RESULTS

In our results we explore the relationships between uncertainty and teamwork miscoordination. Three coordination approaches that rely on the GPGP framework are considered. In the first approach, both the monitoring framework and the synchronization mechanism are disabled. The second approach is same as the first approach but with the synchronization mechanism added. The third approach, which is our proposed approach, has both, the synchronization mechanism and the monitoring framework, enabled. Using the Attack City case study, the three approaches were tested under different degrees of uncertainty (to be described shortly) and at different team sizes. For each combination (team size and degree of uncertainty) 10 simulation runs were executed and averaged to compute different measures. First we will define two terms: degrees of uncertainty and teamwork miss-synchronization. Next we present the results, which ensure the need for a monitoring framework in addition to synchronization.

5.1 Degrees of Uncertainty

In TAMES, uncertainty is modeled through the probability distributions of quality, cost, and duration. We will focus our attention on duration uncertainty as one of the major causes of teamwork miscoordination. We tested different coordination approaches against four duration distributions:

- Distribution A: P(5)=1 (i.e. the probability that the method duration takes 5 time units equal 1)
- Distribution B: P(4)=0.9 and P(14)=0.1
- Distribution C: P(4)=0.8 and P(9)=0.2
- Distribution D: P(3)=0.5 and P(7)=0.5

We deliberately adjusted each distribution such that the average duration remains constant, 5, to focus attention on the uncertainty.

5.2 Teamwork Miss-Synchronization

In order to evaluate teamwork coordination mechanisms, we need to define a measure of teamwork miss-synchronization. Previous work of Tambe did not define formal measure of teamwork miss-synchronization (Tambe focused on communication costs). We propose below a scalar measurement for teamwork miss-synchronization (TMSM) based on heuristics.

Teamwork miss-synchronization occurs when members M of team Θ either start or terminate a team task T_i at different times. In other words, Θ split into subteams $S_{i1}, S_{i2}, ., S_{in_i}$ where n_i is the number of subteams for T_i. Function $C(\Theta)$ returns cardinality of a team Θ (number of members). The proposed measure, $TMSM(T_i)$, should satisfy the following criteria for two tasks T_1 and T_2 performed by same team Θ:

- If $n_i = 1$ then $TMSM(T_i) = 0$
- If $n_1 > n2$ then $TMSM(T_1) > TMSM(T_2)$
- If $n_1 = n_2 = n$, and $\exists S_{1i}$ such that $\forall j, 1 \leq j \leq n, C(S_{1i}) > C(S_{2j})$ then $TMSM(T_1) < TMSM(T_2)$

We define our measure of teamwork miss-synchronization for task Ti to be:

$$TMSM(T_i) = 1 - \sum_{j=1}^{n_i} \left(\frac{C(S_{ij})}{C(\Theta)} \right)$$

This measure is domain independent compared to actual penalties imposed on miss-synchronized team, which is domain specific.

5.3 Lower Teamwork Miss-Synchronization

Figure 6 shows average TMSM when synchronization mechanism is disabled (original GPGP). Average TMSM is plotted against number of agents per team and for different degrees of uncertainty. Generally, the uncertainty decreases from degree D down to degree A. The curve MAX shows the maximum miss-synchronization (when all agents start as one team and end up with each agent consitituting a seperate subteam). As expected, as number of agents increases, TMSM increases. It is less probable that all agents start a team task at the same time as number of agents increases. This observation becomes more evident from figure 7, which shows the average possibility (without explicit coordination) that all members of a given task start/terminate their team task at the same time. In addition, an early miss-synchronization will, usually, cause subsequent miss-synchronization in following team tasks (hence increasing the overall average TMSM). The curve plotted for degree of uncertainty D is the closest to the upper bound (MAX curve), confirming that as uncertainty increases TMSM increases. With synchronization mechanism enabled, TMSM drops to zero assuming reliable communication.

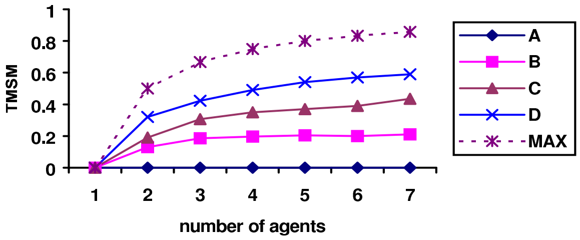

Figure 6: TMSM without synchronization

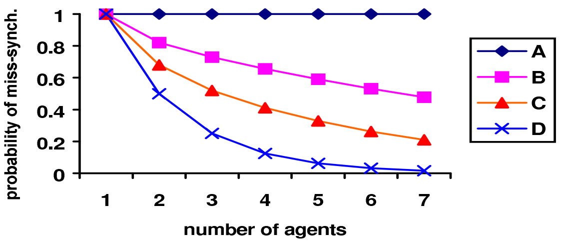

Figure 7: Probability of joint start/termination without synchronizing

5.4 Lower Average Execution Time

Figure 8 shows average execution time per method divided by mean (for normalization) and compares the three approaches under degree of uncertainty D. In this degree of uncertainty, 50% of the time the task takes 3 cycles, i.e.

the agent finishes early. In the other 50%, the agent finishes late. By execution time we mean the time interval since an agent starts executing a method in service of a team task and until the same agent believes the team task of service has completed. We ignored communication latency, which is negligible compared to methods duration (flying to a city, returning to base, ... etc.). For our approach (with synchronization and monitoring enabled) a fixed achievement condition is used, team-task quality *ge* 9. We modeled each agent's contribution by 5 quality units; hence at least two agents must complete the task before considering it achieved. As expected, as number of agents per team increases, redundancy increases and hence our proposed approach shows more savings (it becomes more likely that two members of the team finishes early). With only synchronization mechanism enabled, average execution time increases as number of agents increases. This is mainly because agents that finish early must wait for slower agents to finish. Using only the GPGP's original mechanism (with no synchrnization or monitoring), average execution time is independent of the number of agents. This is because agents are independent from each other. The upper limit of the average execution time is 1.4, which is the maximum possible duration of a method, 7, divided by the expected duration, 5. Similarly, the lower limit of the average execution time is 0.6, which is the minimum possible duration of a method, 3, divided by the expected duration, 5.

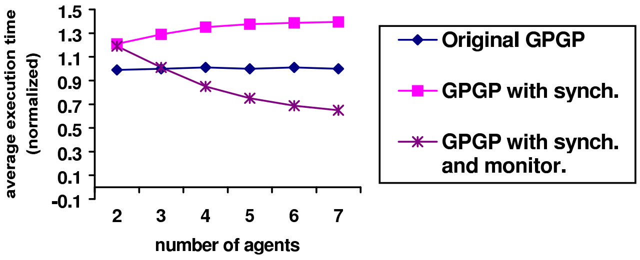

Figure 8: Average execution time for the three approaches

5.5 Conclusion

This work illustrates how GPGP and its underlying model, TAEMS, can be used to reason about teamwork. It takes STEAM as an example of teamwork oriented coordination framework, then illustrates how GPGP along with TAEMS can be extended to provide same capabilities. The implemented coordination module substrate supports rule-based coordination mechanisms, thus providing more flexibility in the design of coordination mechanisms. A new GPGP mechanism has been added to support synchronization between team members. We also proposed extensions of TAEMS to explicitly support achievement and unachievability conditions of tasks and implemented a framework supporting it to monitor team tasks performance.

While achievement and unachievability conditions are originally used for recognizing the termination of a task, how these conditions affect the scheduler performance is an interesting area of future work. Normal schedulers unaware of these conditions might prefer schedules that are unachievable from the start. This triggers the need for more flexible schedulers.

For large teams with hundreds of members, scalability is a key issue. It is clear from the experiments that imposing a hierarchical organizational structure over TAEMS task structure is essential to cope with complexities. While the presented framework supports hierarchical organization of teams and subteams, we aim at testing our framework in large variety of domains to measure the generality, flexibility, and reusability of our framework.

6. ADDITIONAL AUTHORS

Additional authors: Ihab Talkhan (Computer Eng. Cairo University, email: `ihat@alpha1-eng.cairo.eun.eg`).

7. REFERENCES

[1] S. Abdallah. *Soar2Java package*. World Wide Web, http://www.geocities.com/sharios/soar.htm, 2000.

[2] K. Decker. *Environment Centered Analysis and Design of Coordination Mechanisms*. Ph.D. thesis, University of Massachusetts, 1995.

[3] K. Decker and V. Lesser. Designing a family of coordination algorithms. In *ICMAS*, 1995.

[4] B. Grosz and S. Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86(2):269–357, 1996.

[5] B. Horling. *A Reusable Component Architecture for Agent Construction*. M.Sc. thesis, University of Massachusetts, 1998.

[6] J. Laird, A. Newell, and Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[7] V. Lesser and et al. *Evolution of GPGP Domain Independent Coordination Framework*. Technical Report 1998-5, University of Massachusetts, 1998.

[8] V. Lesser and et al. *TAEMS White Paper*. World Wide Web, http://mas.cs.umass.edu/research/taems/white/, 1998.

[9] H. Levesque and et al. On acting together. In *AAAI-90*, Boston, MA, 1991.

[10] P. Nayak and et al. Comparison of the rete and treat production matchers for soar. In *AAAI-88*, pages 693–698, 1988.

[11] M. Tambe. Towards flexible teamwork. *Artificial Intelligence Research*, 7:83–124, 1997.

[12] T. Wagner and et al. *Investigating Interactions Between Agent Conversations and Agent Control Components*. Technical Report 1999-7, University of Massachusetts, 1999.