A Multiagent Planning Architecture

David E. Wilkins and Karen L. Myers
Artificial Intelligence Center, SRI International
333 Ravenswood Ave., Menlo Park, CA 94025
email: {wilkins, myers}@ai.sri.com

Abstract

The Multiagent Planning Architecture (MPA) is a framework for integrating diverse technologies into a system capable of solving complex planning problems. Agents within MPA share well-defined, uniform interface specifications that facilitate integration of new technologies and experimentation with different problem-solving strategies. MPA provides a central repository for storing plan-related information in a shared plan representation, and metalevel agents that control and customize the interactions between other agents. The MPA framework has been validated through its use in developing several large-scale problem-solving systems for Air Campaign Planning.[1]

Introduction

The Multiagent Planning Architecture (MPA) is a framework for integrating diverse technologies into a system capable of solving complex planning problems. MPA has been designed for application to planning problems that cannot be solved by individual systems, but rather require the coordinated efforts of a diverse set of technologies and human experts. MPA agents can be sophisticated problem-solving systems in their own right, and may span a range of programming languages. MPA's open design facilitates rapid incorporation of tools and capabilities, and allows the planning system to capitalize on the benefits of distributed computing architectures for efficiency and robustness.

Agents within MPA share well-defined, uniform interface specifications, making it possible to explore a broad range of cooperative problem-solving strategies. This paper describes two areas in which such exploration was undertaken. Sophisticated systems for planning and scheduling have been decomposed into modules, each of which has been transformed into an agent, allowing experimentation with different degrees of coupling between the planning and scheduling capabilities. A second area is the definition of organizational units for agents that permit flexible control policies in generating plans. Within MPA, notions of *baselevel planning cells* and *metalevel planning cells* have been defined, where the baselevel cells provide sequential solution generation and the metalevel cells employ baselevel cells to support parallel generation of qualitatively different solutions. Metalevel cells provide the ability to rapidly explore the space of solutions to a given planning problem.

The MPA framework has been used to develop several large-scale problem-solving systems for the domain of Air Campaign Planning (ACP). One such application integrated a set of technologies that spanned plan generation, scheduling, temporal reasoning, simulation, and visualization. These technologies cooperated in the development and evaluation of a complex plan containing more than 4000 nodes. This integration has validated the utility of MPA for combining sophisticated stand-alone systems into a powerful integrated problem-solving framework.

MPA is distinguished from other agent architectures (such as (Moran *et al.* 1997)) in its emphasis on application to *large-scale planning* problems. The architecture includes agents designed specifically to handle plans and planning-related activities. Interagent communication protocols are specialized for the exchange of planning information and tasks. Another distinguishing feature of MPA is the emphasis on facilitating the integration of agents that are themselves sophisticated problem-solving agents. One of the primary goals of MPA is to facilitate such integrations. Most agent architectures develop specialized agents that are suited for operation within that specific architecture rather than incorporating legacy systems.

MPA provides the infrastructure necessary to support a broad range of distributed planning capabilities. At present, however, it does not include mechanisms for coordinating subplans generated by distributed planning agents (Georgeff 1984). We intend to explore algorithms for distributed planning in the future, and believe that our infrastructure will support them.

MPA Overview

MPA is organized around the concept of a *planning cell*, which consists of a collection of agents committed to a particular planning process. A planning cell contains two distinguished agents — the *planning-cell manager* and the *plan server*. The *planning-cell manager* composes a planning cell from a pool of available agents and distributes the

Figure 1: Sample Organization of a Planning Cell

planning task among the selected agents. The *plan server* is the central repository for plans and plan-related information during the course of a planning task. It accepts incoming information from planning agents (PAs), performs necessary processing, stores relevant information, and makes this information accessible to any PA through queries. An optional *domain server* stores application-specific information of relevance to the planning process, which may be shared among all cell agents. The domain server might store, for example, situation models, an object hierarchy, legacy databases, and the set of action descriptions to be used in planning.

The MPA architecture rests upon a significant amount of infrastructure. One component is a *shared plan representation* that can be understood by all planning agents. MPA employs the Act formalism for this purpose (Wilkins & Myers 1995). Additional components include a communication substrate to support asynchronous interagent message passing across networks, and tools to facilitate the construction of agents and planning cells. MPA provides both a message format and a message-handling protocol to support the sharing of knowledge among agents involved in cooperative plan generation.

Planning Cells

MPA planning cells are hierarchically organized collections of planning agents that are committed to one particular planning process for a given period of time (see Figure 1). Each planning cell has a meta-PA (a PA that controls other PAs) that serves as the *planning-cell manager* (or, simply the *cell manager*), which decomposes a planning task and distributes it to the PAs of the planning cell.

A cell manager may distribute tasks to both PAs and other meta-PAs. The planning cell is the closure of all PAs registered by the cell manager along with any PAs they invoke. Individual cell managers compose their planning cells to reflect their own planning approach. For example, a mixed-initiative cell manager may include human agents in its cell. One cell manager might loosely couple planning and scheduling, while another might tightly couple them.

The cell manager maintains a Planning-Cell Designator (PCD) that defines a set of *roles* to be filled. A role constitutes a capacity or responsibility, such as plan generation or scheduling. Each role must be filled by an agent with the capabilities required by that role. The PCD records the name of the agent that fulfills each role in the current planning cell at any given point in time. (The agent playing a role may change during the planning process.) The cell manager stores and maintains the PCD, broadcasting changes to the cell agents as appropriate. Local copies of the PCD are maintained by each agent to eliminate the need to query the cell manager for current role assignments prior to sending each message (thus substantially reducing interagent message traffic).

Composing a planning cell involves establishing communication with an agent for each role specified in the PCD, where these agents are then committed to this particular planning process. Currently, the MPA cell manager can employ a prespecified planning-cell configuration, or have a human user compose the cell from available agents.

As MPA expands to include different technologies with similar or overlapping capabilities, it will be desirable to define a *capabilities model* for each agent that describes the functionality and I/O requirements for the agent. A cell manager could then compose a planning cell automatically by broadcasting for agents with the required capabilities. Most current technologies for planning have capabilities or input languages that differ substantially from related technologies, so automated construction of a planning cell based on capabilities descriptions is not yet practical.

Plan Model

The MPA plan model distinguishes the concepts of task, plan, and action network.

A *task* is defined by a set of objectives that is to be achieved through planning. An *action network* is a partial order of *nodes*, where a node represents a planning entity, such as an action, goal, or condition. Action networks can span multiple levels of abstraction, and can represent both intermediate and final collections of actions for a plan. A *plan* consists of a set of linked action networks that encode some or all of the derivation structure for a plan. Additionally, a plan may include information that was used in generating the plan, such as assumptions, planning advice, or control rules. A number of alternative plans can be produced for a given task.

MPA Communication

MPA provides a rich interagent communication framework, composed of a set of MPA specific protocols layered on top of a lower-level communication substrate. Three software packages can provide this substrate for MPA: the Knowledge Query and Manipulation Language (KQML)(Finin *et al.* 1992), Inter-Language Unification (Janssen *et al.* 1997), and the Open Agent Architecture (Moran *et al.* 1997). MPA

is usually run on top of KQML, but preliminary implementations on top of the other two also exist.

MPA provides both a message format and a message-handling protocol to support the exchange of knowledge and requests among agents involved in cooperative problem-solving. The message structures incorporate KQML's notion of performatives, referred to as *communication performatives* within MPA. MPA further specifies its own set of *plan performatives*, which specialize KQML messages to planning activities. Combinations of communication and plan performatives define the message protocols that enable agents to interact with each other.

An MPA message consists of a communication performative and a set of fields specified as keyword/value pairs. The two most important fields are :content and :reply-with. The :reply-with field indicates whether or not a reply is expected by the sending agent. The :content field consists of the plan performative, an optional value, and a set of optional keyword/value pairs for that plan performative.

Below are two sample messages sent by the cell manager to the plan server. The first is used to verify that the receiving agent is alive; for this reason a reply is requested. A sample response is provided. The second message informs the receiving agent of a new PCD; it requires no response.

```
(EVALUATE :SENDER PCM :REPLY-WITH PING
          :CONTENT (:PING))
(REPLY :SENDER "plan-server"
       :CONTENT (:PING :OK))

(TELL :SENDER PCM :CONTENT
 (:PCD ((:MANAGER "pcm")(:PLANNER "sipe")
        (:SCHEDULER "opis")
        (:PLAN-SERVER "ps1"))))
```

These messages are handled by all agents. Additionally, agents service messages that are specific to the roles that they are filling (as described below).

MPA provides wrappers and agent libraries that support interagent message-passing, multi-threaded processing, tracing of messages, and logging of messages to a history file. These wrappers are designed to facilitate rapid integration of new technologies into MPA. Other sites have been able to used these wrappers to convert legacy LISP programs into communicating MPA agents in a single day.

Plan Servers

A central problem in a multiagent architecture is how to maintain and communicate various alternative plans being generated to the PAs. MPA accomplishes this by having a *plan server* agent in each planning cell, although a single plan server agent can be shared by multiple planning cells. The plan server provides the central repository for plans and plan-related information, making this information accessible to all cell agents through a rich query language. The plan server is a *passive* agent in that it responds to messages sent by other agents but does not issue messages to other agents on its own initiative.

The plan server accepts incoming information from agents, performs necessary processing, and stores relevant information in its internal representation. The plan server stores representations of plans (both final and intermediate) for various tasks. To support distributed planning, the plan server must also store information about the planning process: contexts, backtracking points, declarative information about the state of the plan (e.g., a list of flaws), and so forth. The plan server is also responsible for notifying various agents of relevant planning events.

An MPA plan server supports different *views* of a plan, where a view constitutes some coherent subset of the content of a plan. For example, resource usage and graphical representations constitute two sample views of a plan. Views enable more efficient exchange of information within MPA: rather than having to retrieve an entire plan from the plan server to access certain information, agents can instead request a view with only the information that they require. Certain views are directly retrievable from the plan server, while others must be computed by performing some traversal of plans and action networks.

Annotations

Annotations are declarations of high-level attributes of tasks, plans, action networks, or the state of the planning process. Useful annotations for tasks, plans, and action-networks (called *product annotations*) include flaws or problems to be repaired, plan quality information, pedigree (how and by whom parts of the plan were derived), and comparative relationships among alternative plans and plan fragments. Annotations related to the planning process (called *process annotations*) include time spent, backtracking points, and the current state of development for a given plan (e.g., :ongoing, :completed, :failed).

Annotations are stored as predicates in the plan server's database. Annotations can be posted to the plan server by any cell agent, including the plan server itself. Product annotations are indexed by task/plan/action-network, thus allowing flexible retrieval (e.g., finding all plans or action networks that have a given annotation, or finding all annotations for a given plan, task or action network).

Triggers

A trigger is a form of event-response rule whose function is to notify specified PAs of a designated plan server event. An individual trigger is activated by the occurrence of that designated event. For now, events are limited to the insertion of annotations into the plan server. Activation of a trigger results in the dispatch of a trigger-dependent message to a designated PA. The triggered message may simply inform the receiving agents of the triggering event, or request that some action be taken.

Triggers can be supplied by various sources. Certain of them may be built into the plan server, while others may be dynamically added and removed during the planning process. For instance, the cell manager may post triggers at various times to influence the overall planning process. Individual PAs may post and remove triggers to request notification of particular events.

Plan Performative	Communication Performatives
:annotation	Insert, Delete, Ask-All, Ask-One
:trigger	Insert, Delete, Ask-All, Ask-One
:update-task	Tell, Delete
:update-plan	Tell, Delete
:query-plan	Ask-All, Ask-One
:query-node	Ask-All, Ask-One

Figure 2: Performatives for the Plan Server

Annotations and triggers can be used in tandem to provide dynamic control of the planning process. For example, suppose that temporal conflicts are arising frequently during the planning process, causing backtracking at the end of every level. The cell manager can be made aware of this situation by monitoring annotations posted to the plan server. After noticing the temporal problems, the it decides to invoke the temporal reasoning agent after every node expansion, in order to catch temporal conflicts earlier and reduce backtracking. This change in control can be easily accomplished by the cell manager inserting a trigger in the plan server that sends the appropriate message to the temporal reasoning agent whenever a Node-Expanded annotation is posted.

Communication and Plan Performatives

The combinations of communication and plan performatives in messages handled by an MPA plan server are shown in Figure 2. Ask-One is the appropriate communication performative for plan queries when only one answer is desired, while Ask-All is used when all answers (i.e., a set of answers) are desired.

Annotations and triggers are sufficiently similar to database objects that the plan server uses simple database communication performatives for manipulating them. Updating and querying a plan is more complex than adding and deleting something from a database. Therefore, a richer plan performative language is used for manipulating plans.

Act Plan Server

We have implemented a specific plan server, named the Act Plan Server, which employs the Act formalism (Wilkins & Myers 1995) for plan representation.

An Act is the basic structure used to represent an action network in the Act formalism. Acts can be expressed either in a format with embedded graphical information or in plain-text format (to facilitate translation to other languages). Graphical browsing and editing capabilities are provided by the Act-Editor system; persistence is provided through the explicit writing of Acts to files. Alternative plans can be stored for the same task. No access control or versioning is provided currently (other than the ability to store alternative plans for the same task).

The Act Plan Server supports a reasonably broad set of queries, at the level of tasks, plans, and action networks. For example, queries can be used to extract the set of known tasks, the set of plans for a given task, and the set of action networks for a given task or plan. Queries are also possible at the level of nodes within an action network (e.g., predecessor, successor, ancestor, descendant relationships). Annotations and triggers can be queried with a fairly general query language. Plan and action-network queries support a range of views including the plain-text Act representation, the graphical Act representation, the subplans associated with a given plan, the resource constraints, and the resource allocations.

The Act Plan Server is implemented as a PRS agent. PRS (Wilkins *et al.* 1995) was chosen as the implementation framework because of its ability to combine both declarative and procedural representations of knowledge, as well as to support a mixture of event- and goal-driven processing as required for the maintenance of annotations, the handling and distributing of incoming messages, and the execution of triggers.

Planning-Cell Managers

A cell manager is a persistent agent that can continuously accept tasks from other agents (human or software), decompose those tasks into subtasks to be performed by its cell agents, then recompose the results into final solutions.

A planning cell can operate as a stand-alone problem-solving unit. Additionally, sets of planning cells can be aggregated into larger cells, which are in turn controlled by a *meta planning-cell manager*. Different schemes are appropriate for different applications. We have implemented a scheme where the meta planning-cell manager accepts planning requests from human or software agents, parcels out the requests to a prespecified set of planning cells (two in our demonstration), and gathers solutions for the requesting agent, thus generating multiple plans in parallel.

Here, we describe implementations of both a baselevel cell manager (the PCM), and our meta planning-cell manager (the Meta-PCM). Their designs serve as templates for additional types of planning cell managers. In the long term, MPA will contain a library of such templates which users can adapt as appropriate for their applications.

Both the Meta-PCM and the PCM are implemented as PRS agents. PRS provides several capabilities that make it suitable for constructing such managers. Because cell managers direct the activities of multiple agents, they must be capable of smoothly interleaving responses to external requests with internal goal-driven activities. The uniform processing of goal- and event-directed behavior within PRS is ideal for supporting such behavior. PRS supports parallel processing within an agent, thus enabling multiple lines of activity to be pursued at any given time. The Act language, used to represent procedural knowledge within PRS, provides a rich set of goal modalities for encoding activity, including notions of achievement, maintenance, testing, conclusion, and waiting. Finally, the extensive textual and graphical tracing in PRS provides valuable runtime insights into the operation of cell managers. A user can interact directly with either the PCM or Meta-PCM through the PRS

Plan Performative	Comm. Performative
:annotation :advice	Tell
:solve	Evaluate

Figure 3: Performatives for the PCM

PCD Role	Agent Names
:manager	PCM
:plan-server	Act-Plan-Server
:planner :search-manager :critic-manager	SIPE-2
:scheduler	OPIS
:temporal-reasoner	OPIS, Tachyon
:requestor	Meta-PCM, User

Figure 4: Plan Cell Descriptor Roles and Possible Fillers

interface by posting appropriate goals, and adding or retracting information from the agent's database.

Overview of the PCM

Planning requests to a single planning cell are serviced sequentially rather than concurrently; thus, the PCM can solve multiple tasks but only one at a time. If the PCM receives a request to generate a plan while servicing an earlier request, it returns a message indicating that it is busy.

The combinations of communication and plan performatives currently handled by the PCM are shown in Figure 3. Annotation messages advise the PCM of annotations that have been posted in the plan server. Such messages are sent by triggers posted by the PCM itself. Solve messages request the PCM to generate a plan. Advice messages provide problem-solving advice (Myers 1996) to be used by plan generation agents within the cell; the PCM will pass along specified advice to those agents when making planning requests.

The roles in a PCM planning cell and their possible agent fillers are listed in Figure 4. Each role is filled by zero, one, or a set of agents, depending on the nature of the problem-solving process. Within the current PCM, the :scheduler and :temporal-reasoner roles are optional.

The PCM supports a small number of planning styles, all of which assume a level-by-level plan generation model, derived from the hierarchical task network (HTN) approach to planning. For each level, a more refined plan is first generated, then critiqued. If critic roles are left unfilled, critiquing is skipped. This model, while not applicable to all planners, is reasonably general. In particular, nothing is assumed about the definition of a level, thus enabling a range of level-refinement methods (for example, expansion of a single goal, or all goals). In addition to HTN planners, causal-link planners fit naturally into this scheme, with goal selection, operator selection, and subgoal generation viewed as forms of plan expansion, and causal link protection and checking constraint consistency as forms of plan critiquing.

The PCM planning styles vary in their choice of agent to perform the plan refinement, the selection of critic agents for the critique phase, and the frequency of critic invocation. The PCM takes different actions based on the information returned by the planner about the status of the expansion process. The following values can be returned:

:final-plan The returned action network is a complete and validated solution. Receipt of such a response terminates the PCM planning operations for the current task.

:plan-complete The returned action network is a completed plan that needs to be verified by the critics.

:level-complete The returned action network is a successful refinement of the previous level's action network. The PCM continues with this new level.

:no-expansion No expansion was found for this level. The PCM initiates backtracking by setting expansion parameters and then reinvoking the planner.

:plan-failure No plan was found and no backtracking is possible. The PCM abandons the task.

Plan Performative	Comm. Performative
:solution :failed	Tell
:multiple-solve :new-agent	Evaluate

Figure 5: Performatives for the Meta-PCM

Overview of the Meta-PCM

The Meta-PCM accepts planning request messages from humans or other agents, each of which can request multiple solutions to a given task. For each request, the Meta-PCM locates an appropriate number of available planning cells and sends messages to the manager of each cell requesting a solution. The messages include the task to be solved, and advice for how to solve the task. The Meta-PCM may distribute only a subset of the received requests when planning cells are busy.

The Meta-PCM responds to a variety of messages from planning cells that describe a solution or a failure for a planning request. Results are then forwarded to the requestor of the planning task by sending a message. The Meta-PCM also notices when all requests from a given incoming message have been serviced, and produces a summary.

The combinations of communication and plan performatives in the messages handled by the Meta-PCM are shown in Figure 5. Multiple-solve messages are the incoming planning requests. Solution and Failed messages report results from planning cells. A New-agent message declares the availability of a new planning cell.

Integrating Planning and Scheduling

The previous sections described agents that were created specifically for MPA: the Meta-PCM, the PCM, and the Act Plan Server. Here we describe the modularization of legacy software systems into MPA agents. In particular, we describe the use of an existing planner (SIPE–2 (Wilkins *et al.* 1995)) and an existing scheduler (OPIS (Smith, Lassila, & Becker 1996)) within MPA. (The integration of planning and scheduling within MPA has been a collaboration with Dr. Steve Smith of Carnegie Mellon University.)

One shortcoming of the planner in our domain is its inability to perform a capacity analysis early in the planning process. For example, when there are 75 airplanes and the plan requires 83, the "capacity" of airplanes is inadequate. Part of the MPA decomposition of OPIS includes an MPA agent for capacity analysis.

Planner Agents

The specification for the planner agent assumes a level-by-level plan generation process with a critique of the plan after each expansion, as described above. The planner can expand the plan at each level to some arbitrary extent, as defined by its algorithms. The critique can post annotations to the plan server, resulting in failure for the planning process if unresolvable conflicts are found. Plan expansion is accomplished by the Search Manager agent, while the critique is accomplished by the Critic Manager agent.

The Search Manager handles a range of role-specific messages. The primary plan performatives for Evaluate messages are :expand-plan and :expand-plan-and-critique. The former causes the plan to be expanded to the next level, while the latter additionally calls the declared cell critics on the plan. Other Evaluate plan performatives include requests for initializing a problem (by translating the problem into the planner's internal representation), displaying a plan, and resetting various context information (to recover from aborted planning processes). The Search Manager also accepts a Tell message with the Advice plan performative, which can be used to provide advice to guide the planner in its search for solutions.

The role-specific messages for the Critic Manager all employ the communication performative Evaluate. Several plan performatives are supported, corresponding to different kinds of critiques to be performed. Plan-ok? causes the planner to apply all critics known to it; Schedule-ok? causes the agent to invoke the scheduler agent of the planning cell; Temporal-ok? causes the agent to invoke the temporal-reasoner agent of the planning cell.

The SIPE–2 planning system (Wilkins *et al.* 1995) and the Advisable Planner (Myers 1996) have been used as the basis for planner agents within all MPA applications to date. SIPE–2 has a precise notion of a planning level, and plan critics that fit naturally into the above scheme. To serve as an MPA planner agent, it had to be modularized, separating out its search control algorithm into the Search Manager agent, its plan critics algorithm into the Critic Manager agent, and its temporal reasoning critic into the Temporal Critic agent, which was extended to use any temporal reasoner in the planning cell. In addition, a new critic was created using the scheduler agent to do capacity analysis and resource allocation. The Schedule Critic agent was written to interact with whatever scheduler agent is in the planning cell.

Scheduler Agent

The use of the scheduler in our current implementation is tied closely to the ACP domain. This domain has been modeled within OPIS to produce a scheduling agent for the planning cell. This agent currently provides two types of service to support the planning process:

- Capacity analysis - profiles the resource demand over time, and identifies periods where available assets are oversubscribed.
- Resource allocation - commits assets to specific activities, precluding their use on other activities.

The main invocation of the scheduler is through an :allocate-resources plan performative (with an Evaluate communication performative), which causes the scheduler to analyze capacity and allocate resources. This process involves the scheduler retrieving the :resource-constraints view of the current plan from the plan server, performing its analysis, and posting annotations in the plan server about capacity overruns and resource allocations for use by other agents.

MPA Configurations

We use the term *configuration* to refer to a particular organization of MPA agents and problem-solving strategies. Here, we describe two MPA configurations: a single-cell configuration for generating individual solutions to a planning task, and a multiple-cell configuration for generating alternative solutions in parallel. The use of these configurations for performing planning/scheduling in an Air Campaign Planning domain is also described.

Single-Cell Configuration

The single-cell configuration, illustrated in Figure 6, includes multiple planner and scheduler agents, together with a temporal reasoning agent, in addition to the Plan Server and PCM agents. The temporal reasoner role can be filled by either an OPIS-based agent or a Tachyon-based agent (Arthur & Stillman 1992). The Tachyon agent is written in C; all other agents are written in LISP. The agents run on different machines, both locally and over the Internet. Within this configuration, the agents cooperatively generate a plan with the cell manager dynamically reconfiguring the planning cell during planning.

All agents send messages to and from the plan server. The plan server supports annotations and triggers that are used to record features of the plan and notify agents of the posting of those features. The plan is written to the plan server in the Act formalism, which can be understood by the scheduler and the planner. The Search Manager is based

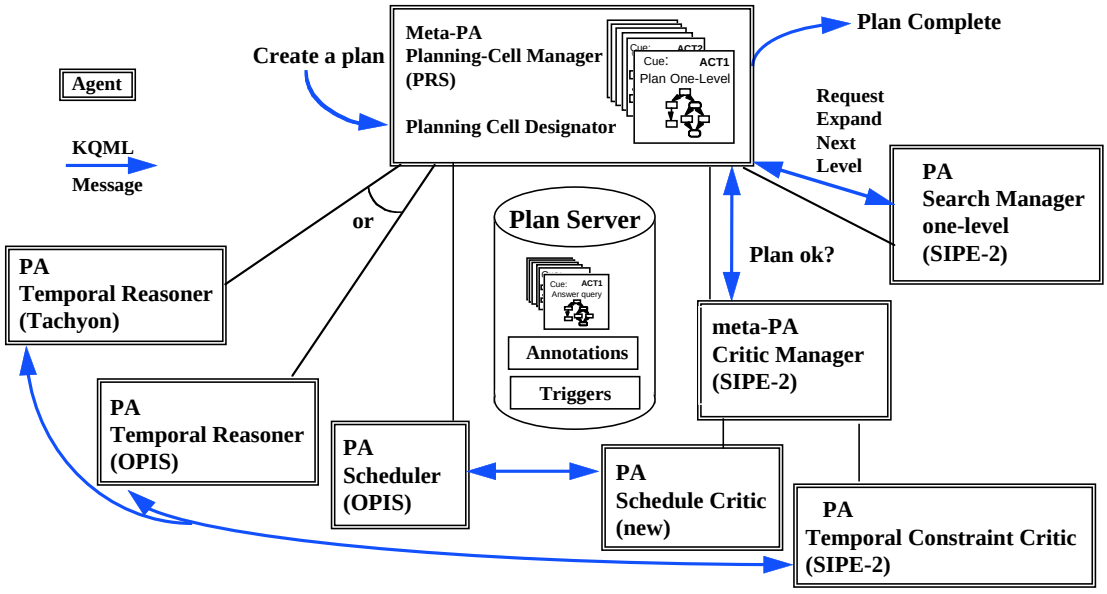

Figure 6: MPA Single Cell Configuration. Arrows represent message flow; because all agents communicate with the plan server, those arrows are omitted. Lines without arrowheads show planning cell composition.

on SIPE–2's interactive search routine, which has been extended and modified to record its backtracking points and other information in the plan server. Another extension permits starting the search at various backtracking points.

The configuration generates a two-day plan to achieve air superiority over two specified countries. The PCM generates a plan to the target level, using a planning cell including Tachyon as the temporal reasoner but no scheduler. During planning of support missions, the PCM reconfigures the planning cell to include the scheduler and to exclude the temporal reasoner (the temporal constraints are not impacted by the support missions).

The scheduler is then called periodically to check the resource allocations. Depending on the PCM planning style, the period can be once per node, once per level, or once per plan. The Scheduler can recommend new resource assignments, which causes the Schedule Critic to modify the plan. OPIS posts annotations declaring which resources are overutilized or near capacity. If resource constraints are sufficiently unsatisfiable, OPIS reports a schedule failure.

The configuration develops a plan in which fuel tankers are overutilized. The PCM has posted a trigger on such an annotation and is immediately notified. It responds by sending an :advice plan performative to the planner, which causes the planner to choose options requiring less fuel for the remainder of the plan expansion. The plan will still have flaws because resources were already overutilized before the PCM issued the advice. Therefore, the PCM also invokes a second search for another plan, this time using fuel advice from the start. This produces a fuel-economic plan in which tankers are not overutilized.

Multiple-Cell Configuration

The multiple-cell configuration (see Figure 7) includes multiple instances of the single-cell configuration coordinated by the Meta-PCM. The planning cells share common plan server, temporal reasoner, and scheduler agents, to reduce the number of running jobs, but multiple instances of the shared agents can also be chosen.

The Meta-PCM controls the entire process, including initialization of planning cells, distribution of tasks and advice, and reporting of solutions. The planning cells operate exactly as described for the single planning-cell configuration, except that they are invoked by the Meta-PCM instead of the user, and they refuse requests if they are already busy.

The multiple planning-cell configuration can be used to produce alternative plans for the same task in parallel. Different advice can be provided with each plan generation request, thus resulting in plans that differ in significant characteristics. As such, the multiple-cell configuration provides the means to rapidly explore varying portions of the overall set of candidate plans.

Additional extensions include the integration of agents for plan evaluation, user interaction, and plan visualization. The ARPI Plan Authoring Tool (APAT) from ISX, a legacy system written in Java, fills the role of user interface and advice manager (depicted in Figure 6) and plan visualization (a service also provided by the VISAGE system from MAYA). The Air Campaign Simulator (ACS) (Cohen, Anderson, & Westbrook 1996) from the University of Massachusetts, written in LISP, provides Monte Carlo simulations of plans. The VISAGE system provides plan visualization for simulation outputs. Both of these agents read

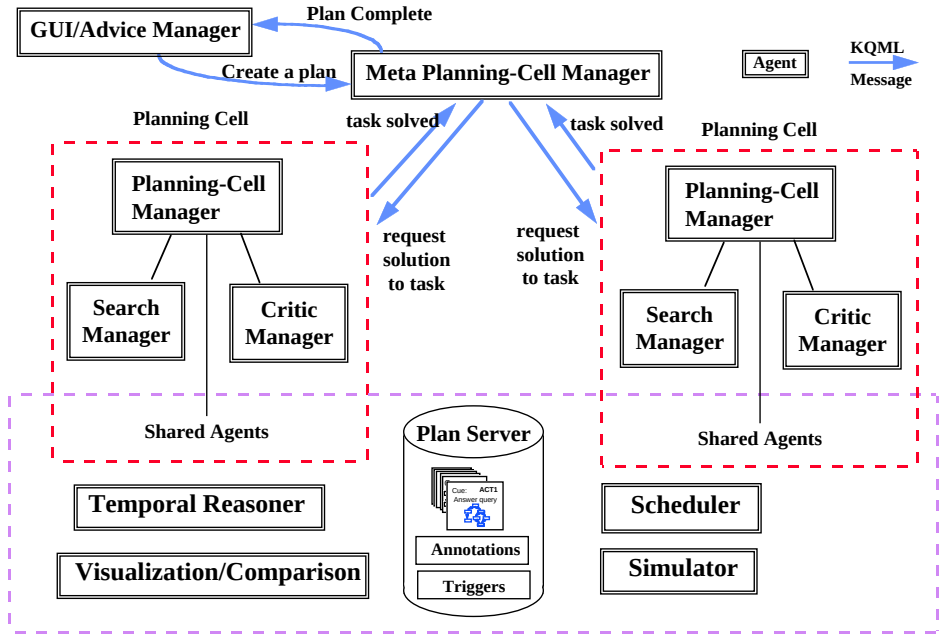

Figure 7: MPA Configuration for Multiple Planning Cells.

Acts from the Act Plan Server and translate them to their internal representations.

Evaluation

One critical evaluation issue for MPA is determining whether the overhead derived from its agent-based organization and interagent message-passing outweighs the benefits of the framework. To measure the overhead, we timed the generation of ACP plans for two cases. The first case involved using a planner (SIPE–2) in isolation; the second case employed an MPA planning cell with a cell manager, a plan server, and a planner, each running on a separate machine. Sun Ultras were used in all cases.[2]

The results are shown in Figure 8. The two columns display timings for two ways of communicating the plan to the plan server. Times under the File column refer to writing the plan to a file and (in the MPA case) sending the name of the file to the plan server. Times under the Direct column refer to translating the plan to plain-text Acts and (in the MPA case) sending these Acts to the plan server. File communication has the advantage of limiting the size of interagent messages, but works only when the plan server and planner share a common file system. For Plan 1, the PCM sends about 50 messages (mostly to the planner) and the plan server receives about 60 messages (mostly from the planner).

Using file communication, MPA has minimal overhead for both plans. This overhead derives primarily from the search control process interpretively executing Acts in the PCM, as opposed to using compiled code in the planner. Using direct communication, however, planning time increases by two-thirds for Plan 1 and quadruples for Plan 2 (where over half a megabyte of data is sent). MPA has not been optimized for direct communication of large amounts of data (known improvements exist).

A primary goal of MPA was to make it easy to integrate new technologies. As evidence of success, researchers from the University of Massachusetts required only one day to instantiate ACS as a functional MPA agent capable of sending and receiving messages. (A few more days were required to build a translator from Act to the internal ACS representation for plans). Our integration experience prior to MPA

	File	Direct
Plan 1 (4282 nodes, 363K)		
Planner Only	3.1	3.6
MPA	3.3	6.0
Plan 2 (6437 nodes, 554K)		
Planner Only	8.5	9.0
MPA	8.8	38.4

Figure 8: Elapsed Times for Plan Generation (minutes). The number of nodes listed for each plan encompasses action/goal and control nodes for all 15 levels of each plan. Following the number of nodes is the total amount of disk space required for storing textual representations of all levels of each plan.

[2] All product and company names mentioned in this document are the trademarks of their respective holders.

was that such an effort would have taken several weeks to a few months, as specific interfaces would be designed to bridge differences in representation, programming language, and communication mechanisms.

Limitations and Future Work

Generality was a key objective in designing the architecture and communication protocols for MPA. However, the design was necessarily influenced by the specifics of the technologies and applications that were considered. We intend to continue to expand the breadth and generality of the system, through consideration of both additional technologies for the categories of agents already in the system, as well as new categories (such as execution agents).

Additional directions for extending this work are numerous. SRI has begun work on mechanisms for coordinating subplans generated by distributed planning agents (Wolverton & desJardins 1998). Other directions include experimenting with additional configurations and cooperative problem-solving methods, automatically composing planning cells based on capabilities models, defining a broader range of cell manager control strategies and planning styles, and extending the plan server's shared plan representation.

Summary

MPA is an open planning architecture that facilitates incorporation of new technologies and allows the planning system to capitalize on the benefits of distributed computing for efficiency and robustness. MPA provides protocols to support the sharing of knowledge and capabilities among agents involved in cooperative problem-solving. These protocols support generation of multiple plans in parallel.

There is a tradeoff between overhead from MPA and the flexibility it provides. Our evaluation shows the overhead is insignificant in many cases. Our experience indicates that MPA does facilitate the integration of new technologies, thus encouraging experimentation with new technologies, as well as the use of smaller, more modular software components. MPA also facilitates the integration of legacy systems, including systems written in different programming languages. Little tuning of legacy systems is required.

These advantages are shown by our use of MPA to integrate several sophisticated stand-alone systems cooperating on a large-scale problem. Separate software systems (OPIS, Tachyon, ACS, APAT, the Advisable Planner, and SIPE–2, using KQML, the Act-Editor, and PRS for support) cooperatively generate and evaluate plans, generating multiple, alternative plans in parallel. These systems have been combined in multiple ways through the flexible architecture. The integrated problem-solving framework generated and evaluated complex plans (containing more than 4000 nodes) in the ACP domain, and included agents written in C, C++, LISP, and Java.

The Act Plan Server allows flexible communication of the plan among agents through the use of annotations, triggers, and views. The PCM encodes different strategies for controlling the planning process, demonstrating dynamic strategy adaptation in response to partial results. The planner and scheduler use legacy systems to provide a new integration of planning and scheduling, including the use of advice as a means for the scheduler to influence the planner.

Acknowledgments This research was supported by Contract F30602-95-C-0235 with the Defense Advanced Research Projects Agency, under the supervision of Air Force Research Lab – Rome. Pauline Berry, Marie desJardins, and Tom Lee of SRI and Steve Smith and Marcel Becker of Carnegie Mellon University made significant contributions.

References

Arthur, R., and Stillman, J. 1992. Tachyon: A model and environment for temporal reasoning. Technical report, GE Corporate Research and Development Center.

Cohen, P.; Anderson, S.; and Westbrook, D. 1996. Simulation for ARPI and the Air Campaign Simulator. In Tate, A., ed., *Advanced Planning Technology: Technological Achievements of the ARPA/Rome Laboratory Planning Initiative*, 113–118.

Finin, T.; Weber, J.; Wiederhold, G.; Genesereth, M.; Fritzson, R.; McKay, D.; and McGuire, J. 1992. Specification of the KQML Agent-Communication Language. Technical Report EIT T R92-04, Enterprise Integration Technologies, Palo Alto, CA.

Georgeff, M. P. 1984. A theory of action for multiagent planning. In *Proceedings of the 1984 National Conference on Artificial Intelligence*, 121–125.

Janssen, B.; Spreitzer, M.; Larner, D.; and Jacobi, C. 1997. ILU 2.0 reference manual. Technical report, Xerox PARC.

Moran, D. B.; Cheyer, A. J.; Julia, L. E.; Martin, D. L.; and Park, S. 1997. Multimodal user interfaces in the Open Agent Architecture. In *Proc. of the 1997 International Conference on Intelligent User Interfaces (IUI97)*.

Myers, K. L. 1996. Strategic advice for hierarchical planners. In Aiello, L. C.; Doyle, J.; and Shapiro, S. C., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR '96)*. Morgan Kaufmann Publishers.

Smith, S. F.; Lassila, O.; and Becker, M. 1996. Configurable, mixed-initiative systems for planning and scheduling. In Tate, A., ed., *Advanced Planning Technology*. Menlo Park, CA: AAAI Press.

Wilkins, D. E., and Myers, K. L. 1995. A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation* 5(6):731–761.

Wilkins, D. E.; Myers, K. L.; Lowrance, J. D.; and Wesley, L. P. 1995. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI* 7(1):197–227.

Wolverton, M. J., and desJardins, M. 1998. Controlling communication in distributed planning using irrelevance reasoning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*. AAAI Press.