Plan Execution and Coordination

Pedro Szekely
Robert Neches
University of Southern California

Marcel Becker
Stephen Fitzpatrick
Kestrel Institute

Chris van Buskirk
Doug Fisher
Gabor Karsai
Vanderbilt University

Abstract

We investigate the problem of keeping the plans of multiple agents synchronized during execution. We assume that agents only have a partial view of the overall plan. They know the tasks they must perform, and know the tasks of other agents with whom they have direct dependencies. Initially, agents are given a schedule of tasks to perform together with a collection of contingency plans that they can engage during execution in case execution deviates from the plan. During execution, agents monitor the status of their tasks, adjusting their local schedules as necessary and informing dependent agents about the changes. When agents determine that their schedule is broken or that a contingency schedule may be better, they engage in coordinating plan changes with other agents. We present a "dynamic partial centralization" approach to coordination. When a unit detects a problem (task delay, inability to perform a task), it will dynamically form a cluster of the critically affected agents (a subset of all potentially affected agents). The cluster will elect a leader, who will retrieve all task and contingency plan information from the cluster members and compute a solution depending on the situation.

Introduction

Plan execution often involves a collection of agents, such that each agent gets a copy of its own plan, but does not know the overall plan, for all the agents. This is a very common situation in the military, where fielded human units do not have access to the full plan. Equally often, plan execution fails and there is a need for repairing the plans of the individual agents such that the overall goals of the agent ensemble are achieved. In the case of human agents this repair is facilitated by a coordination process that includes rapid communication and ad-hoc adaptation of plans by humans. In our work, we are looking at computational approaches that tie monitored plan execution to rapid plan repair and coordination among autonomous executor/planning agents.

We assume that an active component: a coordinator agent, is monitoring the execution of its plan. It receives notifications from its external world about events that could indicate the success or failure of plan execution. When the execution of its plan fails at a specific point in time, it goes into a coordination mode, where it computes changes to its plan such that the overall goals of the plan are achieved. The problem is that the plans of the individual coordinators were created dependencies among them, and failure in one plan step may impact the agent but also other, dependent agent's plans as well. Our interest is to develop distributed algorithms that facilitate rapid coordination: plan repair across a multitude of related and dependent coordinator agents.

Our approach to coordination is "dynamic partial centralization". When a unit detects a problem (task delay, inability to perform a task), it will initiate a process that will dynamically form a cluster of the critically affected units (a subset of all potentially affected units). We believe this cluster formation is crucial for improved performance. The problem of coordination can be solved in a fully centralized or in a fully distributed way. In the first case, a central server is needed (organizationally unacceptable for our domain), in the second case a distributed constraint solving process can be used (which has performance problems). The cluster solution combines the advantages of the centralized approach (better performance) without the problems of the fully distributed solution (using too much communication).

Modeling

Our work utilizes the TAEMS modeling framework (Decker and Lesser 1993). TAEMS models plans hierarchically: the leaves of the tree are called *methods*, and represent activities that agents can perform directly. The internal nodes are called *tasks*, and represent procedures to combine multiple activities (tasks or methods) to achieve higher level goals.

The TAEMS formalism offers several features appropriate for modeling the dynamics of a real world execution environments. We present the main concepts here (formal specifications are described in (Decker and Lesser 1993), details of the TAEMS framework are described in (Lesser *et al.* 2004)).

1. *Quality*: methods define a probability distribution for the quality that results when a method is executed.
2. *Duration*: methods define a probability distribution for the amount of time it takes to perform a method.
3. *Cost*: methods define a probability distribution of the cost that will be incurred when executing a method.

Each task defines a *quality accumulation function* (qaf) that specifies how the quality of the task is computed based on the quality of the children. TAEMS offers a collection of about 20 qafs. The most relevant to our work are:

- *Min*: the quality of a task is the minimum of the quality of the children. Min corresponds to the traditional And logical operator given that unattempted or failed tasks receive quality zero.
- *Max*: the quality of a task is the maximum of the quality of the children. Max corresponds to the traditional Or logical operator given that the quality of the task will correspond to the best quality of any attempted subtask.
- Sum, Xor, Sequence, etc. enable modeling of other situations that arise in real world applications.

TAEMS also provides a capability to define interrelationships among tasks:

- *Enables, Disables*: these are hard constraints among tasks or methods. If A enables B, then attempting to perform B before A completes will result in B failing and accumulating zero quality. Disables is defined similarly.
- *Facilitates, Hinders*: these are soft constraints. If A facilitates B, then when B is executed, its quality will be multiplied by a facilitation factor that depends both on a power factor defined in the Facilitates relation and the percentage of maximum quality that A obtained. Hinders is defined similarly.

The TAEMS model of agents is very simple. Methods and Tasks can be associated with a collection of agents that can perform it. During execution, methods can only be performed by a single agent, so part of the planning/coordination process involves selecting a single agent from the collection of possible agents.

TAEMS distinguishes between subjective and objective views of the world. The subjective view of the world is a TAEMS structure that defines the portion of plans that an agent knows about. Initially, agents are given a TAEMS structure containing the tasks and methods where the agent is listed, as well as all tasks an methods of other agents directly linked to the agent's tasks and methods (via Enables, Disables, Facilitates, Hinders and Parent relationships). During execution agents may communicate their subjective structures to other agents. The objective view is a conceptual entity containing all TAEMS structures of all agents. It may not be known to any agent, but for experimentation, the objective view is known to a simulator.

Plan Execution

The main objective of our work is to build agents that reason in real-time about the outcomes of method execution (quality, duration and cost) and adjust their plans in order to optimize the quality of the root level goal of the plan while staying within cost deadline and cost constraints.

Our work does not assume that execution follows the follows the subjective models faithfully. Our system is reactive and will always attempt to optimize the final outcome with respect to the current state, irrespective of how the current state came to be.

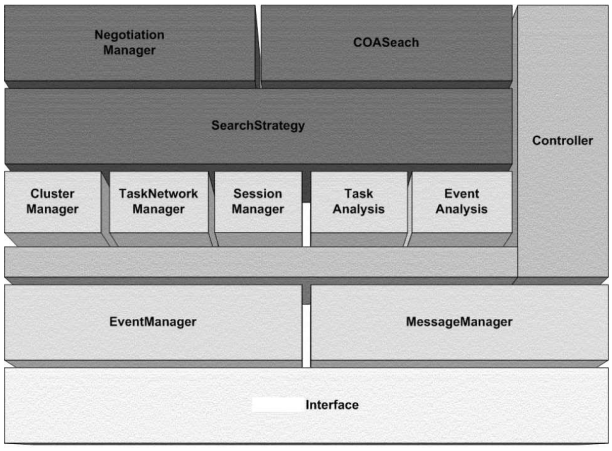

Figure 1: Architecture

Assumptions

The main assumption of our work is that the agents model human activities, whose duration is typically measured in minutes. This is in contrast to sensor network domains where time frames are in the order of seconds or less (e.g., react to an incoming missile).

The human activity assumption means that response times for adjusting plans can often be in the order of tens of seconds. For example, if I miss my flight at the airport, it is OK to wait 10 or 30 seconds for a new plan that directs me to take an alternative flight, redirects me to a new city, etc.

Architecture

Figure 1 shows the main components of an individual agent. The architecture builds upon an Interface that links the agent to the external world. The agent receives events in the Event Manager and sends and receives messages via the Message Manager. Events inform the agent about the success or failure of the execution of plan steps, while messages are used to communicate (and coordinate) with other agents. These managers convert events and messages into the internal representations of our system and give control to the Controller for further processing.

Events trigger Event Analysis to determine if there is a potential impact of the event on the plan of the agent. For example, the event may indicate that a method that a task depends on has not finished on time as expected. In such cases Event Analysis produces a task impact report and gives control back to the Controller.

Task impact reports are processed in the Task Analysis component. This component uses a low cost distributed constraint propagation algorithm to repair the schedule when execution deviates from the plan scheduled. When a task or method slips, this algorithm will tighten the start window for dependent tasks and methods. When the window of a task or method owned by a different agent is tightened, then a message is sent to the other agent to force it to tighten its representation of the time window. The effects on time windows are propagated as necessary. If the schedule has

enough slack to absorb the delay, then the propagation will die out. If a start window becomes empty (i.e., a task or method should finish before it starts), then the impact of the triggering event cannot be absorbed by simply adjusting the start time of the methods in the current schedule. At this point the impacted agent must engage a more sophisticated search algorithm that will change the methods for achieving the current goals. This search is started by the producing a clustering report and returning control to the Controller.

The clustering reports are first processed in the Cluster Manager, which initiates the cluster formation processes, by defining a new cluster that initially contains just the task whose start window is empty. A search report is produced and control returns to the Controller.

The Search Strategy initiates the cluster-based search algorithm by first expanding the cluster to contain the neighbors of the seed element of the cluster, and then selects a COA Search (Course of Action Search) algorithm to try to find a feasible schedule within the elements of the cluster, but without changing the time windows or dependencies for any of the tasks or methods in the boundary of the cluster. If such a solution exists it a search report is produced containing the new solution. If no solution is found or a given time threshold is exceeded, then a failure search report is produced requesting cluster expansion.

The Cluster Manager expands clusters following dependency links on the tasks and methods already in the cluster and sending messages to their agents to query the corresponding TAEMS structures. The Cluster Manager ensures that no task or method belongs to more than one cluster.

When the request to grow a cluster fails (clusters collide) the Search Strategy decides whether to merge clusters, nominating one of the leaders as the leaders of the new merged cluster, or whether to stop centralization and engage in distributed constraint satisfaction between the leaders of the colliding clusters. These interactions are governed by the Negotiation Manager that implements a negotiation protocol among cluster leaders.

When the COA Search algorithms stop, producing a new schedule (even if it is only a partial schedule), the leader will distribute the new COA to all the cluster members, who will incorporate it into their subjective view. The receiving agents will apply all events that have arrived since the cluster-based search process started.

Given the human activity assumption, it is expected that in the majority of cases cluster sizes can be kept small enabling the use of fast, centralized solutions techniques that enable leaders to quickly compute high quality solutions before the world changes in a significant way. As a last resort, if merging of clusters would result in large clusters, leaders will negotiate using slower, less optimal distributed constraint satisfaction techniques.

Depending on circumstances, human users may need to approve these plans. For this reason, plan choices are presented in a ranked order of utilities to a human, who can then approve the chosen plan.

Conclusions and Status

The approach presented here represents an interesting compromise between approaches based on traditional planning representations (e.g., PDDL 2.2) and an approaches based on a task-oriented representation such as TAEMS. TAEMS is not a planning language: preconditions and effects of methods are not defined explicitly . The effects are encoded implicitly in the inter-relationships among tasks and methods. TAEMS is expressive enough to encode contingency plans, i.e., alternative ways to accomplish goals. Each of the contingency plans is a fragment of a larger plan and they must be combined into a consistent plan. However, TAEMS provides a good model of task and method execution.

The partial centralization approach for distributed coordination has been explored before (Scerri *et al.* 2004). Our approach is more closely related to Mailler's work on cooperative mediation of distributed constraint optimization (Mailler and Lesser 2004).

This paper presents the approach for a new system. We have designed the system architecture, have high level ideas for the search algorithms involving centralized search over TAEMS structures. We plan to demonstrate the system created on small scale, abstract examples by the end of the current year (Nov 2005).

Acknowledgments

The work presented here is funded by the DARPA COORDINATORS Program under contract FA8750-05-C-0032. The U.S.Government is authorized to reproduce and distribute reports for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

References

Keith S. Decker and Victor R. Lesser. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance, and Management*, 2(4):215–234, 1993.

V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. NagendraPrasad, A. Raja, R. Vincent, P. Xuan, and X.Q. Zhang. Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143, July 2004.

Roger Mailler and Victor Lesser. Solving Distributed Constraint Optimization Problems Using Cooperative Mediation. In *Proceedings of Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 438–445. IEEE Computer Society, 2004.

Paul Scerri, Regis Vincent, and Roger Mailler. Comparing Three Approaches to Large Scale Coordination. *Proceedings of the First Workshop on the Challenges in the Coordination of Large Scale Multi-agent Systems*, July 2004.