A Decision-Theoretic Planner with Dynamic Component Reconfiguration for Distributed Real-time and Embedded Systems

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

Middleware is increasingly being used to develop and deploy components in distributed real-time and embedded (DRE) systems, such as the NASA sensor web[1] composed of networked remote sensing satellites, atmospheric, oceanic, and terrestrial sensors. Such systems must perform sequences of autonomous coordination and heterogeneous data manipulation tasks to meet specified goals [Chien *et al.*, 2005], *e.g.*, accurate weather prediction requires multiple satellites to fly coordinated missions that collect and analyze large quantities of atmospheric and earth surface data. The utility and quality of service (QoS) 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 and 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.

We have developed a novel computationally efficient algorithm called the *Spreading Activation Partial Order Planner* (SA-POP) to support dynamic (re)planning under uncertainty in DRE systems. To handle the complementary problems of resource allocation and dynamic control, we combined SA-POP with a *Resource Allocation and Control Engine* (RACE) [Shankar *et al.*, 2005]. RACE is a reusable component middleware framework that separates resource allocation and control algorithms from the underlying middleware deployment and configuration mechanisms to enforce QoS requirements.

To ensure applications do not violate resource constraints, SA-POP 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 use a shared *task map* that maps each task to a set of parameterized components, and their associated resource information. SA-POP produces a deployment plan for an application, where each goal is mapped to an individual *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 the input to RACE, whose algorithms then (re)deploy components in the string onto nodes and manage system resource usage and overall performance.

SA-POP operates on a spreading activation network [Bagchi *et al.*, 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 steps of propagation are performed, SA-POP's activation network computes expected utilities by considering progressively longer sequences of tasks, as well as utilities associated with multiple goals. When there is sufficient time for full deliberation, this propagation is performed until the network has reached a steady state. If there is only limited time before planning decisions must be made, the spreading activation algorithm can be stopped at any point, and the operational string can be generated using the expected utility values computed to that point. The expected utility values may not be at their optimal value, but they provide a useful metric for choosing tasks.

SA-POP uses four hierarchical decision points with backtracking to generate the operational strings. Each

[1]See `sensorwebs.jpl.nasa.gov` for details.

step in the generation of an operational string involves the following layered decision points:

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 preference is a tradeoff between 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. 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 have integrated 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 cannot correct or prevent a QoS or resource violation, RACE notifies SA-POP, and this triggers a plan repair that produces a revised operational string. This separation of concerns between SA-POP and RACE allows for more efficient planning and resource allocation.

Combining the decision-theoretic, resource-constrained planning of SA-POP with the component allocation and runtime management of RACE provides an efficient and scalable architecture for DRE systems operating in dynamic and uncertain domains. The loose coupling of SA-POP and RACE enables the generation of operational strings as a search through a limited space of potential resource-committed plans, without considering node-level allocation. Similarly, RACE need not 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. We have run experiments with the SA-POP and RACE system to address data gathering and analysis for multi-satellite NASA systems used in weather predictions [Kinnebrew *et al.*, 2007]. 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, such as a large sensor web, without becoming intractable.

SA-POP and RACE are open-source software available from `www.dre.vanderbilt.edu/RACE`.

References

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

[Chien *et al.*, 2005] Steve Chien, Benjamin Cichy, Ashley Davies, Daniel Tran, Gregg Rabideau, Rebecca Castano, Rob Sherwood, Dan Mandl, Stuart Frye, Seth Shulman, Jeremy Jones, and Sandy Grosvenor. An Autonomous Earth-Observing Sensorweb. *IEEE Intelligent Systems*, 20(3):16–24, 2005.

[Heineman and Councill, 2001] George T. Heineman and Bill T. Councill. *Component-Based Software Engineering: Putting the Pieces Together.* Addison-Wesley, Reading, Massachusetts, 2001.

[Kinnebrew *et al.*, 2007] John S. Kinnebrew, Ankit Gupta, Nishanth Shankaran, Gautam Biswas, and Douglas C. Schmidt. A Decision-Theoretic Planner with Dynamic Component Reconfiguration for Distributed Real-Time Applications. Submitted to International Symposium on Autonomous Decentralized Systems, 2007.

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

[Shankar *et al.*, 2005] Nishanth Shankar, Jai Balasubramanian, Douglas C. Schmidt, Gautam Biswas, Patrick Lardieri, Ed Mulholland, and Tom Damiano. A Framework for (Re)Deploying Components in Distributed Real-time and Embedded Systems. In *Poster paper in the Dependable and Adaptive Distributed Systems Track of the 21st ACM Symposium on Applied Computing*, Dijon, France, April 2005.