Coordinating Complex Problem-Solving Among Distributed Intelligent Agents

Richard M. Adler
Symbiotics, Inc.
875 Main Street
Cambridge, MA 02139

ABSTRACT

This paper describes a process-oriented control model for distributed problem-solving. The model coordinates the transfer and manipulation of information across independent networked applications, both intelligent and conventional. The model was implemented using SOCIAL, a set of object-oriented tools for distributed computing. Complex sequences of distributed tasks are specified in terms of high-level scripts. Scripts are executed by SOCIAL objects called Manager Agents, which realize an intelligent coordination model that routes individual tasks to suitable server applications across the network. These tools are illustrated in a prototype distributed system for decision support of ground operations for NASA's Space Shuttle fleet.

Keywords: distributed control, intelligent coordination, distributed artificial intelligence

INTRODUCTION

End-user tasks in distributed systems typically decompose into sequences of interactions between independent applications. For example, scheduling engines are often driven by task, resource, and constraint networks derived from independent planning systems. Scheduling a space mission may therefore depend on a succession of individual data transfers and manipulations across several decision support tools and databases. Similar task decompositions arise in operations support for complex control networks such as the Space Shuttle Launch Processing System (Adler, 1990).

Interactions among distributed applications and data stores must be initiated and managed. In the absence of direct interprocess links, human intervention is required to effect transfers of data and control. Such involvement, whether by end-users or supporting network operators, impacts the productivity and cost of frequent, high-level activities such as decision support. Moreover, the likelihood of human errors may compromise quality and safety.

Intelligent systems have the capacity to coordinate distributed problem-solving autonomously. However, considerable latitude exists in designing architectures for distributed intelligent control (Bond and Gasser, 1988). For example, interaction sequences can be automated piecemeal, by establishing directed, data-driven control links between individual applications. Distributing sequential control logic in this manner is cumbersome in application networks that address multiple complex tasks. Moreover, directed links are difficult to maintain, extend, and verify when network applications and tasks are added or modified with any frequency. Finally, highly distributed control schemes incur processing overhead to ensure focus and coherence of autonomous problem-solving activities.

This paper describes a process-oriented model for distributed coordination. The model enables complex sequences of distributed tasks to be specified in terms of high-level scripts. Each script element represents a distinct data transfer task or request for problem-solving skills between complementary applications. The model also encompasses intelligent control modules that execute these process scripts automatically: individual tasks are routed to suitable distributed servers and results

are retrieved for the requesting applications. This model alleviates many of the difficulties faced by more decentralized coordination schemes.

The new process control model was implemented as an extension to SOCIAL, a development tool for distributed computing across heterogeneous hardware and software environments. The next section of the paper reviews SOCIAL's architecture and functionality. The following section describes the design and implementation of the process control model. The model is then illustrated with a prototype distributed system for decision support of Space Shuttle fleet ground operations at the NASA Kennedy Space Center.

OVERVIEW OF SOCIAL

SOCIAL consists of a layered collection of object-oriented tools for distributed communication, data management, and control (cf. Figure 1). These generic capabilities are bundled into active objects called *Agents*. SOCIAL provides an extensible library of predefined Agent classes with specialized integration and coordination behaviors. An application is linked non-intrusively to an Agent via calls to a high-level Application Programming Interface (API). Applications make API calls to invoke their mediating Agent objects to execute desired distributed behaviors. Agents interact using asynchronous message-passing. SOCIAL's underlying layers transparently manage interprocess message communication across heterogeneous languages, operating systems, and networked hardware platforms (Symbiotics, 1990).

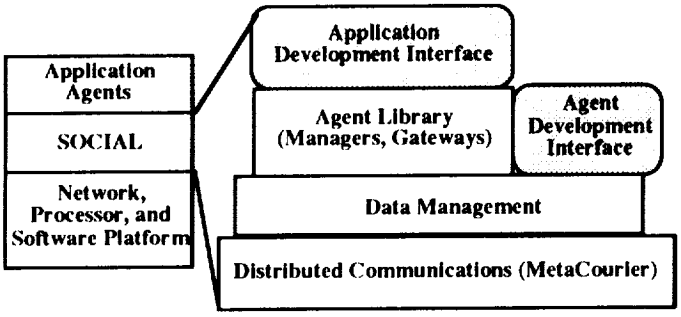

Figure 1: Architecture of SOCIAL

SOCIAL Gateway Agents

SOCIAL Gateway Agents provide a uniform design model and methodology for integrating heterogeneous applications, both conventional and intelligent (Adler, 1991b). The root Gateway Agent class defines a full peer-to-peer control model that is inherited by all specialized Gateway subclasses. This model invokes a set of Agent methods in a data-driven manner to process: (a) outgoing messages from the Gateway's associated application to other Agents; (b) incoming messages from other Agents; and (c) responses to prior outgoing messages.

An application is integrated by creating a new Gateway subclass, which involves specializing two sets of Agent methods. One set establishes custom mappings for application-specific data models and control interfaces. To simplify interactions between heterogeneous applications, SOCIAL transports information in a neutral exchange format. Accordingly, each Gateway subclass must define conversion methods for translating from the application's native knowledge representation model and command interface into SOCIAL's neutral data format, and vice versa. Native and neutral exchange data structures are accessed and manipulated using functions from the application's programmatic interface and the API for SOCIAL's data management layer.

The second set of Gateway methods defines the application's desired interactions with other elements of the distributed system. These control methods are constructed using the Gateway conversion methods for extracting data and knowledge, injecting information, or invoking application commands, as required. One method establishes ***server*** behaviors, which process incoming messages from other application Gateways and generate suitable responses. A second method defines ***client*** behaviors. Applications configured as clients initiate outgoing messages containing service requests via their Gateways. A Gateway client behavior typically injects responses to previous request messages back into the associated application for follow-on processing. A given application Gateway can support multiple client and server interactions with any number of other application Agents.

SOCIAL Manager Agents

SOCIAL Manager Agents provide predefined control models for coordinating activities in complex networks of application Agents. Coordination among distributed problem-solvers can be achieved through different strategies. One approach is to distribute control, localizing it within individual applications. A second approach is to centralize control, either in a preferred application or in a dedicated, independent module. SOCIAL Gateway Agents provide flexible vehicles for implementing either of these opposing alternatives. Manager Agents were developed to support a third design strategy, which is to combine localized and centralized control into *hybrid* coordination architectures.

The first SOCIAL Manager Agent defined a hierarchical, distributed control (HDC) model (Adler, 1991a). This HDC-Manager mediates interactions among autonomous "subordinate" Agents much like a human manager. Application Gateways communicate exclusively with their Manager Agent, requesting information or problem-solving resources, and receiving responses to those requests. Subordinate Agents do not need to know about the functionality, structure, or even the existence of other application Agents; they only need to know (a) the high-level API for interacting with the HDC-Manager and (b) the names of the services available within the HDC-Manager's scope.

The basic operational model for the HDC-Manager is summarized in Figure 2. The HDC-Manager functions as an intelligent router of task requests, based on a directory knowledge base. This directory describes: (a) individual information resources and problem-solving capabilities; (b) the application Agent that supports each such service; (c) the message format for requesting that service; and (d) the server Agent's logical address. Request messages from application Gateway Agents are posted to the HDC-Manager's agenda queue. The HDC-Manager processes and dispatches requests asynchronously to suitable server application Gateways. These Agents, in turn, post responses from their applications to the HDC-Manager's "bulletin-board" database. The HDC-Manager subsequently retrieves such responses and forwards them back to the original requesting Agents.

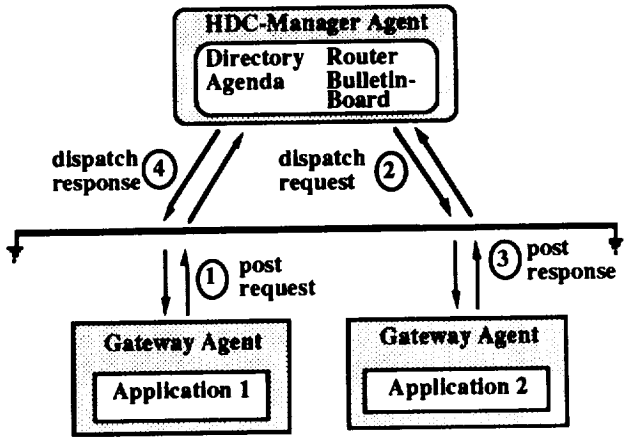

Figure 2: HDC-Manager Operational Model

In essence, the HDC-Manager establishes a layer of control abstraction that decouples application Gateways from direct connections with one another. The centralized directory promotes maintainability and extensibility over the evolutionary lifecycle of complex distributed systems.

SOCIAL'S PROCESS-PLANNER AGENT

SOCIAL's HDC-Manager Agent supports simple interactions between independent distributed systems. For example, an intelligent scheduling tool might query a remote shop floor production database to determine the availability of equipment or labor resources. Similarly, an intelligent operations support application for a power management system might collect data to confirm a power bus fault hypothesis, or command an experiment management system to minimize power consumption.

The applications in these examples are loosely coupled. The scheduler uses the database solely as a source of current status information about its target domain. Similarly, operation management systems only interact in situations where the structural and functional interfaces between their target subsystems appear to be relevant. Simple *discrete* transactions (e.g., query-response, sensor polling, command-acknowledgment interchanges), provide

sufficient coupling to enable distributed problem-solving activities in these contexts. The HDC-Manager Agent contains all of the apparatus and control functions required to coordinate discrete transactions within a distributed system.

However, many kinds of distributed problem-solving activities cannot be accomplished within the scope of an individual logical transaction between two remote applications. Consider, for example, a distributed decision support system composed of two or more independent tools, such as a planning system and a scheduling engine. Suppose that the planning system incorporates the master database for all decision support information, including all operational plans and schedules for the target domain. Assume also that the models used to represent data and knowledge are incompatible across the two tools, which is common for independent systems specialized to solve different problems.

In this context, an elaborate set of information and control exchanges has to take place to perform scheduling. Data must be extracted from the planning system's database, transferred to the scheduler, translated into a format that is compatible with the scheduler, and then loaded. At this point, the primary scheduling activity itself can proceed. Once scheduling has been completed, a similar set of support transactions must be accomplished in reverse order. The completed schedule must be translated into a format acceptable to the planner, transferred back to the planner's host platform, and incorporated into the master decision support database.

Clearly, such sequences need to be automated, both for end-users and for autonomous management systems. It should be possible to invoke scheduling or comparable functions through simple, high-level commands. Such commands should specify only the few data items that are required to characterize particular instances of the desired task type (e.g., a mission identifier, options to override default control parameters for the scheduling engine). This amounts to a requirement to automate ***composite*** activities, or ***processes***, composed of multiple discrete interactions between independent distributed applications. The HDC-Manager Agent currently lacks the requisite capabilities either to define such distributed problem-solving processes or to coordinate their execution. We considered several design approaches to extend SOCIAL to provide the desired functionality.

One alternative would be to configure a distributed system so that one message would initiate the desired activity sequence by triggering the first application Gateway to perform its assigned task, pass the results onto a second Gateway to perform the second required task, and so forth. In other words, a single message to the first server Agent would automatically initiate the desired chain of interactions. SOCIAL's communication layer maintains a "travel log" for each message as it is passed through Agents. Once the "terminal" Agent completes its activities, results are automatically returned and post-processed through all preceding Agents that appear in the message's log.

Unfortunately, the logic for parsing and forwarding messages within individual application Gateway Agents can become quite involved for complex processes. Moreover, a given application Agent may have to perform a given function within multiple process sequences, with different successor Agents and post-processing activities for each distinct chain. Maintainability and extensibility are compromised in that each time a new process is defined, the control logic for Gateway Agents in that process chain must be modified. Consequently, for mission critical applications, the entire suite of behaviors for each affected Agent has to be verified again. Finally, SOCIAL's communication model only supports ***acyclic*** message forwarding, precluding processes involving "back-and-forth" exchanges or iterative looping.

A second alternative would be to extend the HDC-Manager directly to support the specification and execution of distributed sequential processes. On this strategy, special "macro" tasks would be definable in the HDC-Manager's directory knowledge base, corresponding to composite processes. A message requesting execution of such a composite task would activate extended control logic.

This logic would decompose macro tasks into constituent service requests and post individual steps to the task agenda, in suitable order, for the HDC-Manager to process and route.

This approach resolves the objections raised against the previous design strategy. First, messages are only passed between the extended HDC-Manager and individual application Agents, eliminating the hardwiring of interaction sequences directly into the control logic for individual Gateways. Second, the new macro processes are modular, maintainable, and readily extensible. In particular, processes are modeled independent from and external to individual application Agents. System testing is simplified because new processes can now be defined without affecting previously verified processes and Agent behaviors. Finally, the extended HDC-Manager mediates all interactions between application Gateways as separate message transactions. SOCIAL's acyclic message-passing model can accommodate cyclic behaviors that are broken up in this manner.

The main objections to extending the HDC-Manager are performance and complexity. This Agent's primary design role is to eliminate direct connections between application Agents by mediating interactions. The proposed functional extensions decompose macro tasks and manage queueing of process subtasks. These capabilities for managing distributed processes impose computational overheads that reduce the responsiveness of this core routing capability. Moreover, these design extensions also complicate the original control logic of the HDC-Manager significantly.

We adopted a third approach, which distributes the functionality of the extended HDC-Manager to overcome these design problems. Specifically, centralized process definition and management functions are retained, but *decoupled* from the HDC-Manager and assigned to a new subclass of Manager Agents called Process-Planners. The distributed control model realized in the Process-Planner is then configured to *drive* the HDC-Manager through the individual process steps comprising composite activity sequences. It does this by posting successive service requests to the HDC-Manager's agenda for distributed routing. This basic architectural configuration is depicted in Figure 3.

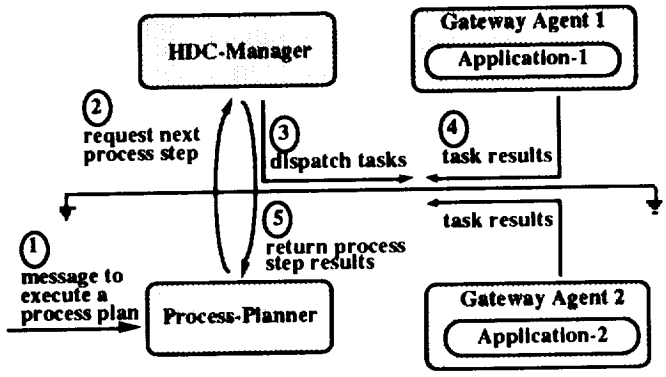

Figure 3: SOCIAL Process Control Architecture

This distributed design is attractive because it enables the HDC-Manager to function identically for two distinct distributed computing models – transaction-based and process-based. The Process-Planner manages process decomposition and activity sequencing. It transmits individual process steps as individual service requests to the HDC-Manager, replicating the type of inputs that would be expected from ordinary application Gateway Agents. Consequently, the HDC-Manager need not distinguish between discrete and composite service requests within its agenda. In fact, process steps and service requests representing discrete Gateway transactions can be interleaved on the HDC-Manager's agenda, enabling both kinds of interactions to be coordinated concurrently. In addition, partitioning distributed control logic across two Manager Agents fosters modularity, maintainability, and extensibility.

The message traffic between the Process Planner and HDC-Manager entails some performance overhead. However, the two Agents communicate asynchronously and can operate concurrently on dedicated processors, compensating at least in part for message-passing overhead. Overall performance depends strongly on the particular distributed system and its ratio of communication and coordination to local application problem-solving loads.

Implementing the Process-Planner Agent

The Process-Planner Agent was implemented as a subclass of SOCIAL Gateways. Consequently, it inherits the standard Gateway peer-to-peer control model and API methods. These methods were specialized to interface with the core process planning application. The data injection API method parses two SOCIAL neutral exchange types. Character strings are interpreted as pathnames for files containing process scripts, which the planning system loads into memory. Lists are treated as commands and command arguments, which are executed through the application's control interface (e.g., initialize, reset). Extensions to handle other data types amount to straightforward program Case statement clauses. The API method for extracting information was not required, because the process planning system functions solely in a client role.

The process planning application examines a process script to determine the next step to perform. Currently, a script consists of an ordered list of entries that correspond to services identified in the directory knowledge base of the associated HDC-Manager Agent. The :compute-next-step command retrieves the first script item that has not been instantiated. An item has been instantiated if it has been annotated with execution results returned from the HDC-Manager.

The planning program also computes predecessors and successors to current steps in process scripts. The HDC-Manager supports a generic file transfer service based on SOCIAL utility agents that send and receive files across network nodes. This service is context-sensitive in that it presupposes source and target file pathnames and host names. The HDC-Manager can determine these items given the previous and succeeding script steps to the current file transfer task.

The Process-Planner Agent starts up the embedded planning program in response to a message that specifies initialize and reset commands, together with the name of a script file to load. A second message initiates the following control cycle:

1. the Agent determines the next process step from the process planner program and dispatches a suitable service request message to the HDC-Manager;
2. the HDC-Manager dequeues the request from its agenda, finds the server application Agent (e.g., a Gateway for a scheduling engine, a Send-File utility), and dispatches an appropriate task message to that Agent;
3. the target application Agent performs the assigned task and posts its response to the HDC-Manager's bulletin-board. Typically, the target Agent is a Gateway, which interacts with its embedded application by injecting data or commands and collecting query or problem-solving results;
4. the HDC-Manager automatically routes the posted task results back to the Process-Planner Agent;
5. the planning program updates the instantiated script with service request results and computes the next step from the script. The Process-Planner dispatches this new process step back to the HDC-Manager for routing.

The Process-Planner then reiterates this execution cycle. The Agent terminates looping when notified by its embedded process planning program that the script has been completely instantiated.

The Process-Planner plays the role of a Manager Agent in that it performs distributed control functions rather than integrating domain-specific applications. However, it was designed and implemented as a subclass of Gateway Agents. Sophisticated process planning tools are beginning to appear commercially in CAE, CAM, and CASE domains. These tools are used to specify task decompositions and automate control of work flows for machining complex parts, other manufacturing processes or managing large projects. The Gateway's uniform, high-level interface architecture preserves design flexibility to replace the SOCIAL process planning program with a more powerful dedicated engine.

Distributed Decision Support Prototype

A prototype was developed to validate this design model for coordinating processes in SOCIAL. This prototype simulates a distributed decision support system for ground operations activities for the Space Shuttle fleet at the NASA Kennedy Space Center. Specifically, a Process-Planner Agent was implemented and coupled to an HDC-Manager. These two Agents automatically coordinate the complex sequence of distributed activities required to schedule Shuttle missions.

Two (simulated) decision support applications were integrated using SOCIAL Gateway Agents (cf. Figure 4). One Agent represents a commercial planning system called Artemis, which NASA has modified with a frontend interface customized for planning ground support activities for Shuttle missions. The second Agent represents an intelligent, constraint-based scheduling engine called Gerry (Zweben, 1990), which is being developed by the NASA Ames Research Center. Artemis is based on a proprietary fourth generation language and resides on an IBM mainframe host. Gerry, written in Common Lisp and CLOS, runs on Unix workstations.

The Artemis Gateway Agent is configured to simulate three tasks: (a) downloading data files for a particular mission from the Artemis master planning database; (b) uploading data files representing a completed mission processing schedule back into Artemis; and (c) running an analysis program to detect and report resource conflicts between the new schedule and existing schedules for other Shuttle missions. The Gerry Gateway also simulates three tasks: (a) translating and loading mission plan files into the scheduler; (b) computing the mission schedule; and (c) extracting and translating the completed schedule back into Artemis-compatible file format.

The Gerry scheduler requires four types of plan information: a network of tasks to be performed to prepare the Shuttle vehicle and its associated payload(s) for launch; a specification of available resources (e.g., labor schedules, equipment such as cranes, and other materials); a set of constraints on tasks and resources; and a data dictionary that describes the information fields in the preceding three datasets. Artemis generates these datasets as four ASCII files in a standardized record format. Gerry requires data to be input from ASCII files in a custom object-oriented format.

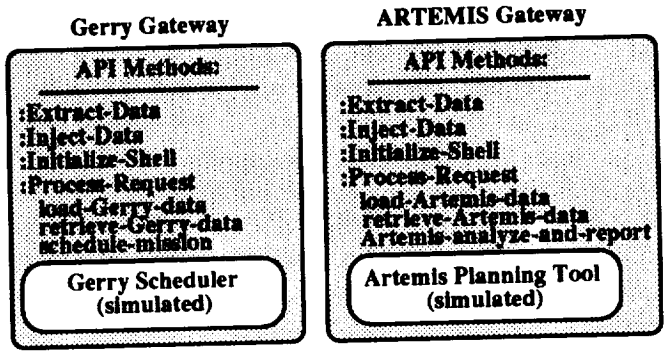

Figure 4: Prototype Decision Support Gateways

SOCIAL's data management subsystem was used to define custom neutral exchange data structures. Translators were written to map between Artemis and SOCIAL data models and between Gerry and SOCIAL data models (cf. Figure 5). The translators were hooked into the application Gateway Agent interface API methods to perform appropriate conversions of data file formats. Data files are translated into neutral exchange format structures in memory, and then written to new files in the target converted format.

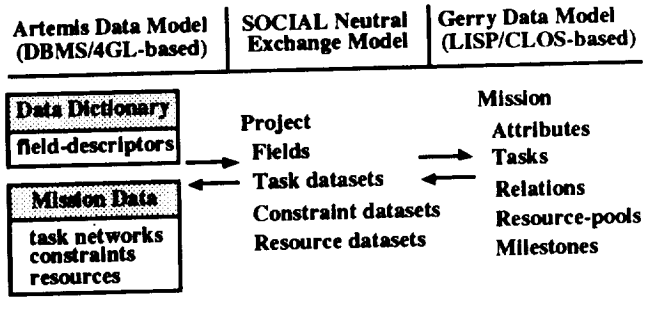

Figure 5: Disparate Decision Support Data Models

A subclass of HDC-Manager Agent, called the DSS-MGR, was created to coordinate interactions

between the two decision support applications (cf. Figure 6). Three steps were required to specialize the DSS-MGR Agent for this purpose. First, conditions were defined for prioritizing agenda service requests. The DSS-MGR sorts requests with respect to an ordinal list of service types. Requests of the same type are ordered by increasing values of a numeric priority attribute. Second, the DSS-MGR directory knowledge base was constructed. The directory identifies all services available from all application Gateways subordinate to the DSS-MGR, plus the generic file transfer capability. Both DSS-MGR attributes are defined using the high-level declarative API specific to HDC-Manager Agents. Third, dispatching functions were written for each directory service entry. These functions manipulate data arguments contained in service request messages into a task message that the HDC-Manager routes to the relevant application Gateway server.

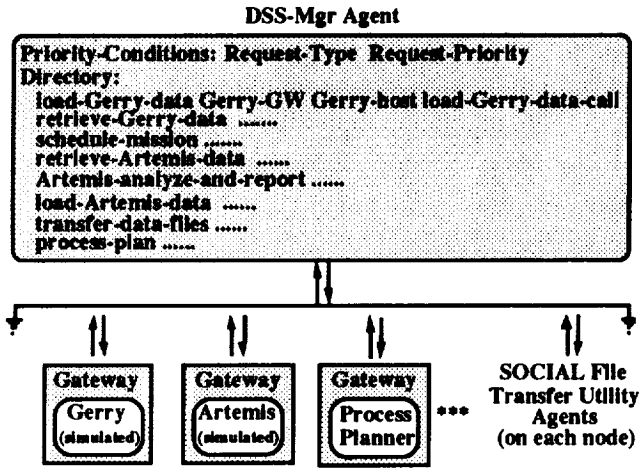

Figure 6: Decision Support HDC-Manager Agent

Next, a script was written for the Process-Planner Agent, defining the distributed process for scheduling Shuttle missions. This sequence consists of the following steps:

```
(retrieve-Artemis-data)
(transfer-data-files)
(load-Gerry-data)
(schedule-mission)
(retrieve-Gerry-data)
(transfer-data-files)
(load-Artemis-data)
(Artemis-analyze-and-report)
```

File format conversions currently take place within the load-Gerry-data and retrieve-Gerry-data tasks. Once the various Agents are loaded and initialized, the mission scheduling sequence is initiated through a simple message to the Process-Planner Agent to compute the next step for a particular mission, such as STS-40. The Process-Planner Agent then executes the control loop described in the previous section against the mission scheduling script.

All demonstration Agents and simulated applications were written in Common Lisp. The demonstration system can be run on a single platform or combination of platforms that currently run SOCIAL/Lisp, including Apple Macintosh IIs, Lisp Machines, and Unix workstations. The Agent Library is currently being converted to C, to run on SOCIAL/C workstation hosts. A planned port of SOCIAL/C to the IBM/VM environment will establish direct interprocess interfaces across mainframes and workstations. NASA's distributed decision support system can then be implemented on the intended target platforms.

FUTURE DIRECTIONS

The Process-Planner Agent is being redesigned with extended functionality. The original planning program only supports simple sequential scripts. These scripts cannot specify data-driven processes, in which successive steps are determined dynamically at runtime, contingent on the results of preceding process steps. Moreover, the initial Process-Planner drives the HDC-Manager to execute script steps individually, in a strictly synchronous, "execute and wait" sequence. Ideally, the Process-Planner should be able to request the HDC-Manager to route all script activities that are mutually independent within a single control cycle. To overcome these limitations, a more expressive scripting language will be developed for specifying processes that incorporate conditional branching, iteration, and concurrent tasking. The Process-

Planner's control logic will be extended accordingly. A capability for executing multiple scripts simultaneously will also be added.

A second set of enhancements will provide more formal development tools for creating and managing script libraries, replacing the ad hoc techniques used in the prototype Process-Planner. A menu-based editor will be developed to access and manipulate process scripts. Also, scripts will be stored in a central database of process plans rather than in an arbitrary collection of independent files.

Other development efforts will extend SOCIAL's library of Manager Agents. The current HDC-Manager is adequate for distributed systems in which a single application Agent represents the unique source for a given resource or service. However, additional control requirements arise for application networks in which ***multiple*** application Agents can provide data, knowledge, or problem-solving skills redundantly. For example, identical copies of a program may be available on several nodes. In addition, some applications may have overlapping functionality for planning, scheduling, or other tasks. A dedicated "Server-Group" Agent, inspired by the ISIS model for group-based tasking (Birman, 1990), is being designed to address distributed control issues for functionally redundant application networks. Like the Process-Planner, this Agent will be configured to offload new control capabilities and work cooperatively with the HDC-Manager. Specifically, the Server-Group will monitor availability of server Agents, determine the best server for a task, and enable redundancy-based approaches to fault tolerance.

RELATED WORK

Alternative frameworks for developing heterogeneous, distributed intelligent systems include ABE (Hayes-Roth, 1988), MACE (Gasser, 1987), Agora (Bisiani, 1987), and Cronus (Schantz, 1986). MACE incorporates dedicated manager agents for centralized routing of messages among application agents. However, MACE managers lack the other capabilities of SOCIAL HDC-Managers, such as shared memory, transparent returning of message responses, and extensibility for multi-level control hierarchies. Like SOCIAL, ABE, Agora, and Cronus all provide virtual environments to shield users from platform dependencies and networking mechanics. However, they do not implement generic distributed services in uniform, object-oriented layers that are accessible to developers for customizing. Agora relies on communication through shared-memory, reflecting its orientation towards parallel multi-processing architectures. The other tools use message-passing models comparable to SOCIAL. ABE and Agora provide predefined control frameworks such as data flow and blackboard models. Unlike SOCIAL Manager Agents, these models explicitly couple individual applications directly to one another. Moreover, heterogeneous SOCIAL Manager Agents can be configured to work together cooperatively. It is unclear whether the other tools support such combinations within a given distributed systems.

The literature on distributed artificial intelligence (DAI) contains many interesting architectures for cooperative problem-solving, including blackboard systems, contract nets, and collections of autonomous agents (Bond and Gasser, 1988). In this context, cooperation refers to loosely-coupled networks of intelligent Agents working to solve a single complex problem through collective action. Most such DAI architectures rely on purely localized control models duplicated across homogeneous, autonomous agents. These designs can be replicated within the generic communication and control model provided by SOCIAL Gateway Agents.

More recent DAI research focuses on theories of cooperation for open-ended systems composed of arbitrarily heterogeneous applications (Gasser and Huhns, 1989). The critical problem here is to design dynamic interaction protocols for communicating self-descriptive goals, plans, and intentions among agents with radically different knowledge and perspectives. SOCIAL Managers currently address more modest closed-world domains, in which the resources available in an agent network are specified ***a priori*** and statically. A synthesis of Manager control models with dynamic interaction pro-

tocols could contribute to a powerful theory of cooperation for open networks of autonomous agents: agents and their resources could be registered dynamically in the context of a partially centralized control architecture that mediates agent interactions.

CONCLUSIONS

SOCIAL applies a highly modular, ***non-intrusive*** object-oriented approach to simplify the design and implementation of complex distributed systems (cf. Figure 7). High-level Agent APIs partition generic distributed computing and application-specific functionality. Gateway Agents provide a uniform methodology and design architecture for integrating heterogeneous applications, both intelligent and conventional. Manager Agents provide high-level distributed control building blocks for tying application Gateways together. Coordinating via Managers eliminates direct connections between individual application Agents that are difficult to maintain and extend.

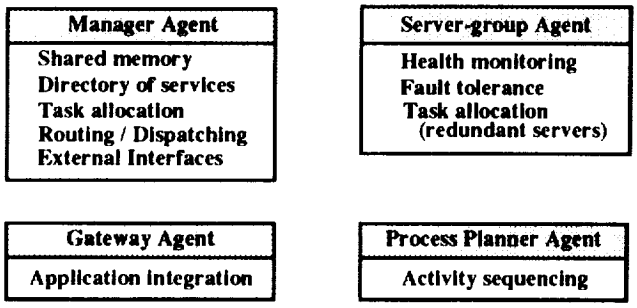

Figure 7: SOCIAL Library Building Blocks

The prototype distributed decision support system described earlier illustrates capabilities for:

- integrating independent planning and scheduling engines across a computer network;
- automating distributed interprocess communication, namely "fine-grained" exchanges of data and control between remote executing applications;
- automating conventional "coarse-grained" interactions such as file transfers across distributed application platforms;
- automating the coordination of complex sequences of fine- and coarse-grained interactions between distributed applications through high-level, declarative scripts.

The coordination capabilities provided by SOCIAL Manager Agents have broad applicability for distributed intelligent systems in space-related domains. For example, process scripts could be used to coordinate routine shop floor activities. Task control and work-in-progress status data could be routed automatically among Shuttle and payload processing facilities scattered across the Kennedy Space Center complex. Similarly, shop floor statistics could be collected, summarized, and transmitted to higher-level decision support systems. Mission schedules could be monitored and managed more effectively. This feedback could also be used to tune the processing estimates that drive long-term planning of Shuttle missions.

In addition, process scripts could be used to automate standardized launch processing and mission control disciplines, enhancing productivity, safety, and quality assurance. Beyond decision and operations support applications, process scripts can be used to automate routine flows of information in office automation and concurrent engineering contexts. Finally, process scripts can be applied in space science domains for automating sequences of data retrieval, analysis, and graphic visualization activities. End-users could develop, maintain, and extend their own application-specific scripts.

Acknowledgments

Development of SOCIAL has been sponsored by the NASA Kennedy Space Center under contracts NAS10-11606 and NAS10-11763. Artemis is a trademark of Metier Management Systems. Monte Zweben and Bob Gargan provided Gerry software. Brad Young provided technical assistance relating to Artemis.

REFERENCES

Adler, R.M. (1991a). A Hierarchical Distributed Control Model for Coordinating Intelligent Systems. *Proceedings of the 1991 Goddard Conference on Space Applications of Artificial Intelligence.* NASA CP-3103. pp. 183-198.

Adler, R.M. (1991b). Integrating CLIPS Applications into Heterogeneous Distributed Systems. *Proceedings of the Second CLIPS Users Conference.* NASA Johnson Space Center. Houston, Texas, September 23-25, 1991.

Adler, R.M. and Cottman, B.H. (1990). EXODUS: Integrating Intelligent Systems for Launch Operations Support. *Fourth Annual Workshop on Space Operations, Applications, and Research (SOAR-90).* NASA CP-3103, Volume 1. pp. 324-330.

Birman, K., Joseph, T., Kane, K., and Schmuck, F. (1989). *The ISIS System Manual V1.2.* Department of Computer Science, Cornell University, Ithaca, New York.

Bisiani, R., Alleva, F., Forin, A., Lerner, R., and Bauer, M. (1987). The Architecture of the Agora Environment. In M. Huhns (Ed.). *Distributed Artificial Intelligence.* Morgan-Kaufmann, San Mateo, California.

Bond, A.H., and Gasser, L. (1988). (Eds.) *Readings in Distributed Artificial Intelligence.* Morgan-Kaufmann, San Mateo, California.

Durfee, E.H., Lesser, V.R., and Corkill, D.D. (1989). Trends in Cooperative Distributed Problem Solving. *IEEE Transactions on Knowledge and Data Engineering.* 1(1), 63-83.

Gasser, L. and Huhns, M.N. (1989). (Eds.). *Distributed Artificial Intelligence, Vol. II.* Morgan-Kaufmann, San Mateo, California.

Gasser, L., Braganza, C., and Herman, N. (1987). MACE: A Flexible Testbed for Distributed AI Research. In M. Huhns (Ed.). *Distributed Artificial Intelligence.* Morgan-Kaufmann, San Mateo, California.

Hayes-Roth, F., Erman, L.D., Fouse, S., Lark, J.S., and Davidson, J. (1988). ABE: A Cooperative Operating System and Development Environment. In A.H. Bond and L. Gasser (Eds.). *Readings in Distributed Artificial Intelligence.* Morgan-Kaufmann, San Mateo, California.

Schantz, R., Thomas, R. and Bono, G. (1986). The Architecture of the Cronus Distributed Operating System. *Proceedings of the 6th International Conference on Distributed Computing Systems.*

Symbiotics, Inc. (1990, March). *Object-Oriented Heterogeneous Distributed Computing with Meta-Courier.* Technical Report, Cambridge, Massachusetts.

Zweben, M. and Gargan, R. (1990). The Ames-Lockheed Orbiter Processing Scheduling System. *Fourth Annual Workshop on Space Operations, Applications, and Research (SOAR-90).* NASA CP-3103, vol. 1. pp. 290-295.