Coordinating Complex Decision Support Activities Across Distributed Applications

Richard M. Adler
Symbiotics, Inc.
725 Concord Avenue
Cambridge, MA 02138
Tel (617)876-3635 Fax (617)876-0157
rma@symbiotics.com

KEY WORDS AND PHRASES

distributed computing, distributed decision support, intelligent coordination, work flow, work groups

INTRODUCTION

Knowledge-based technologies have been applied successfully to automate planning and scheduling in many problem domains [1,2]. Automation of decision support can be increased further by integrating task-specific applications with supporting database systems, and by coordinating interactions between such tools to facilitate collaborative activities. For example, end-to-end decision support for space missions involves a succession of interactions to transfer and manipulate data across diverse tools: deriving mission task, resource, and constraint networks via a planning engine; storing these results in a database; retrieving the mission plan for use as input to a scheduling engine; comparing the resulting schedule against current schedules for other missions to detect resource conflicts; and replanning or rescheduling to resolve problems. Ideally, no human intervention should be required to carry out such activity sequences, which, despite their complex distributed implementation, are otherwise well-defined and routine.

Unfortunately, the technical obstacles that must be overcome to achieve this vision of transparent, cooperative problem-solving are daunting. Intelligent decision support tools are typically developed for standalone use, rely on incompatible, task-specific representational models and application programming interfaces (APIs), and run on heterogeneous computing platforms. Getting such applications to interact freely calls for platform independent capabilities for distributed communication, as well as tools for mapping information across disparate representations [3]. Similarly, coordinating interactions dynamically presupposes capabilities for: identifying and locating required resources and capabilities across a network; capturing relationships between decision support activities such as task decomposition, data dependencies, and synchronization constraints; and autonomously controlling the execution of tasks across applications to reflect such relationships. These system engineering issues are largely orthogonal to the interests and skills of developers and end-users of decision support applications.

Symbiotics is developing a layered set of software tools (called NetWorks!) for integrating and coordinating heterogeneous distributed applications. The top layer of tools consists of an extensible set of generic, programmable coordination services. Developers access these services via high-level APIs to implement the desired interactions between distributed applications. Current API-based services enable developers to: register application services and information resources, their locations, and calling interfaces; model the decomposition or workflow sequence of composite decision support activities in terms of simpler units; and invoke automated control engines that execute composite models to carry out complex activities such as end-to-end decision support for space missions. The high-level coordination services are built on top of a communication substrate layer, which utilizes object-oriented technology to conceal the complexity of platform dependencies, data mapping, and network communication. The remainder of this abstract describes these various tools and how they interoperate as a nonintrusive, extensible framework for developing complex distributed applications.

DISTRIBUTED COMMUNICATION SUBSTRATE

NetWorks! is a communication tool that is based on object-oriented message-passing technology [4]. Messaging systems typically

enable applications to interact by posting and retrieving messages from local queues that are connected transparently across network nodes. This minimal architecture tends to push more complex control behaviors (e.g., coordinating sequences of interactions) into the applications themselves, which impacts their modularity, maintainability, and extensibility. The NetWorks! Messaging Facility (NMF) provides the customary messaging queues, queue management and network transport services across heterogeneous platforms. However, NetWorks! also incorporates active objects called *Agents*, which mediate interactions between applications and local NMFs and isolate any additional behaviors required for integration or distributed control.

Agents consist of object methods that contain: conventional C or C++ code; calls to the native APIs of local applications; and calls to the NetWorks! API library for creating, sending, and retrieving messages. The messaging API provides both blocking and non-blocking (asynchronous) communication models. A supporting Data Management System (DMS) provides an extensible, machine independent "neutral exchange" representation for translating messages across incompatible application data models. Applications initiate distributed interactions via simple messaging API calls to Agents. Agents can: (1) manipulate and forward messages from applications to other Agents via NMFs; (2) interact with applications by injecting or extracting data and commands; and (3) provide other dedicated services. In particular, Agents can integrate generic distributed control models for coordinating interactions among other Agents and applications. The following sections review the three Agent-based coordination services that are already implemented [5].

BROKERING DISTRIBUTED APPLICATION RESOURCES AND SERVICES

A request broker is a dedicated control mechanism that mediates interactions between client applications needing particular resources or problem-solving services and server applications capable of providing them [6]. Brokers free individual applications from the burden of maintaining information locally as to where and how to obtain services that they may require, such as database queries or particular planning and scheduling engines. Instead, all applications within a distributed system register the services they support, their locations, and their calling interfaces with the broker, which typically maintains this information in a directory or naming service. Client applications can then query the broker to find and request the desired interaction. The broker uses the naming service to relay requests from clients to the relevant server applications, retrieve responses, and relay them back to the client.

The NetWorks! Service Request Manager (SRM) consists of an Agent that integrates a dedicated request broker application. The SRM control model also incorporates a shared memory bulletin board structure, which applications can use to post or retrieve information of common utility. The SRM supports a high-level API that includes functions for: dynamically registering applications and services; requesting services; adding and deleting information items from the bulletin-board; and querying the services directory and bulletin-board to search for particular items of interest. The SRM API is built on top of lower level NetWorks! messaging and DMS APIs.

COORDINATING DISTRIBUTED WORKFLOW

One approach to automating sequences of activities such as end-to-end decision support for space missions is to establish directed, data-driven control links between the relevant applications. Distributing control logic in this manner is cumbersome to maintain and extend, particularly in systems that support many composite activities and that evolve through incremental additions of applications and services.

The NetWorks! Process Planner provides an alternative process-oriented model for distributed coordination, which consists of a high-level scripting language and a control Agent that executes scripts. This model alleviates the difficulties of highly distributed schemes by capturing the coordination logic for workflows in centralized, compact scripts that are easily maintained and extended. Individual script steps take the form of "atomic" requests for specific services or tasks, such as transferring data between two applications or scheduling ground operations for a Shuttle mission. The scripting language also

incorporates control structures to represent data dependencies, temporal ordering, and other synchronization constraints across atomic script steps, including binding variables to store input arguments or results of script steps for later use, conditional branching, and iteration. Mutually independent tasks can be grouped explicitly for concurrent execution. A nested invocation primitive enables scripts to be embedded in other scripts.

Clients use a simple message-based API call to invoke the Process Planner to initiate a specified script. The Process Planner Agent incorporates a script interpreter engine that instantiates that script and executes its constituent steps in the specified sequence. For example, the mission support script would input specified mission profile parameters to an intelligent planning engine, transfer the results to a database, and so on. Script steps are executed by sending messages requesting the specified services to an associated SRM. The SRM forwards requests to the relevant servers, and relays responses back to the Process Planner, which then reiterates these behaviors, updating interim variables, testing control constraints, and requesting any script steps that are ready to be executed. Upon completing the script, the Agent returns results to the client, such as a verified mission schedule. In essence, the Process Planner functions as a workflow driver to the SRM, which brokers individual task requests. The two coordination engines act in tandem to support process-oriented interactions between applications.

BROKERING DECOMPOSABLE SERVICES

The SRM mediates interactions between clients and individual servers, while the Process Planner coordinates the execution of multiple, interdependent services. A third coordination service, called the Server Group, coordinates multiple, independent services, such as system-level planning tasks that decompose into subplanning tasks for independent subsystems, or decision support queries that reduce to subqueries to independent databases. The Server Group is implemented as a specialized subclass of the SRM Agent. The Server Group Agent inherits most of the SRM's control behavior, but selectively extends the SRM's service registration and request services. The registration extension enables developers to register composite services in the Server Group directory. A composite service entry contains pointers to functions for (1) decomposing that defined service into other concrete services that are registered with the Server Group, and (2) combining results for those services. In response to requests for a composite service, the Server Group uses these functions to transparently: decompose that service into constituent tasks; dispatch requests to the appropriate servers for concurrent execution; and collect and combine results from those servers into a single response for the client. Examples of combination functions to merge results include voting algorithms, logical union, intersection, and relational join operations.

CONCLUSIONS

The NetWorks! tool suite enables complex coordination behaviors to be modeled and executed external to independent decision support applications, through a supporting layered infrastructure for distributed computing. Current generic control services include request broker, workflow, and group-oriented coordination models. The resulting partitioning of application and distributed behaviors results in improved modularity, maintainability, and extensibility of individual applications, whether intelligent or conventional. The infrastructure is extensible at both the control service (i.e., Agent) and message-passing layers. Individual control services are also interoperable, which means that they can be combined much like building blocks to match application-specific coordination requirements. These tools are directly applicable to domains other than decision support, including operations support, process control, concurrent engineering, and office automation.

ACKNOWLEDGMENTS

NetWorks! technologies have been developed in part with funding from the Small Business Innovative Research Program by the U.S. Army (contract DAAB10-87-C-0053) and NASA (contracts NAS10-11606, NAS10-11882, NAS5-31920, and NAS8-399905). Kirsten Kissmeyer, Tom Trautz, and Craig Hughes contributed to the development of the Agent-based coordination models.

REFERENCES

[1] M. Fox and S. Smith. 1984. ISIS: A Knowledge-Based System for Factory Scheduling. *Expert Systems*. 1: 25-49.

[2] S. Minton. ed. 1993. *Machine Learning Methods for Planning*. San Mateo, CA.; Morgan Kaufmann.

[3] G. Coulouris and J. Dollimore. 1988. *Distributed Systems: Concepts and Design*. Reading, MA.; Addison-Wesley.

[4] R. M. Adler. 1992. Object-Oriented Tools for Distributed Computing. *NASA Proceedings for Technology 2002 Conference*, NASA CP-3189.

[5] R. M. Adler. 1993. *Distributed Coordination Models for Client-Server Computing*. Technical Report, Symbiotics, Inc. Cambridge, MA.

[6] Object Management Group and X/Open. 1991. The Common Object Request Broker: Architecture and Specification. OMG Document No. 91.12.1 (revision 1.1), OMG, Framingham, MA.

Planning and Scheduling Workshop: Mission Support