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

John S. Kinnebrew, Nishanth Shankaran, Gautam Biswas, and Douglas C. Schmidt
Department of Electrical Engineering and Computer Science & ISIS,
Vanderbilt University, Nashville, TN 37203, USA

Abstract

Middleware is increasingly being used to develop and deploy components in large-scale distributed real-time and embedded (DRE) systems, such as the proposed NASA sensor web composed of networked remote sensing satellites, atmospheric, oceanic, and terrestrial sensors. Such a system must perform sequences of autonomous coordination and heterogeneous data manipulation tasks to meet specified goals. For example, accurate weather prediction requires multiple satellites that fly coordinated missions to collect and analyze large quantities of atmospheric and earth surface data. The efficacy and utility of the task sequences are governed by dynamic factors, such as data analysis results, changing goals and priorities, and uncertainties due to changing environmental conditions.

One way to implement task sequences in DRE systems is to use *component middleware* (Heineman & Councill 2001), which automates remoting, lifecycle management, system resource management, and deployment and configuration. In large DRE systems, the sheer number of available components often poses a combinatorial planning problem for identifying component sequences to achieve specified goals. Moreover, the dynamic nature of these systems requires runtime management and modification of deployed components.

To support such DRE systems, we have 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. To address this problem, we 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 (see http://www.dre.vanderbilt.edu/ schmidt/WCCD.pdf for an overview of RACE). The separation of concerns between SA-POP and RACE allows more efficient planning and resource allocation.

To ensure applications do not violate resource constraints, the planner requires knowledge of each task's resource consumption and execution time. A given task may be associated with multiple parameterized components, each with different resource information. SA-POP and RACE therefore use a shared *task map* that maps each task to a set of parameterized components and their associated resource information. SA-POP produces a plan for an application, called 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.

SA-POP operates on a spreading activation network (Bagchi, Biswas, & Kawamura 2000), whose structure captures the functional relationships between tasks (implemented as components) and system/environmental conditions (including goals). In this network, utility values capture the importance of desired goals, and probabilities capture the likelihood of tasks succeeding under different conditions.

Probability values are propagated forward through the network from preconditions through tasks to effects. Preconditions and effects can represent both traditional conditions and data streams, defining sequential and producer/consumer relationships between tasks. Utility values are propagated backward through the network from effects through tasks to preconditions, which allows preconditions of potentially useful tasks to accumulate utility as subgoals. The combination of utility and probability of success results in an expected utility value for each task.

As more rounds of propagation are performed, the network computes expected utilities "considering" progressively longer sequences of tasks. When there is sufficient time for full deliberation, this is performed until the network has reached a steady state. If there is only limited time before planning decisions must be made, this process can be stopped at any point. In this case, the computed values correspond to expected utilities when only considering task

sequences of limited length.

SA-POP uses four hierarchical decision points with backtracking in the generation of operational strings. Each step in the generation of an operational string 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 briefly 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 usually resource intensive and tend to be concurrent with other data tasks in DRE domains, SA-POP gives priority to data flow conditions. This heuristic also enables early detection of resource violations in operational strings.

Task Choice. Task choice is based on expected utility. Tasks with higher expected utilities are preferred, provided their likelihood of success for the open condition exceeds a pre-defined threshold. This is a tradeoff between the total expected utility, which may accumulate from multiple goals, and the likelihood of achieving the subgoal currently under consideration.

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.

Scheduling Decision(s). In tracking resource constraints and finding resource violations, SA-POP makes extensive use of the ordering constraints between tasks. In DRE systems, 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 limit start/end time windows.

We are currently integrating the planning capabilities of SA-POP with RACE allocation and control algorithms. RACE determines an efficient allocation for deployment of the operational string generated by SA-POP and monitors the performance of the deployed application. If performance falls below specified QoS requirements, RACE control algorithms take corrective actions, such as dynamically updating task implementations from the task map, and/or redeploying components to other target nodes. If these control adaptations can not correct/prevent a QoS or resource violation, however, RACE notifies SA-POP, triggering plan repair to produce a revised operational string.

Combining the decision-theoretic, resource-constrained planning of SA-POP with the component allocation and runtime management of RACE promises an efficient and scalable architecture for DRE systems operating in dynamic and uncertain domains. The loose coupling of SA-POP and RACE enables SA-POP to generate operational strings as a search through a limited space of potential resource-committed plans, without considering node-level allocation. Similarly, RACE does not have to consider the cascading task choices of planning to find an allocation of components to available nodes, so its search space is also limited to a more manageable size. Moreover, SA-POP only considers the *feasibility* of resource allocation and scheduling at a system level, while RACE considers the harder node-level resource and allocation *optimization* problem, but limits it to the 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 we expect to scale to large planning and allocation problems without becoming intractable.

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.

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.

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.