A Decision-Theoretic Planner with Dynamic Component Reconfiguration for Distributed Real-Time Applications

AAAI 2006 Paper ID: 955

Abstract

Distributed real-time embedded (DRE) systems often must perform sequences of coordination and heterogeneous data manipulation tasks to meet specified goals. Autonomous operation of DRE systems in dynamic and uncertain environments can benefit from the integrated operation of (1) a *Spreading Activation Partial Order Planner* that combines task planning and scheduling in uncertain environments with (2) a *Resource Allocation and Control Engine* middleware framework that integrates multiple resource management algorithms for (re)deploying and (re)configuring task sequence components in DRE systems. We demonstrate the effectiveness of the decision-theoretic mission planner and the middleware framework in the context of managing and executing mission goals for a multi-satellite system. Our results show that coupling a dynamic planner that takes into account scheduling and resource constraints using our allocation and management middleware framework enables the efficient implementation of autonomy in DRE systems.

Introduction

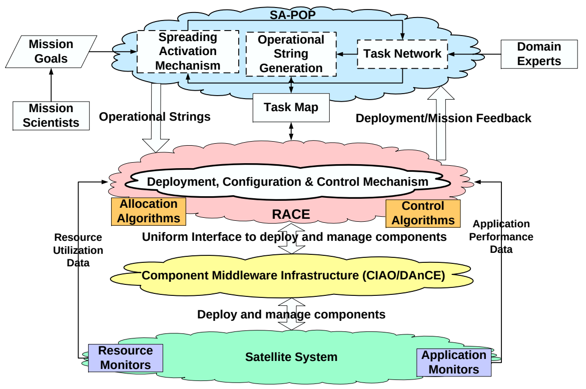
Distributed real-time embedded (DRE) systems, such as multi-satellite and multi-robot formations, perform sequences of heterogeneous data collection/manipulation and coordination tasks to meet specified goals. For example, weather prediction requires multiple satellites that fly coordinated missions with multiple sensors to collect and analyze large quantities of atmospheric and earth surface data. The data collection, analysis, and earth transmission task sequences are governed by dynamic factors, such as data analysis results, changing goals and priorities as weather conditions change, and uncertainties due to environmental conditions and changing resource availability.

Presently task sequence implementations in DRE systems use *component middleware* (Heineman & Councill 2001), which automates remoting, lifecycle management, system resource management, deployment, and configuration. In large-scale DRE systems, the sheer number of component sequences often poses a combinatorial deployment problem, *i.e.*, mapping components to computing nodes. Moreover, the dynamic nature of the operations requires runtime management and modification of deployments. Effective solutions to this problem must also include planning and replanning capabilities to ensure the task sequences being executed match current mission goals and resource availability.

For example, the NASA Earth Science Enterprise's Magnetospheric Multi-Scale (MMS) mission uses five satellites with six sensors on each satellite as a solar-terrestrial probe. The satellites orbit the earth in formation and collect electromagnetic and particle data in the earth's magnetosphere. The mission operates in three data modes: slow, fast, and burst. Each mode includes different goals, orbits, and data priorities.

An automated planner for mission task sequences must handle changing goal prescriptions specified by mission scientists and autonomous mode changes driven by satellite positions and the results of analyzing collected data. The task sequences include components for coordinating the trajectory and orientation of satellites, sensor selection and data collection for individual satellites, and data integration and compression to create telemetry streams beamed down to the earth stations. Changing resource usage and environmental conditions may affect the likelihood that software components will successfully complete a task. The runtime computational architecture must include dynamic planning and replanning, as well as dynamic monitoring and reallocation of resources, to accommodate changing goals and operating conditions.

To support such DRE systems, we developed a novel computationally efficient algorithm called the *Spreading Activation Partial Order Planner* (SA-POP) for dynamic (re)planning under uncertainty. Prior research (Srivastava & Kambhampati 1999) identified scaling limitations in earlier AI approaches that combine planning and resource allocation/scheduling in one computational algorithm. We therefore combined SA-POP with a *Resource Allocation and Control Engine* (RACE), which is a reusable component middleware framework that separates resource allocation and control *algorithms* from the underlying middleware deployment, configuration, and control *mechanisms* to enforce quality of service (QoS) requirements.



SA-POP
Mission Goals
Mission Scientists
Spreading Activation Mechanism
Operational String Generation
Task Network
Domain Experts
Task Map
Operational Strings
Deployment/Mission Feedback
Deployment, Configuration & Control Mechanism
Allocation Algorithms
RACE
Control Algorithms
Resource Utilization Data
Application Performance Data
Uniform Interface to deploy and manage components
Component Middleware Infrastructure (CIAO/DAnCE)
Deploy and manage components
Resource Monitors
Satellite System
Application Monitors

Figure 1: SA-POP and RACE in a Multi-Satellite System

DRE System Architecture

Figure 1 shows the computational architecture of a DRE system with two subsystems: (1) the SA-POP planner that generates *operational strings* given application goals and (2) the RACE framework that monitors and manages runtime resource allocation to enforce QoS requirements. DRE systems for shipboard computing (Schmidt *et al.* 2001), avionics mission computing (Sharp & Roll 2003), and intelligence, surveillance and reconnaissance (Sharma *et al.* 2004) view applications as groups of domain-related tasks that can be implemented by parameterized and executable software components using component technologies, such as the OMG's Lightweight CORBA Component Model (CCM) and Web Services. In our architecture, *components* are units of implementation and composition that contain parameterized executable code with specified QoS requirements (such as maximum latency and minimum throughput values) and resource consumption profiles (such as expected CPU and memory usage).

To devise an application that achieves a given set of *goals*, *e.g.*, study the physics of plasma reconnection and charged particle acceleration for the MMS mission, the SA-POP planner shown in Figure 1 first generates partial order *task sequences* that achieve specified goals using a spreading activation mechanism (Bagchi, Biswas, & Kawamura 2000). Individual tasks in the sequences are then mapped to available executable software components, *e.g.*, the planner may pick a data compression task and then select an appropriate component implementation for a chosen compression algorithm. The planner must know which preconditions to satisify for a task component to execute successfully, the input data stream and the output that will be generated from this data stream, and other postcondition effects resulting from their operation.

The component ouput is a function of the input and environmental conditions during actual operation. Other computational properties of the component, *e.g.*, the throughput and the quality of the output, depend on the available computational resources. As a result, there is uncertainty whether the component will produce the desired output. This uncertainty is captured by conditional probabilities associated with the component definitions. Together, the task-component relations and the conditional probability of success of components defines the *functional signature* of the task. Different parameterizations of a given component may produce different functional signatures. Conversely, different components that have the same functional signature may vary in time to completion, resource usage, and QoS parameters.

We define a *task* as one or more parameterized components with a single functional signature. The functional signature of each task is captured in a *task network*, which is a directed graph that represents both tasks and conditions (preconditions, data input, effects, and data output) with links encoding the requisite probability information. With the task network and a given set of utility values for goal conditions/data, the planner computes expected utility values for each task using the spreading activation mechanism.

To ensure applications do not violate resource constraints, the planner also requires knowledge of each task's resource consumption and execution time, *i.e.*, its *resource signature*. A given task may be associated with multiple parameterized components, each with different resource signatures. SA-POP and RACE therefore use a shared *task map* that maps each task to a set of parameterized components and their associated resource signatures. The combination of functional and resource signatures in a task sequence defines an *operational string*, which specifies the tasks, a suggested implementation for each task, the control (ordering) dependencies, the data (producer/consumer) dependencies, and required start and end times for tasks, if any.

Operational strings are given as input to RACE, which provides reusable algorithms for (re)deploying components onto nodes and managing application performance, as well as utilization of system resources. RACE allocates resources to application components based on their resource requirements and QoS characteristics and monitors application and infrastructure performance and resource usage. Since component resource use and end-to-end QoS for operational strings are sensitive to runtime changes and changes in system performance, *e.g.*, due to loss of resources and transient overload, RACE can also redeploy and/or reconfigure application components using the implementation options available in the task map to ensure the desired end-to-end QoS requirements of operational strings are not violated.

The Spreading Activation Planner

This section describes the primary algorithmic steps in the spreading activation planner (SA-POP), which include (1) a decision-theoretic spreading activation mechanism to identify task sequences that maximize an expected utility measure given a set of goals and (2) an operational string generation mechanism that uses the computed expected utilities of tasks and their associated implementation resource signatures to ensure that the extracted task sequences in the operational string have high expected utility and meet resource, time, and other QoS constraints.

Spreading Activation Networks

The spreading activation task network captures the links between task sequences and goal conditions (Bagchi, Biswas, & Kawamura 2000). An example network from the MMS mission scenario is shown in Figure 2, which consists of condition nodes (ovals) and task nodes (rectangles) with directed links that indicate the pre- and post-conditions for executing individual tasks. Condition nodes are represented as Boolean variables with associated probabilities that define the maximum likelihood of that node achieving true/false values. Environmental/system conditions (e.g., a particular sensor is active) and generated data (e.g., a data stream from a sensor) are represented as condition nodes. The data condition nodes represent the availability (true) or non-availability (false) of the corresponding data.

The weight, w_{ij}, of the link from a condition node, c_i, to a task node, t_j, defines the likelihood that t_j succeeds in given c_i, *i.e.*

$$w_{ij} = \frac{P(t_j^s|c_i = true) - P(t_j^s|c_i = false)}{P(t_j^s|c_i = true) + P(t_j^s|c_i = false)}, \qquad (1)$$

where t_j^s indicates that task t_j is successful. This encoding allows for *hard constraints* (weight $= 1$ (-1)), *i.e.,* the condition must be true (false) for the task to succeed, and *soft constraints* (weight < 1 (> -1)) *i.e.,* the true (false) value of the condition increases the probability of task success. Soft constraints model inferred conditions in uncertain environments, where an actual precondition can not be sensed directly but is probabilistically related to other conditions that can be sensed. For example, an imperfect (noisy) sensor for detecting an environmental condition necessary to the success of a task can be modeled using a soft constraint.

The weight, w_{jk}, of the link from a task node, t_j, to a condition node, c_k, defines the probability that c_k will be true/false after t_j executes, *i.e.*:

$$w_{jk} = \begin{cases} P(c_k = true|t_j^x) & \text{if } t_j \text{ sets } c_k = true \\ -P(c_k = false|t_j^x) & \text{if } t_j \text{ sets } c_k = false, \end{cases} \qquad (2)$$

where t_j^x indicates that task t_j is executed.

The likely contribution of a task toward a desired goal is computed as an expected utility (EU), *i.e.,* the product of the task's utility toward meeting the goal requirements and its likelihood of success. Probability values are propagated forward through the network from preconditions through tasks to effects. Utility values are propagated backward through the network from effects through tasks to preconditions, which allows preconditions of potentially useful tasks to accumulate utility, making them useful subgoals toward meeting the specified goal requirements.

Forward propagation of probabilities assumes that the preconditions of a task are independent conditioned on the success of the action. Even when the preconditions of a task are not truly independent, this is a reasonable approximation, which prevents the probability calculations from becoming intractable (Bagchi, Biswas, & Kawamura 2000).

Forward propagation is illustrated for the network in Figure 2. Assume that initially sensor 1 is active on all three satellites (solid, dashed, and dotted outlined nodes represent tasks and conditions for satellites 1, 2, and 3, respectively), and all other conditions are false. The links from "Sensor 1 Active" to the "Focus Sensor Data" tasks all have weight 1, so the probability of success for this node is computed to be 1.0.

Continuing forward propagation from the "Focus Sensor Data" task node to the "Reliable Sensor1 Data" condition node, the link weights are less than 1, *i.e.,* the execution of either Focus task does not ensure that this condition will be satisfied. The spreading activation mechanism assumes that if a condition becomes a subgoal in a plan, then the action most likely to accomplish it will be selected. The probability propagated forward is computed as the maximum of the two probabilities from the Focus tasks, *i.e.*, $1.0 * 0.9 = 0.9$.

The "Sensor1 Data Analyzed" and "Sensor1 Data Transmitted" conditions are goals selected by the mission scientists, who assign these goals utility values that imply their relative importance to the overall mission. The goal utilities are backpropagated through the task network links to the "Reliable Sensor1 Data" condition, which accumulates utility from both goals (by summation), increasing its importance as a subgoal that will contribute to the overall mission success. Computed utilities are further propagated backward taking into account the likelihood of being set to the desired value, *i.e.*, the product of the condition's utility and link weight. Backward propagated utilities multiplied with the forward propagated probability of success at each task node determines the EU for the node. For the example, the EU of "Focus Multi Sensor1 Data" is greater than that of "Focus Sensor1 Data" because it is more likely to set the "Reliable Sensor1 Data" condition to true, and both have equal forward propagated probabilities of successful execution.

Operational String Generation

For MMS and similar DRE systems, the fewer constraints imposed by the operational string, the easier it is for RACE to make initial deployment decision and manage resources at runtime. To ensure this, we adopt a modified *Partial Order Causal Link* (POCL) design (Smith, Frank, & Jonsson 2000) to generate operational strings. The least commitment strategies typical of partial order planning allow SA-POP to impose

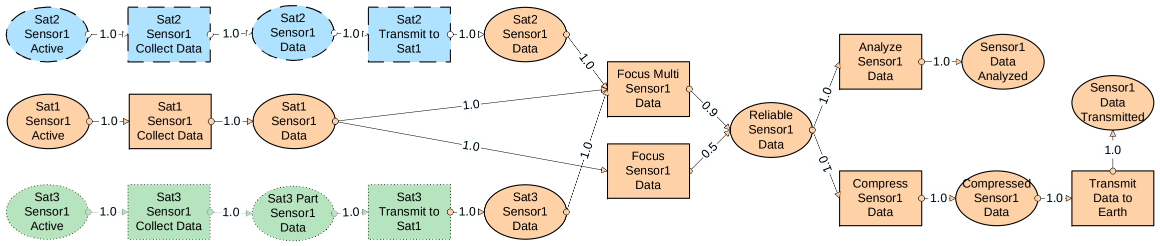


Figure 2: A Spreading Activation Network for the MMS Scenario

relatively few constraints compared to other popular planning techniques, such as state space search and constraint satisfaction problem based planners. Recent research (Nguyen & Kambhampati 2001) also indicates that in many cases the performance of partial order planning can be brought up to par with these other approaches.

SA-POP uses four hierarchical decision points with backtracking in the generation of operational strings. Each step in the generation of an operational string from the task network with assigned probability and EU values involves the following layered decision points: (1) *Goal/subgoal choice*: choose an *open condition*, which is goal or subgoal unsatisfied in the current plan, (2) *Task choice*: choose a task that can achieve the open condition from 1, removing the open condition and adding the task's unsatisfied preconditions to the set of open conditions, (3) *Task instantiation*: choose an implementation (parameterized component) for this task from the task map, and (4) *Scheduling decision(s)*: adjust task start/end time windows and/or add ordering constraints between pairs of tasks to avoid resource violations. We describe each of these decision points below.

Goal/Subgoal Choice. SA-POP begins with the mission goals as the set of open conditions. Since data manipulation tasks are resource intensive and tend to be concuurent with other data tasks in DRE domains, SA-POP gives priority to data operation nodes. This heuristic also enables early detection of resource violations in operational strings.

Task Choice. Task choice is based on EU (tasks with higher EU's are preferred) provided their likelihood of success exceeds a pre-defined threshold. This represents a tradeoff between the total expected utility, which may accumulate from multiple goals, and the likelihood of achieving the subgoal currently under consideration. In the creation of an operational string for the example network from Figure 2, the "Reliable Sensor1 Data" condition node is a subgoal needed to achieve both mission goals, so at some point it will be an open condition that needs to be mapped onto a task execution. Since the "Focus Multi Sensor1 Data" task has both a greater expected utility and greater probability of achieving this condition than "Focus Sensor1 Data", it is added to the operational string. This corresponds to traditional partial order planning with preference for tasks with high expected utility.

Task Instantiation. This step moves from pure plan generation to task selection that meets stated resource requirements. SA-POP first determines the change in potential resource usage for possible components (from the task map), given current task orderings. The percentage decreases in available resource capacities are summed to provide a resource impact score, and the component with the lowest score is chosen to implement the task. This heuristic is comparable to the least constraining value heuristic often used in general constraint satisfaction problems. For example, there may be multiple compression components that are associated with the "Compress Sensor1 Data" task from Figure 2, each with a different tradeoff between memory and CPU usage requirements. If other tasks in the operational string that operate concurrently (e.g. tasks connected through data nodes to this task and tasks on entirely different paths in the task network) are causing CPU utilization to reach its upper limit, but this is not the case with memory use, SA-POP chooses the component that uses more memory but has a low CPU usage profile.

Scheduling Decision(s). In tracking resource constraints and finding resource violations, SA-POP employs the ordering constraints between tasks. In DRE systems, such as the MMS scenario, a significant number of the tasks in an application may be data manipulation tasks. Often, these data handling tasks operate over long time windows with a required start time, but no defined end time. Rather the end time is dynamically determined by ongoing analysis of the data. This limits the effectiveness of many popular scheduling approaches such as timetabling (Pape 1994), edge-finding (Baptiste & Pape 1996), and classical energetic reasoning (Laborie 2003). Instead of primarily relying on start/end time window manipulation, as in those approaches, SA-POP leverages the ordering constraints common to partial order plans. These constraints are used to create precedence graphs (Laborie 2003) that partition all other tasks into sets based on their ordering with respect to a particular task under consideration. With this information, SA-POP applies Laborie's energy precedence constraint and balancing constraint techniques (Laborie 2003) to detect potential resource violations and add other ordering constraints or decrease start/end time windows.

When an unresolvable resource violation is detected during the scheduling step, backtracking is employed. In the Figure 2 example, the choice of "Focus Multi Sensor1 Data" instead of "Focus Sensor1 Data" may yield operational strings for which there are no possible implementation and scheduling choices that meet resource constraints. Backtracking will return to the decision point and choose the lower EU task, "Focus Sensor1 Data" over "Focus Multi Sensor1 Data." This demonstrates how SA-POP trades off EU versus feasibilty of execution based on meeting resource constraints.

Resource Allocation and Control Engine

The architecture of RACE and its interplay with SA-POP is illustrated in Figure 1. RACE is built atop of CIAO and DAnCE, which are open-source implementations of the OMG Lightweight CCM (Obj 2003b), Deployment and Configuration (D&C) (Obj 2003a), and Real-time CORBA (Object Management Group 2002) specifications. RACE provides a range of resource allocation and control algorithms that use middleware deployment and configuration mechanisms to allocate resources to operational strings and control system performance after operational strings have been deployed. RACE uses *Resource Monitors* and *ApplicationQoS-Monitors*, which are implemented as CCM components, to track system resource utilization and application QoS respectively.

RACE's algorithms determine how to (re)deploy an application specified by operational strings and ensure desired QoS requirements are met, while maintaining resource utilization within desired bounds at all times. The allocation algorithms determine the initial component deployment by determining the best mapping of these components to the appropriate target nodes based on the availability of system resources. Likewise, RACE's control algorithms adapt the execution of an operational strings' components at runtime in response to changing environments and variations in resource availability and/or demand.

Discussion and Conclusions

This section demonstrates the power of combining the decision-theoretic, resource-constrained planning of SA-POP with the component allocation and runtime management of RACE to produce an efficient and scalable architecture for DRE systems operating in dynamic and uncertain domains. SA-POP produces partial-order plans that contain sufficient information to be instantiated with parameterized component implementations that do not violate coarse-grained resource constraints. For example, in the MMS system, SA-POP considers the computational resources for each satellite, such as CPU, memory, and communication bandwidth to be monolithic, discrete resources. In actuality, there are multiple nodes with individual CPU and memory capacities within each satellite. In general, each task only uses a small fraction of these resources, so the course-grained resource constraints used by SA-POP helps ensure that RACE can find a valid deployments for components on the real node resources at runtime.

Through the association of multiple functionally equivalent implementations for each task in the task map, RACE can find valid (re)allocations by substituting the original task components suggested by SA-POP with ones that are more resource firendly under the current conditions. In the unusual case that no such allocation is possible, RACE provides feedback to SA-POP indicating its failure to find a valid allocation due to one or more resource constraints. If this occurs, SA-POP generates a new operational string that uses less resources (and probably has less EU). However, this does not require the spreadinga ctivation process to be repeated.

This loose coupling of SA-POP and RACE through a feedback loop, enables operational string generation as a search through a smaller space of potential resource-committed plans. The search is computationally less intensive than if resources were considered at the fine-grained node level. Similarly, RACE does not have to consider the cascading task choices of planning to find a valid allocation, so its search space is also limited to a manageable size. Moreover, SA-POP only considers the *feasibility* of resource allocation in generating operational strings, while RACE considers the harder resource *optimization* problem, but limits it to a given operational string. The limited size and complexity of the search spaces used in SA-POP and RACE, as well as the flexibility afforded by the task map, yields an architecture that can scale to large planning and allocation problems without becoming intractable.

In generating the operational string from mission goals, SA-POP takes into account domain uncertainty by preferring operational strings of high expected utility. Rather than attempting the often intractable problem of finding operational strings with the highest overall expected utility, SA-POP's generates operational strings using a greedy approximation algorithm. The greedy choice of high expected utility tasks still yields a robust application as specified by the resulting operational string, but does not require the much greater search time needed to find the optimal solution.

After an application specified by one or more operational strings is deployed, RACE monitors application performance and domain resource utilization using its *Application Monitors* and *Resource Monitors*. If the performance of an operational string falls below its QoS requirement, RACE's control algorithms take corrective actions to achieve the specified QoS requirement. For example, a control algorithm could (1) modify input parameters of one or more parameterized components of the operational string, (2) dynamically update task implementations from the choices available in the task map, and/or (3) redeploy all or part of an application's components to other target nodes to meet end-to-end QoS requirements. These actions help ensure that the QoS requirements of each operational string are met and resource utilization is maintained within specified bounds. If these control adaptations can not correct/-prevent a QoS or resource violation, however, RACE notifies SA-POP, triggering replanning.

For example, when a resource shortage that cannot be corrected occurs due to unexpectedly high resource consumption by one or more tasks, RACE notifies SA-POP of the problem and the offending task(s). Replanning is then done efficiently by continuing the original operational string extraction after removing the offending task(s) and reintroducing the resulting unsatisfied (sub)goals as open conditions, which is analogous to the plan repair performed by other dynamic partial order planners. After a revised operational string has been generated, SA-POP transmits it to RACE. The application specified by this operational string will usually only be minimally different from the currently deployed application because the expected utilities and resource usage of other tasks and task implementations have not changed, thereby minimizing the amount of work necessary for RACE to reconfigure or redeploy components.

In addition to varying levels of resource utilization, runtime changes can occur in the environmental/system conditions represented in the task network. RACE continuously monitors these conditions and provides feedback on changes to SA-POP. SA-POP uses this information to incrementally update the probability values of conditions in the network, running forward propagation as necessary. Most changes correspond to the expected behavior of applications specified by operational strings. By keeping the task network up-to-date in this manner, when a critical, unexpected change does occur, it can be handled more quickly. Critical changes are those that render the current application deployment nonfunctional for the achievement of some mission goal(s). As in the case of resource shortages, SA-POP performs plan repair by continuing operational string extraction with an open condition corresponding to the unexpectedly changed condition.

Revisions to mission goals are other runtime changes that may require modifications to deployed applications, *e.g.*, due to autonomous data analysis onboard a satellite or revisions from mission scientists on the ground. In either case, the new/changed utility values for goals are inserted into the task network and the spreading activation mechanism is used to update it. These changes generally occur only for a small subset of the mission goals and thus must be propagated through a relatively small portion of the full network. Moreover, only backpropagation of utility is necessary since probability values already forward propagated through the network are unchanged, which allows the task network to be updated efficiently.

With the updated task network, a new operational string is extracted using the same process as for the original operational string. In this case, the operational string extraction usually takes much longer than for plan repair because it must be completely regenerated in order to take advantage of the changed expected utilities. Fortunately, revised mission goals rarely render the current application deployment nonfunctional for all goals. In fact, unless the goals have changed drastically, the current operational string is probably still of high utility. As such, an immediate response to goal changes is not as critical as in the cases necessitating plan repair, so the time to extract a completely new operational string is insignificant in practice.

References

Bagchi, S.; Biswas, G.; and Kawamura, K. 2000. Task Planning under Uncertainty using a Spreading Activation Network. *IEEE Transactions on Systems, Man, and Cybernetics* 30(6):639–650.

Baptiste, P., and Pape, C. L. 1996. Edge-Finding Constraint Propagation Algorithms for Disjunctive and Cumulative Scheduling. In *Proceedings of the Fifteenth Workshop of the U.K. Planning Special Interest Group.*

Heineman, G. T., and Councill, B. T. 2001. *Component-Based Software Engineering: Putting the Pieces Together.* Reading, Massachusetts: Addison-Wesley.

Laborie, P. 2003. Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results. *Artif. Intell.* 143(2):151–188.

Nguyen, X., and Kambhampati, S. 2001. Reviving Partial Order Planning. In Nebel, B., ed., *IJCAI*, 459–466. Morgan Kaufmann.

Object Management Group. 2003a. *Deployment and Configuration Adopted Submission*, OMG Document ptc/03-07-08 edition.

Object Management Group. 2003b. *Light Weight CORBA Component Model Revised Submission*, OMG Document realtime/03-05-05 edition.

Object Management Group. 2002. *Real-time CORBA Specification*, OMG Document formal/02-08-02 edition.

Pape, C. L. 1994. Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems. *Intelligent Systems Engineering* 3(2):55–66.

Schmidt, D. C.; Schantz, R.; Masters, M.; Cross, J.; Sharp, D.; and DiPalma, L. 2001. Towards Adaptive and Reflective Middleware for Network-Centric Combat Systems. *CrossTalk.*

Sharma, P.; Loyall, J.; Heineman, G.; Schantz, R.; Shapiro, R.; and Duzan, G. 2004. Component-Based Dynamic QoS Adaptations in Distributed Real-time and Embedded Systems. In *Proc. of the Intl. Symp. on Dist. Objects and Applications (DOA'04).*

Sharp, D. C., and Roll, W. C. 2003. Model-Based Integration of Reusable Component-Based Avionics System. In *Proc. of the Workshop on Model-Driven Embedded Systems in RTAS 2003.*

Smith, D.; Frank, J.; and Jonsson, A. 2000. Bridging the Gap Between Planning and Scheduling. *Knowledge Engineering Review* 15(1):61–94.

Srivastava, B., and Kambhampati, S. 1999. Scaling up Planning by Teasing out Resource Scheduling. In Biundo, S., and Fox, M., eds., *ECP*, volume 1809 of *Lecture Notes in Computer Science*, 172–186. Springer.