Integrating Task Allocation, Planning, Scheduling, and Adaptive Resource Management to Support Autonomy in a Global Sensor Web

John S. Kinnebrew, Gautam Biswas,
Nishanth Shankaran, and Douglas C. Schmidt
EECS Department & ISIS,
Vanderbilt University,
Nashville, TN 37203, USA
john.s.kinnebrew@vanderbilt.edu

Dipa Suri
Lockheed-Martin Space Systems Company
Advanced Technology Center
3251 Hanover Street,
Palo Alto, CA. 94304
dipa.suri@lmco.com

Abstract

NASA's Earth Science Vision calls for a global sensor web comprised of heterogeneous platforms with on-board information processing, capable of orchestrating real-time collaborative operations with other platforms and ground stations. Such a global sensor web will be a system of systems, including many distributed real-time embedded (DRE) systems, such as multi-satellite formations. Individual systems of the sensor web must collect and analyze large quantities of data via sequences of heterogeneous data collection, manipulation, and coordination tasks to meet specified goals for earth science applications. In large DRE systems, such as those composing a global sensor web, 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 present the design of the Multi-agent Architecture for Coordinated Responsive Observations which includes two novel services contributing to the design and deployment of autonomous, predictable, and high performance DRE systems that operate in dynamic and uncertain environments: (i) the Spreading Activation Partial Order Planner (SA-POP) that performs decision-theoretic planning and scheduling using a spreading activation network to capture the probabilistic functional relationships between tasks (implemented as components) and goals; and (ii) the Resource Allocation and Control Engine (RACE), which is an open-source adaptive resource management framework built atop standards-based QoS-enabled component middleware. We illustrate the effectiveness of our approach in the face of changing operational conditions, workloads, and resource availability, in the context of salient Earth science missions.

1. Introduction

Remote sensing missions for Earth Science provide a wealth of information to help scientists understand the dynamics of our planet. Conventional approaches use a stove-pipe operational model, such as each spacecraft or *in situ* sensor cluster commanded by, and transmitting data to, large dedicated ground operations centers. These approaches, however, introduce untenable latencies in developing data products that hinder model building and refinement. Moreover, the inherent communication lag and potentially limited bandwidth necessitates autonomous planning and resource allocation at a local level to effectively achieve goals under rapidly evolving environmental and system conditions. To address these limitations, NASA's Earth Science Vision calls for a global sensor web comprised of heterogeneous platforms with on-board information processing, capable of orchestrating real-time collaborative operations with other platforms and ground stations [7].

Many of the platforms comprising the sensor web will be distributed, real-time embedded (DRE) systems, such as spacecraft and airborne systems. Modern DRE systems implement task sequences, such as data processing, using *component middleware* [6], which automates remoting, lifecycle management, system resource management, deployment, and configuration. However, in large-scale DRE systems, the sheer number of component sequences often poses a combinatorial deployment problem, *i.e.*, mapping components to computing nodes [14]. Moreover, the dynamic nature of the operations require runtime management and modification of deployments [5]. At the level of individual platforms (e.g. individual satellites and ground-based sensor installations) these problems necessitate a system with the ability to make resource allocation and control decisions at runtime. More effective solutions to this problem provide greater autonomy with the inclusion of planning and replanning capabilities to ensure the task sequences being executed keep in sync with changing mission goals and resource availability.

In addition to comprising a vast number of sensors and platforms, a global sensor web will also have many "users" (e.g. weather modeling and prediction systems, disaster recognition and management systems, and individual scientists and scientific institutions) requesting access to sensor data and control of the sensor platforms. The scale and scope of such a sensor web requires an efficient, scalable system of control for allocation of high-level task requests to available resources.

To support the needs of future Earth Science Missions, we are developing a Multi-agent Architecture for Coordinated, Responsive Observations (MACRO). MACRO provides a powerful infrastructure for enabling the deployment and operation of a sensor web. Agents using component middleware and novel services, such as SA-POP [10] for supporting dynamic planning/scheduling and RACE [19] for resource allocation/control, provide the necessary local autonomy to react to changing conditions, while efficiently achieving mission goals. At the global, "mission level," agents representing the sensor web users negotiate for sensor web access with agents representing the available sensor platforms through a variant of the well-known contract net protocol [21] for task allocation. This architecture helps overcome current limitations by facilitating real-time, reactive data acquisition, analysis, fusion, and distribution, i.e., a "smart sensing" capability in the sensor web context.

2. Architecture

MACRO employs a set of intelligent agents operating on a quality of service (QoS)-enabled component middleware framework to ensure that the objectives of a remote sensing mission are met, e.g., planning for science operations, science data acquisition, processing, and dissemination. The design of the agents is based on a combination of mature terrestrial standards defined by the Object Management Group (OMG) (`www.omg.org`), the Foundation for Intelligent Physical Agents (FIPA) (`www.fipa.org`), and the Open Geospatial Consortium (`www.opengeospatial.org`). The implementation of the agents is based on a state-of-the-art component middleware implementation of the CORBA Component Model (CCM) to ensure interoperability across heterogeneous computing platforms (e.g., various processors, operating systems, and programming languages), reduce development costs, and improve the software's robustness and scalability [22].

Recent research has identified a number of general organizational structures employed by multi-agent systems, each with its own benefits and drawbacks [11, 8]. However, the scale and scope of a global sensor web does not easily lend itself to any single organizational paradigm. Rather than use a single one-size-fits-all organizational solution, MACRO employs multiple organization structures as appropriate to the design requirements of various aspects of the system.

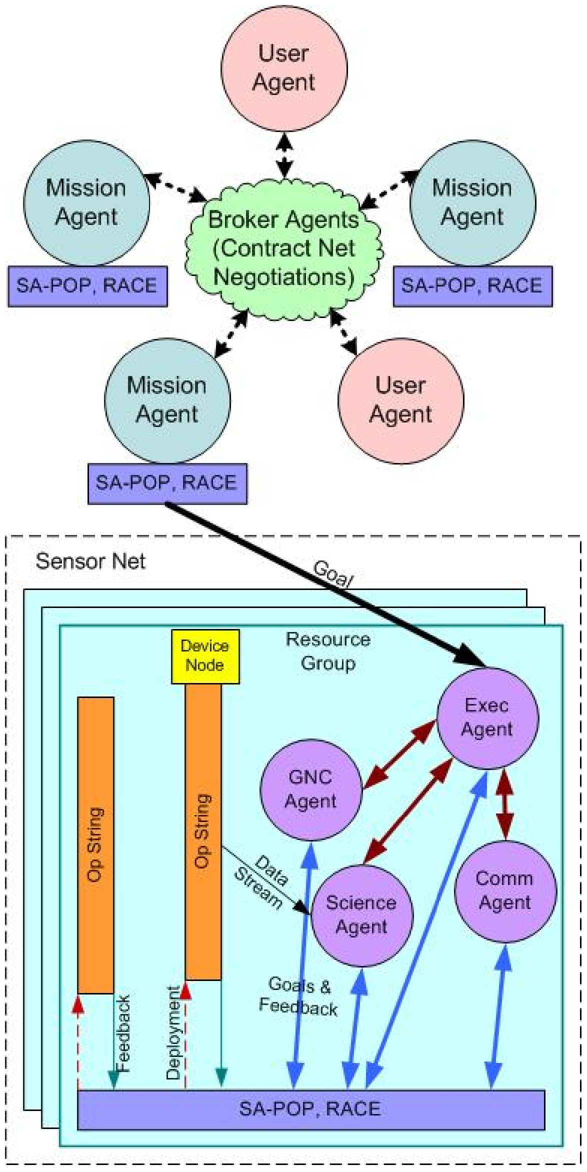

Figure 1. System architecture of MACRO

As illustrated in Figure 1 there is a broad, two-level hierarchy of control. The higher level, or "mission level," is comprised of *user agents*, *mission agents*, and *broker agents*. User agents provide the high-level tasks/goals to be achieved by the system. As such, the user agents are interfaces to mission scientists or wrappers for legacy systems that can request sensor web tasks, such as weather modeling and simulation. The mission agents then perform these tasks with the resources under their control. Specifically, each mission agent controls a *sensor net* as seen in Figure 1. This hierarchy is a natural result of the sensor web's structure in which there are many low-level DRE systems with limited computational resources that are supported and directed by relatively few computing facilities with far greater resources. Moreover, this hierarchical structure can take advantage of the efficiency gained through task decomposition and separation of concerns [8]. The disadvantage of a hierarchical structure is the potential for brittleness and bottlenecks [8]. In MACRO, these disadvantages are mitigated by the fact that the top level of the hierarchy is a group of mission agents rather than a single agent, and this group can be dynamically expanded as additional sensor nets are added to the global sensor web.

Each sensor net controlled by a mission agent is composed of *resource groups*. A resource group is a natural grouping of computational resources that are directly connected to device nodes (*i.e.* sensors and actuators). For example, a spacecraft may have multiple processors with a range of connected device nodes, but it can logically be viewed as a single set of shared resources. Similarly a cluster of *in situ* ground sensors connected by a bus to one or more processing nodes would be a single resource group. The key characteristic of a resource group is that it is a set of *shared* device nodes and computational resources. Within the resource group, the agents are organized by a federation paradigm. There is a single agent, the exec agent, acting as the intermediary between the other resource group agents and the controlling mission agent.

In general, the resource group agents are defined by their roles and responsibilities. The particular set of agents used in a resource group depends on the physical system itself. For example, a spacecraft requires guidance and navigation, while an *in situ* ground cluster does not. The central agent, which exists in any resource group, is the *exec agent* [23]. The exec agent acts as a mediator between the other resource group agents and the controlling mission agent. It also arbitrates conflicting resource requests between other agents in its resource group and is responsible for monitoring overall resource group health (e.g. fault detection).

Other resource group agents are included as appropriate to the platform. For example, a spacecraft may include a *comm agent*, *GNC agent*, and *science agent(s)* [23]. The comm agent manages all aspects of communication between the spacecraft and external facilities. This will entail control of the communication hardware and services, which is particularly important in a limited telemetry bandwidth situation. A GNC ("Guidance Navigation & Control") agent is responsible for spacecraft guidance, navigation, and attitude control through use of appropriate sensors and actuators. Finally, one or more science agents are responsible for decision-making associated with achieving science objectives, such as choice of sensor configurations and data processing.

All MACRO agents communicate using messages based on the FIPA Agent Communication Language (ACL) [4]. It has been noted that the FIPA ACL is ill-suited for some forms of agent interaction, such as argumentation [13]. However, the MACRO architecture primarily consists of cooperative, trusted agents, and as such, their interactions correspond to a rational-choice view of agent society for which FIPA ACL was designed.

While FIPA ACL defines the high-level message structure, it does not require the use of any particular language or ontology for the message content. Further, one of the major issues in ACL design is the specification of an ontology with "broad coverage, [relevance] to its domain, and [extensibility]" [3]. The MACRO ontology addresses this issue by conforming to the current Open Geospatial Consortium (OGC) (`www.opengeospatial.org`) experimental SensorML standard [2] for describing sensors, data processing, and location information. MACRO agents use message content in the SensorML format to effectively communicate capabilities, tasks, and data. This also allows MACRO to support interoperability with other external tools and systems based on the OGC Sensor Web Enablement standards.

3. MACRO Contract Net

One of the major problems addressed by MACRO is the allocation of resources (sensors and computational power for data analysis) to sensor web users such as weather modeling and disaster prediction/management systems. The contract net protocol [21] is one of the earliest and most well-studied algorithms for resource-allocation/task-distribution in multi-agent systems. It has the advantages of conceptual simplicity, a relatively straightforward implementation, and proven effectiveness in real-world applications.

The interactions in a basic contract net implementation are based on a simple marketplace auction metaphor. When an agent needs a task completed that it can not, or does not wish to, do itself, it announces it to all other agents or a subset of agents known to be interested in this type of task. For this interaction the announcing agent is called the "manager" and the other agents are potential "contractors." Over any period of time, each potential contractor may receive multiple task announcements, all of which contain a time deadline for receipt of bids. The potential contractor ranks task announcements by a known preference heuristic and may reply to the highest ranked announcement with a bid to the appropriate manager. The decision and timing of a contractor's bidding is decided by its own timeout clock and potentially a cutoff value for its interest in an announcement (below which it will not bid at all). The manager ranks incoming bids by its own preference heuristic and, before the deadline, awards the contract to the contractor with the highest ranked bid.

One of the major advantages of the contract net protocol is its ability to accommodate a heterogeneous collection of agents with both different resources and different priorities for bidding and choosing bids. However, this can also introduce significant communication and computation overhead for matching capabilities of agents with task announcements. To minimize this overhead, MACRO introduces broker agents that act as intermediaries between managers and contractors. Contractors (mission agents) inform an assigned broker of their interest in task announcements by specifying their sensor capabilities and geographic coverage. Mission agents with mobile resources (*e.g.* spacecraft or airborne platforms) regularly update their broker agent with current and near-term geographic coverage information. This allows broker agents to match task announcements to the subset of interested mission agents, thereby reducing communication overhead. Further, all broker agents are registered with standard middleware naming and trading services, providing some fault tolerance in the event of an individual broker agent malfunction. The small set of broker agents also performs some load balancing amongst themselves in assigning each mission or user agent to a particular broker.

Another extension to the basic contract net protocol in MACRO is the use of currency in bidding and contracts. MACRO broker agents provide and enforce currency allocation to user agents over time. Currency is allocated by a share of total resources and unused currency has a variable decay over time. This is intended to allow an approximation of fair share access to the sensor web by the pre-determined share for each registered user agent. Further, it provides some additional security because mission and broker agents are trusted agents with well known rules for currency assignment and cost calculations. Thus user agents, which can be designed independently of the MACRO mission and broker agents, can be controlled to some extent by the MACRO designers. With well-chosen mission agent cost functions (in terms of computational and sensor workload) and task preference heuristics, this may also help provide load balancing in the sensor web.

While fair share access to sensor web resources is desired in baseline MACRO operation, prediction and management of natural disasters is also an important, although less common, use case. Disaster management is accommodated in MACRO by allowing priority task announcements from pre-specified user agents. All applicable mission agents are required to bid on priority task announcements, overriding the normal fair share operation of the contract net.

4. Middleware and Services

To perform efficiently in a distributed, heterogeneous computing environment, MACRO agents rely on the underlying component middleware. The CORBA Component Model provides a way to logically bundle interfaces into service families and specify the configuration and deployment of objects as complete applications. This results in flexible, scalable implementations that are easier to adapt and maintain. The component model was introduced as a solution where components encapsulate application "business logic" and interact via well-defined ports. Standard container mechanisms provide an execution environment for components with common operating requirements and a reusable/standard infrastructure configures and deploys components throughout a distributed system.

The Component Integrated ACE ORB (CIAO) and the Deployment and Configuration Engine (DAnCE) are open source implementations of the OMG's Lightweight CORBA Component Model (CCM) [17] and Deployment and Configuration (D&C) [16] specifications. CIAO and DAnCE are built atop The ACE ORB (TAO). TAO is a highly configurable, open-source, real-time CORBA Object Request Broker (ORB) that implements key patterns to meet the demanding QoS requirements of distributed, real-time embedded systems. CIAO extends TAO by abstracting key QoS concerns (such as priority models, thread-to-connection bindings, and timing properties) into elements that can be configured declaratively via metadata (such as standards for specifying, implementing, packaging, assembling, and deploying components). Defining and configuring QoS properties as metadata disentangles code for controlling these concerns unrelated to functionality from code that implements the application logic, thus making MACRO development more flexible and productive. DAnCE extends TAO by allowing application deployers to specify how existing components should be packaged, assembled, and customized into reusable services.

In addition to the component middleware infrastructure, MACRO agents require the ability to create and execute plans. Most agent architectures include planning as an integral part of an individual agent's reasoning mechanisms. However, the need for scheduling and awareness of shared resource utilization in a MACRO resource group complicates this picture. Rather than directly intertwining planning with the meta-level reasoning of MACRO agents, they use a (re)planning and scheduling service to efficiently meet these needs. This service is provided by the Spreading Activation Partial Order Planner (SA-POP) [10]. To effectively implement their plans, MACRO agents also require the ability to (re)allocate and manage components as mission goals and resource availability change. This service is provided by the Resource Allocation and Control Engine (RACE) [19], whose operation, in conjunction with SA-POP, is illustrated in Figure 2.

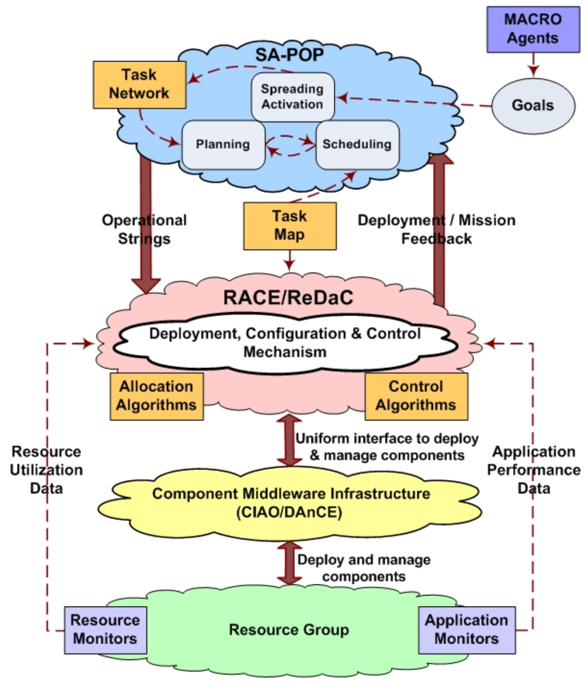
MACRO Agents
SA-POP
Task Network
Spreading Activation
Goals
Planning
Scheduling
Operational Strings
Task Map
Deployment / Mission Feedback
RACE/ReDaC
Deployment, Configuration & Control Mechanism
Allocation Algorithms
Control Algorithms
Uniform interface to deploy & manage components
Resource Utilization Data
Application Performance Data
Component Middleware Infrastructure (CIAO/DAnCE)
Deploy and manage components
Resource Monitors
Resource Group
Application Monitors

Figure 2. SA-POP and RACE architecture

5. Spreading Activation Partial Order Planner

When an agent provides a desired goal, SA-POP evaluates current/predicted conditions and resource availability to generates a plan of high expected utility as an *operational string*. An operational string is primarily a partially ordered set of tasks to accomplish a goal. Each goal specification is mapped onto one operational string, which includes the control (ordering) dependencies, the data (producer/-consumer) dependencies, and required start and end times for tasks, if any. The operational strings also indicate appropriate implementations, *e.g.* components, for each task, as there may be more than one.

For SA-POP to choose appropriate tasks to achieve goals, it must know which preconditions must be satisfied for each task, its input/output data streams, if any, and the pertinent effects that result from its operation. Uncertainty as to whether tasks will produce the desired output or effects is captured via conditional probabilities associated with the preconditions and effects of a task. Together, the input/output definitions, preconditions/effects, and related conditional probabilities define the *functional signature* of the task.

The functional signature of each task, and consequently all task dependencies, are captured in a *task network*. A task network is a directed graph that represents both tasks and conditions (preconditions, data input, effects, and data output) with the links encoding the requisite probability information. In general, the task network can be constructed by a domain expert using domain-specific modeling tools, such as the Generic Modeling Environment (GME) [9]. With the task network, current values for conditions, and a goal, SA-POP's spreading activation mechanism can compute expected utility values for each task. However, to ensure operational strings do not violate resource constraints, SA-POP also requires knowledge of the expected resource consumption and execution time of each possible implementation of a task, *i.e.*, its *resource signature*. SA-POP uses a system-specific *task map* to associate each task with a set of implementations and their individual resource signatures.

To allow greater flexibility in making deployment and runtime control decisions, SA-POP is designed to impose a minimum of constraints in an operational string. To facilitate this, we adopt a modified *Partial Order Causal Link* (POCL) design [20] in the generation of operational strings. The least commitment strategies typical of partial order planning allow SA-POP to impose relatively few constraints compared to other popular planning techniques, such as state space search and constraint satisfaction based planners.

SA-POP leverages information from the partial order planning process when applying resource constraints and finding resource violations. In DRE systems, such as an earth science satellite, many data manipulation tasks operate over long time windows with a required start time, but the end time is dynamically determined by ongoing analysis of the data. This lack of defined end time limits the effectiveness of many popular scheduling approaches such as timetabling [18], edge-finding [1], and classical energetic reasoning [12].

Rather than primarily relying on start/end time window manipulation, SA-POP leverages the ordering constraints common to partial order plans. These constraints are used to create *precedence graphs* [12], which partition all other tasks into sets based on their ordering with respect to a particular task under consideration. SA-POP then applies a modified version of Laborie's energy precedence and balancing constraint propagation techniques [12] to add appropriate timing and ordering constraints to the operational string. SA-POP generates operational strings using mutually recursive planning and scheduling algorithms with backtracking. Each step in the generation of an operational string involves four mutually recursive algorithms: (1) *Plan*, which first chooses an *open condition* (*i.e.* a goal or subgoal unsatisfied in the current plan), then chooses a task that can achieve the open condition, (2) *ResolveThreats*, which resolves causal link threats by promotion or demotion, (3) *Schedule*, which chooses an implementation for the task instance and propagates resource constraints to constrict time windows and impose necessary scheduling links, and (4) *ResolveResources*, which adds scheduling links to resolve potential resource violations. The first two algorithms, *Plan* and *ResolveThreats*, correspond to traditional partial order planning. The other two algorithms, *Schedule* and *ResolveResources*, perform resource constrained scheduling through constraint propagation and a search to resolve potential resource violations.

6. Resource Allocation and Control Engine

The architecture of RACE and its interplay with SA-POP is illustrated in Figure 2. RACE performs autonomous resource (re)allocation and (re)configuration of QoS settings of components that are part of the operational strings generated by SA-POP such that the QoS requirements of the operational strings are met. RACE is built atop of CIAO and DAnCE, which are open-source (see `www.dre.vanderbilt.edu`) implementations of the OMG Lightweight CCM [17], Deployment and Configuration (D&C) [16], and Real-time CORBA [15] 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. In particular, it uses *Resource Monitors* and *ApplicationQoSMonitors*, 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. For example, an allocation algorithm could apportion CPU resources to components in such a way that avoids saturating these 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. For example, a control algorithm could (1) modify an application's current operating mode, (2) dynamically update component implementations, and/or (3) redeploy all or part of an operational string's components to other target nodes to meet end-to-end QoS requirements.

RACE uses mechanisms provided by the underlying middleware to perform the allocation and control decisions made by its algorithms. For example, RACE uses standard mechanisms defined by the Lightweight CORBA Component Model (CCM) [17] to (1) (re)deploy and (re)configure application components, (2) transition application components from idle states to operational states and monitor the performance of the DRE system, and (3) modify components and/or operational strings to realize the adaptation decisions of control algorithms.

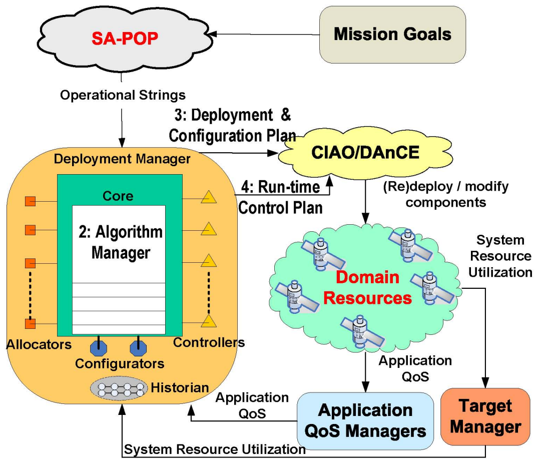

Figure 3. Architecture of RACE

As shown in Figure 3 the RACE architecture consists of the following entities that are implemented as CCM components using CIAO and deployed via DAnCE:

Resource Monitors are CCM components that track resource utilization in a domain. One or more *ResourceMonitors* are associated with each domain resource, such as CPU and memory utilization monitors on each node and network bandwidth utilization monitors on interconnects.

ApplicationQoSMonitors are CCM components that track the performance of application components by observing QoS properties, such as throughput and latency. One or more *ApplicationQoSMonitors* are associated with each type of application component.

The **TargetManager** [19] is a CCM component defined in the D&C specification [16] that receives periodic resource utilization updates from *ResourceMonitors* within a domain. It uses these updates to track resource usage of all resources within the domain. The *TargetManager* provides a standard interface for retrieving information pertaining to resource consumption of each component or assembly in the domain, as well as the domain's overall resource utilization.

The **DeploymentManager** is an assembly of CCM components that encapsulates and coordinates one or more allocation and control algorithms. This manager deploys assemblies by allocating resources to individual components in an assembly. After assemblies are deployed, the *DeploymentManager* manages the performance of (1) operational strings and (2) domain resource utilization. This manager ensures desired performance of the operational strings by performing the following actions to the components that make up the operational strings: (1) (re)allocating resources to the component, (2) modifying component parameters such as execution mode, and/or (3) dynamic replacing the component implementations.

7. Discussion and Lessons Learned

This section summarizes our experiences 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 autonomous operation of DRE systems in the context of MACRO. SA-POP produces partial-order plans as operational strings that contain sufficient information to be instantiated with parameterized component implementations without violating coarse-grained resource constraints.

In a satellite formation, for example, an instantiation of SA-POP on each satellite considers the computational resources, such as CPU, memory, and communication bandwidth to be monolithic, discrete resources. In actuality, there may be multiple nodes with individual CPU and memory capacities. However, each task generally only uses a small fraction of these resources, so the course-grained resource constraints used by SA-POP help ensure that RACE can find valid deployments for components on the real node resources.

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 friendly 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 of a particular resource, but may have a lower expected utility, *without* requiring a repetition of the spreading activation.

Semi-autonomous operation of MACRO resource groups with limited computing capacity requires efficient algorithms to handle the combinatorial problems of planning, scheduling, and allocation. The 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 and scheduling 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 can consider 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 operate with limited computational resources, while scaling to relatively 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 tasks 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.

For efficient sensor web operation, individual resource groups require some degree of autonomy to recognize and react to changes in local conditions. To this end, RACE monitors application performance and domain resource utilization using its *Application Monitors* and *Resource Monitors* after operational string deployment. 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.

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. When a critical, unexpected change does occur, it can be handled more quickly because the task network is up-to-date. Critical changes are those that render the current application deployment non-functional for the achievement of some mission (sub)goal condition. In these cases, SA-POP would produce a revised operational string by performing plan repair (i.e. by continuing from the original operational string generation with an open condition corresponding to the critical change).

Revisions to mission goals, *e.g.*, due to on-board data analysis or revisions from mission agents, are other runtime changes that may require modifications to deployed applications. 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 only need 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.

With the updated task network, a new operational string is generated. In this case, the operational string generation 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 deployed operational string nonfunctional for all goals. In fact, unless the goals have changed drastically, the current operational string is probably still of high expected utility. As such, an immediate response to goal revisions 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.

8. Concluding remarks

In this paper, we described the design of the Multi-agent Architecture for Coordinated Responsive Observations (MACRO). The organization of MACRO agents provides an efficient framework for distributed sensor web operation and control. Further, combining the decision-theoretic (re)planning with resource constraints of SA-POP with the RACE framework for resource allocation and control enables autonomy in the individual systems that make up the sensor web. We showed how SA-POP and RACE can together facilitate autonomous operation by responding to dynamic changes through (re)planning of task sequences and the (re)deployment/(re)configuration of components. RACE and SA-POP are open-source software that can be obtained from `deuce.doc.wustl.edu/Download.html` as part of the CIAO middleware.

References

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

[2] M. Botts et al. Sensor Model Language (SensorML) for In-situ and Remote Sensors. Technical Report OpenGIS Specification Document 04-019r2, Open Geospatial Consortium, November 2004.

[3] B. Chaib-draa and F. Dignum. Trends in Agent Communication Language. *Computational Intelligence*, 18(2):89–101, 2002.

[4] FIPA. Communicative Act Library Specification. Technical Report Technical Report SC00037J, Foundation for Intelligent Physical Agents, December 2002.

[5] X. Gu and K. Nahrstedt. Dynamic QoS-Aware Multimedia Service Configuration in Ubiquitous Computing Environments. In *Proceedings of IEEE International Conference on Distributed Computing Systems*, 2002.

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

[7] P. Hildebrand, W. Wiscombe, M. Albjerg, J. Booth, R. Miller, T. Miller, M. Mlynczak, G. Paules, D. Peterson, C. Raymond, et al. NASA Earth Science Vision 2030: Working Group Report. Technical report, NASA Tech. Report NP-2003-2-611-GSFC. Available at http://neptune. gsfc. nasa. gov/vision, 2004.

[8] B. Horling and V. Lesser. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(04):281–316, 2005.

[9] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-Integrated Development of Embedded Software. *Proceedings of the IEEE*, 91(1):145–164, Jan. 2003.

[10] J. Kinnebrew, A. Gupta, N. Shankaran, G. Biswas, and D. Schmidt. A Decision-Theoretic Planner with Dynamic Component Reconfiguration for Distributed Real-time Applications. In *Proceedings of the 8th International Symposium on Autonomous Decentralized Systems (ISADS)*, Sedona, Arizona, March 2007.

[11] M. Kolp, P. Giorgini, and J. Mylopoulos. Multi-agent architectures as organizational structures. *Autonomous Agents and Multi-Agent Systems*, 13(1):3–25, 2006.

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

[13] P. McBurney, S. Parsons, and M. Wooldridge. Desiderata for agent argumentation protocols. *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*, pages 402–409, 2002.

[14] M. Mikic-Rakic, S. Malek, and N. Medvidovic. Improving Availability in Large, Distributed Component-Based Systems Via Redeployment. In *3rd International Working Conference on Component Deployment (CD 2005)*, Grenoble, France, 2005.

[15] Object Management Group. *Real-time CORBA Specification*, OMG Document formal/05-01-04 edition, Aug. 2002.

[16] Object Management Group. *Deployment and Configuration Adopted Submission*, OMG Document mars/03-05-08 edition, July 2003.

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

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

[19] N. Roy, N. Shankaran, and D. C. Schmidt. Bulls-Eye: A Resource Provisioning Service for Enterprise Distributed Real-time and Embedded Systems. In *Proceedings of the 8th International Symposium on Distributed Objects and Applications*, Montpellier, France, Oct/Nov 2006.

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

[21] R. Smith. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, 29(12):1104–1113, 1980.

[22] D. Suri, A. Howell, D. Schmidt, G. Biswas, J. Kinnebrew, W. Otte, and N. Shankaran. A Multi-agent Architecture for Smart Sensing in the NASA Sensor Web. In *Proceedings of the 2007 IEEE Aerospace Conference*, Big Sky, Montana, March 2007.

[23] D. Suri, A. Howell, N. Shankaran, J. Kinnebrew, W. Otte, D. Schmidt, and G. Biswas. Onboard Processing using the Adaptive Network Architecture. In *Proceedings of the Earth-Sun Science Technology Conference*, College Park, MD, June 2006.