N90-27301

A Development Framework for Artificial Intelligence Based Distributed Operations Support Systems

Richard M. Adler and Bruce H. Cottman
Symbiotics, Incorporated
875 Main Street
Cambridge, Ma 02139
(617) 876-3633

Abstract

Advanced automation is required to reduce costly human operations support requirements for complex space-based and ground control systems. Existing knowledge-based technologies have been used successfully to automate individual operations tasks. Considerably less progress has been made in integrating and coordinating ***multiple*** operations applications for ***unified*** intelligent support systems. To fill this gap, we are constructing SOCIAL, a tool set for developing Distributed Artificial Intelligence (DAI) systems. SOCIAL consists of three primary language-based components defining: models of interprocess communication across heterogeneous platforms; models for interprocess coordination, concurrency control, and fault management; and for accessing heterogeneous information resources. DAI application subsystems, either new or existing, will access these distributed services non-intrusively, via high-level message-based protocols. SOCIAL will reduce the complexity of distributed communications, control, and integration, enabling developers to concentrate on the design and functionality of the target DAI system itself.

Introduction

Operational support of complex space-related systems currently entails expensive manpower requirements. Human labor costs are particularly high in manned space systems such as the Space Shuttle and the planned Space Station: in these remote settings, scarce manpower that is dedicated to operational support cannot be allocated to primary mission objectives. The economic viability of increasingly advanced space systems hinges on significant increases in operational support automation [Ba88].

Standard engineering formalisms such as control theory and operations research can be used to automate simple control, monitoring, and scheduling tasks. However, such methods do not generalize readily to non-routine contexts: assessing and responding to system failures; revising plans in the face of unforeseen conditions; and similarly difficult cognitive tasks. Over the last several decades, artificial intelligence (AI) researchers have addressed these problems by developing symbolic modeling and automated reasoning techniques. These methods offer superior flexibility and generality for modeling human analytic and decision-making processes and for solving combinatorially complex problems.

Expert systems, model-based reasoning, and other knowledge-based tools and methods have been applied to automate tasks including fault detection and diagnosis, planning and scheduling, data analysis, and information storage and retrieval. Several important prototypes systems developed in recent years are being extended and validated in field tests, in preparation for integration into existing operational support systems for complex networks [Ad89b,Br89,Ba88,Mu89,Ru88].

Integrating and coordinating ***multiple*** knowledge-based applications related to a common domain are critical problems that have received little attention until recently [Ad89a]. Existing intelligent applications for operations support rely on system-specific interfaces to users, data feeds, databases, and conventional automation software. These "standalone" systems also lack access and control facilities for working together cooperatively on clearly related operations tasks, such as intelligent diagnosis and error-tracking. As increasing numbers of intelligent support tools are deployed together in common domains, the need for effective tools for integrating such systems into a unified cooperative framework will become critical.

This paper describes SOCIAL, a development framework for distributed systems that is intended to fill the technology gap. SOCIAL consists of three primary language-based tools: MetaCourier supplies functionality for interprocess communication and control access across heterogeneous platforms; MetaAgents defines control models for interprocess organization, data replication, concurrency management, and fault detection and recovery; MetaViews defines a uniform data model for accessing and controlling persistent information stores such as data and knowledge bases. New and existing application elements access these distributed services non–intrusively, via high–level message–based protocols. SOCIAL thereby reduces the complexity of distributed communications, control, and integration, enabling developers to concentrate on the design and functionality of the target system itself.

The next two sections of the paper define the central system integration issues that SOCIAL addresses and review related research. Next, SOCIAL's architecture and user model are described and illustrated with a hypothetical operations support example. The remaining sections outline the design and functionality of SOCIAL's primary language–based subsystems.

Integrating and Coordinating Heterogeneous Intelligent Systems

Several basic problems arise in integrating and coordinating multiple knowledge-based systems related to a common domain. First, different activities within a domain such as operational support generally depend on different kinds of knowledge, skills, tools, and methodologies. Knowledge-based automated assistants tend to require correspondingly diverse representation, reasoning, and internal control models. Integrating such applications thus requires methods for reconciling or accommodating heterogeneous internal architectures.

Second, different tasks within a given domain, while distinctive in many respects, frequently display important commonalities. For example, network operators and managers share background information and expertise concerning configuration procedures, although their respective depth and application of such knowledge may differ. A framework for integrating multiple intelligent applications in a given domain must facilitate sharing of knowledge resources, including symbolic models of domain structures, behavior, and operational expertise. Other resources of common utility across applications include interfaces to: users; databases; target system data feeds and command/control effecters; and conventional software for data analysis, performance monitoring, and (low–level) automated process control and safing systems.

Third, the integration strategy must be non–intrusive. Existing "standalone" knowledge–based and conventional programs and data resources represent significant investments in capital equipment, software development, and safety (i.e., from prior validation and verification). It would be prohibitively expensive to discard such resources or to re–engineer them extensively.

Fourth, applications and resources are generally distributed across heterogeneous software and hardware platforms connected by one or more (local area) networks. A generalized communications capability is needed for data exchange and control access across intelligent applications. Moreover, this functional capability should be accessible through a modular, high–level interface: minimizing the visibility of the mechanics of distributed communication fosters maintainability of application code and accessibility for developers unversed in exotic communication protocols.

The final and perhaps most critical problem is establishing cooperation between knowledge-based applications once they are integrated into a unified framework. Coordination presupposes that applications somehow know about one another, their respective capabilities, activities, intentions, and needs. In addition, coordination also presupposes control and communications models for exchanging requests, commands, suggestions, beliefs, and other information. Again, to facilitate maintainability and extensibility, it is important that application models and interprocess control mechanisms be partitioned from one another *and* from distributed communication functionality.

Related Work

Distributed Artificial Intelligence (DAI) deals with the solution of complex problems by networks of autonomous, cooperating computational processes [Hu87]. These processes, often called agents, can be distributed physically across computational resources and logically across an organizational structure. Typically, cooperation is mediated by message-passing communication between agents.

DAI research to date, has focused almost exclusively on domains in which single organizations of agents cooperate to solve *single* complex tasks [Bo88], including data fusion [Le83] and speech understanding [Bs87]. These "single problem" DAI research efforts have concentrated on developing complex *local* control structures for coordinating a network of homogeneous agents to converge to globally consistent problem solutions. For example, intelligent schedulers prioritize local agent tasks for execution according to heuristics or metrics that gauge probable global problem-solving effectiveness [Le83]. More complex planners create, order, and filter agent tasks adaptively, based on hierarchies of local and global problem-solving goals [Ha86].

Single problem DAI architectures, while suggestive, are not directly applicable to the integration problems described above. DAI research has generally assumed: a single logical organization of homogeneous complex agents, such as distributed blackboards; correspondingly uniform models for intra- and inter-agent communication and control; and homogeneous software and hardware platforms [Hu87,Bo88,Ja89]. All three assumptions are violated in the complex DAI environments of interest here.

Recently, DAI research has broadened to consider domains such as operations support and battle management, which encompass collections of related problems of varying complexity. While requisite problem-solving skills, knowledge and data resources may overlap considerably, the solutions to problems in these domains may be independent or only weakly dependent upon one another. These characteristics favor coarser-grained, more loosely-coupled DAI architectures, comprised of individual agents and organizations of agents that focus on particular problems or problem sets disjoint from one another. A useful human analogy is a legal or medical practice of consultants with different areas of specialized expertise.

Fine-grained scheduling and planning of inter-organization activities tend not to be critical issues in these domains because agent organizations only depend weakly on one another. Instead, the critical design issues are: (a) to integrate agents and agent organizations bounded by different knowledge representation, reasoning, control, and communication models; and (b) to access and integrate existing conventional software and data resources.

Initial "multiple problem" DAI applications have failed to address all of the issues raised in the last section in a generalizable manner. For example, KB-BATMAN integrates three intelligent decision aids for a military tactical command. However, the subsystems only communicate indirectly, through pairwise interactions with a shared relational data base and in a fixed, predefined control pattern [Nu88].

OPERA assists in operations support for NASA's Space Shuttle Launch Processing System [He87,Ad89b]. Its hierarchical blackboard architecture successfully integrates and coordinates heterogeneous expert systems, which share external interfaces and knowledge bases [Ad89c]. However, OPERA applications are all co-resident (i.e., physically *non*-distributed). Knowledge bases are restricted to a common representational model. OPERA also lacks generalized tools for handling errors and accessing data feeds or databases.

Several DAI development tools support integration of intelligent applications with heterogeneous organizational models. ABE and AGORA provide predefined models for inter-organizational control (e.g., dataflow, blackboard, transaction-based) [Bs87,Ha88]. ABE also supplies a high-level graphic editor and an interface to a commercial relational database management system. AGORA uses a shared-memory communication model, while ABE uses message-passing. Both tools employ virtual machine models that map onto particular platforms and network communication services (e.g., MACH, Chaosnet). MACE [Ga86], a message-based DAI testbed incorporates an elegant declarative language for modeling agents' roles, skills, goals, and acquaintances. However, MACE offers limited tools for coordinating multiple agent organizations and lacks support for heterogeneous processing platforms.

Architecture of the SOCIAL DAI Development Framework

SOCIAL is a generalized framework for developing both single and multiple problem DAI applications. Its architecture, shown in Figure 1, consists of a layered, partitioned set of system building blocks and development interfaces.

Developers use the high-level Application Interface to access predefined object classes, called ***Types***. Each Type represents a different, generic DAI control skeleton for intelligent agents or agent organizations. Organization Types are skeletons for agents whose logical functions are to coordinate a collection of agents (i.e., organizational members), and to manage their communications with outside agents and organizations.

DAI systems are constructed by instantiating (or specializing and instantiating) suitable agent Types and embedding application elements within those "wrapper" objects. Application elements access the distributed services of its embedding Type instance through discrete high-level message-based ***Protocols***. A given DAI system can integrate multiple heterogeneous agents and agent organization Types.

Agent Types are structured as an inheritance hierarchy of object classes, whose initial subclasses are shown in Figure 2. Discrete application elements (e.g., knowledge sources), are embedded in basic Receptionist agent skeletons. Specialized subclasses of the Receptionist, called Gateways, are instantiated for embedding protected knowledge or data bases. The Manager Type is the root Agent Organization class. Manager subclasses include variant blackboard architectures and other organizational models such as have been developed in single problem DAI research. These Types are described further in the MetaAgents section of this paper.

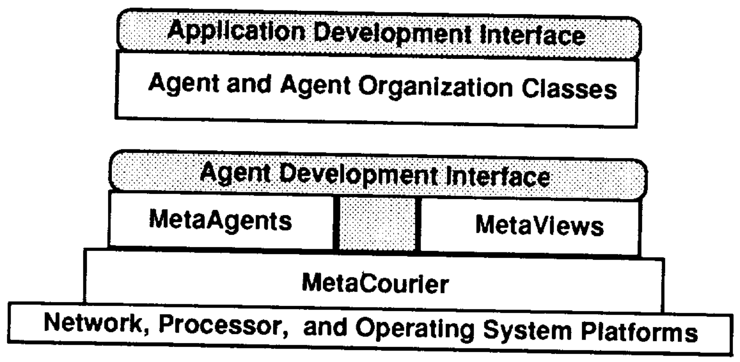
Application Development Interface
Agent and Agent Organization Classes
Agent Development Interface
MetaAgents
MetaViews
MetaCourier
Network, Processor, and Operating System Platforms

Figure .1: SOCIAL Architecture

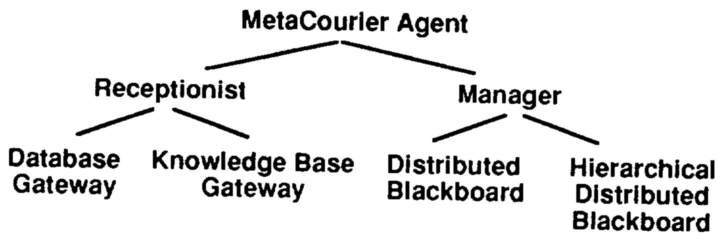

Figure .2: Agent Types in SOCIAL's Application Library

Each Type is comprised of other kinds of objects called ***Models***, which define different aspects of distributed behavior. Models are accessed through a separate Agent Development Interface. At present, SOCIAL describes three types of Models, which are represented in terms of compilable object-oriented languages. The MetaCourier language, SOCIAL's basic substrate, defines a class of Models for distributed communications. MetaAgents defines a class of intra- and inter-process control Models for agent and agent organizations. The MetaViews language defines a class of Models for accessing different models of data and knowledge. Both languages exploit MetaCourier's distributed communication services.

In effect, developers use the Application Interface to access a ***library*** of predefined DAI building blocks. Most of these objects can be customized by setting mode switches that override default services such as error-handling behavior. Applications may sometimes require service options or new behaviors not provided by the library of existing agent Types. In these situations, the dedicated languages comprising the Agent Development Interface can be used to extend the library by defining specialized Models and combining them to create new agent Type subclasses.

Operations Support using SOCIAL

A hypothetical example of a DAI operations support system based on SOCIAL is illustrated in Figure 3. The target domain is a distributed ground control network such as a launch processing system, consisting of user consoles, computers, data links, ground support equipment, and embedded sensors. Sensor monitor programs would be realized as Receptionist agents, with asynchronous or synchronous communication Models, depending on individual polling requirements. A relational database for tracking problems would be integrated using a Gateway agent. A Blackboard-based data fusion Manager Agent would coordinate sensor polling, measurement interpretation, and anomaly detection. A diagnoser agent would generate and test fault hypotheses and issue recovery suggestions to an Executive Manager Agent. Operations users would view ongoing activities and issue queries or commands to the Executive and database Gateway agents.

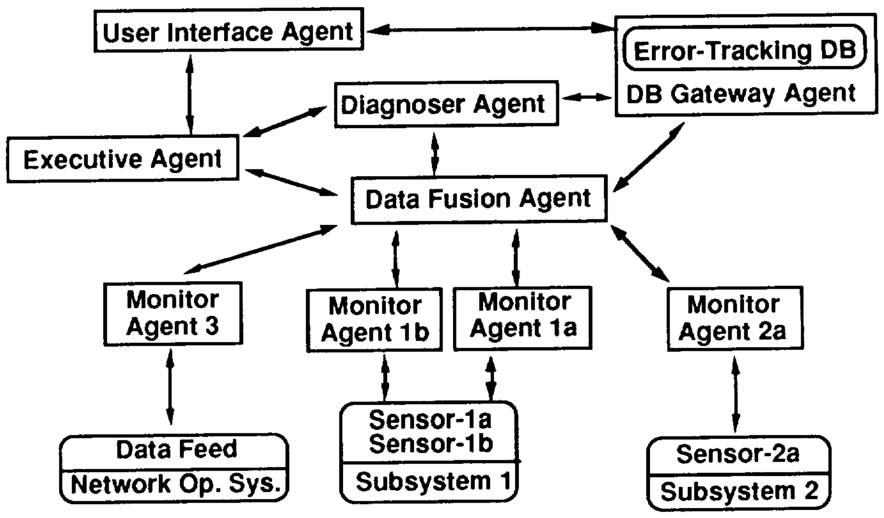

Figure .3: Hypothetical DAI System for Operation Support based on SOCIAL

The remaining sections describe SOCIAL's underlying languages and Models, which enable Agent Types to provide distributed services for integrating and coordinating DAI application elements.

MetaCourier: A Language for Distributed Communication

Advanced operational support architectures for space and ground control systems will have to integrate emerging hardware and software technologies with existing applications (both conventional and intelligent), interfaces, languages, and hardware platforms. Cost and reliability concerns dictate an integration strategy that minimizes intrusive modifications to existing system elements and allows them to be maintained and enhanced independently. Moreover, this strategy should maximize portability, to enable migration of system components to newer, high performance processor platforms. Technology transfer and management risks are also minimized, by reducing adjustments to training and operational procedures, and standdowns for system replacement and validation.

MetaCourier is a high-level object-oriented language for distributed communication that is designed to achieve these system integration objectives [Pa88]. The leading alternative communication model, based on the Remote Procedure Call (RPC) facility, is asymmetric and pairwise-restricted: an active ***client*** process invokes one (and only one) passive ***server*** process, which responds as required. In contrast, MetaCourier services provide fully peer to peer transparent communication between distributed applications.

The MetaCourier language defines four major object classes, Agents, Environments, Hosts, and Messages. ***Agents*** are intelligent, self-contained, autonomous processes. ***Host*** object attributes characterize the structure of network nodes: their processors, operating systems, peripherals, network types and physical addresses. ***Environments*** depict software dependencies for Agents, such as language compilers, and editors. Environments can be specialized to enhance communication performance for particular data types (e.g., sparse arrays), by defining custom encoding and decoding methods.

A MetaCourier ***Message*** defines the specific distributed communication behavior used by an Agent when it executes in an Environment. Both asynchronous and synchronous message-passing Models are available. An application Agent communicates with another by formulating a Message using the relevant Model protocol, for example:

```
Asynchronous: (Tell :agent sensor-monitor :sys Symbl "(poll measurement-Z)"
Synchronous:  (Tell-and-Block :agent user-interface :sys Mac2
                   "(trigger-alarm sensor-1 window-2)")
```

MetaCourier handles message routing, transmission, and delivery services transparently to the source and target agents' associated applications. Conceptually, the Agents' associated Hosts and Environments act as filters that manage processing and network dependencies in the communication process (cf Figure 4). Distributed control is achieved in a DAI application when Agents autonomously invoke other Agents. Concurrency is realized when multiple Agents are invoked simultaneously (across multiple Hosts).

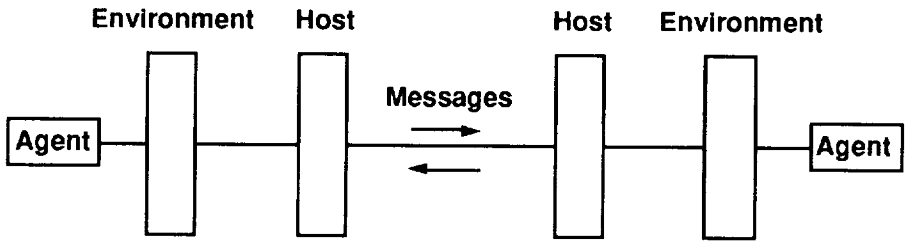

Figure .4: Operational Model of MetaCourier Communication Process

The openness of MetaCourier's communications architecture distinguishes SOCIAL from other DAI development frameworks, such as ABE, AGORA, MACE, ERASMUS [Ja88], and AF [Gr87].

Status: MetaCourier can be used as a standalone development language. It is currently available for: ANSI C and Common LISP programming languages; MS-DOS, UNIX, VMS, Macintosh Multifinder, and Lisp Machine operating systems; PCs, Macintoshes, Lisp Machines, VAX, Sun, and HP workstations. It currently utilizes TCP/IP on Ethernet and Appletalk protocols, but is extensible to other OSI-compatible network protocol suites.

MetaAgents: A Language for Agent Control and Coordination

The MetaCourier language offers a high-level interface that conceals the complexity of interprocess communication in distributed heterogeneous computing environments. Additional development capabilities are needed for internal process control, peer to peer inter-process coordination, and other distributed control services. MetaAgents is an object-oriented language for defining control Models to address these requirements.

The basic kernel MetaAgents Model protocols provide the equivalent of a traditional operating system's executive process control operations: agent creation, duplication, migration, and deletion. These protocols provide development-level options for specifying how to control Type inheritance behavior across distributed environments. The creation and copy protocols are critical because they allow new Agents to be defined dynamically at runtime.

MetaAgents Models support high-level message and concurrency management services. MetaCourier makes minimal assumptions about the ordering behavior of the low-level network protocols for message delivery, providing protocols to enforce simple message ordering schemes such as First In First Out (FIFO) delivery at particular nodes. MetaAgents Models define polices that use such guaranteed orderings to satisfy synchronization requirements of particular DAI applications [Pe89].

For example, MetaAgents supports an "atomic" broadcast protocol, which guarantees a globally invariant ordering of message delivery across all networks nodes. Atomic broadcast requires multiple phases of message exchanges; it should therefore only be used selectively, in situations where partial orderings of many-to-many agent interactions are insufficient and where lower performance can be tolerated. Atomic broadcasts are useful for maintaining consistency in transaction-oriented applications, such as where multiple agents send messages that operate on distributed replicated data.

MetaAgents defines other complex communication Models [Bi89] using a Group-based conversation abstraction: protocols are defined for agents to join a Group, to converse with other Group members via directed messages or broadcasts, and to depart the Group and the conversation. For example, a reliable Group broadcast protocol propagates information from one agent to other Group members such that all operational agents receive this information despite failures in the system.

Groups and broadcasts are very useful for replicating data for concurrency and fault management. Bottlenecks caused by centralized control can be alleviated by distributing task elements among agents that operate concurrently on replicated data and control state information. Similarly, data replicated along time-critical control paths can help to compensate for communication delay latencies in distributed networks (due to packet loss and node lode variances) that lead to violations of real-time processing constraints. Replicated data can also be used to maintain redundant copies of critical state information to facilitate recovery control strategies for fault tolerant behavior in distributed systems. Group protocols also ensure orderly reintegration of agents into DAI applications when dropped network links are recovered.

The following sections describe basic SOCIAL agent Types to illustrate the roles of MetaAgents Models.

Receptionists and Gateways

The Receptionist is the root or kernel MetaAgents Type for single agents. It specifies basic communication services through MetaCourier or more complex MetaAgents protocols and Group protocols. A Receptionist agent is responsible for serializing concurrent requests, for scheduling access to its embedded application, and for detecting and recovering from possible error states that the application might enter. Receptionists manage the control transactions that implement fault tolerant behavior; agents departing from a Group due to failures of nodes or network links and agents rejoining a task processing conversation following network recovery. Receptionists can also be designed to manage security functions, for restricting access to specific application elements.

Databases and application programs are often constructed using commercial development tools such as DBMSs and AI shells. SOCIAL simplifies the design of Receptionists in such cases by abstracting the application-independent aspects of tools' control and data interfaces into specialized, predefined Receptionist subTypes called Gateways. Gateway agents supply predefined interface protocols for formulating queries or commands, concealing variations of syntax across comparable tools. Accessing a resource or program through a Gateway reduces to defining the application-specific aspects of the interface, in particular, formulating queries or commands whose arguments reference particular objects or attributes. Gateways for AI development shells must provide bidirectional interfaces for control as well as for data, so that intelligent applications can initiate queries or commands to other agents in the context of their own environments.

Manager

The distributed services provided by Receptionists enable application agents to interact through a "loosely-coupled" model of cooperation. More sophisticated control is often needed to coordinate a set of agents working together on one or several closely related tasks. The MetaAgents Manager and associated subTypes provide the requisite organizational control functionality.

A Manager regulates all communication between the agents within an organization via directed and broadcast protocols, providing a shared memory and a locus for centralized oversight and control. The Manager agent also mediates communication between external agents and organizational members, such as requests for data or services. To accomplish these various routing functions, the Manager maintains a "database" describing member agents and their relationships. Managers can be replicated to avoid processing bottlenecks and single point failures, although this entails additional control and performance overhead.

Specialized Manager subTypes will realize specific tightly-coupled distributed control frameworks, such as blackboard architectures [Ni86,Ja89]. The Manager Type does not restrict membership based on agent Type. This means that organizations can be arbitrarily complex. In particular, SOCIAL supports hierarchical organizations, in which a Manager coordinates other Managers. Thus, SOCIAL's library of organization Types can incorporate or subsume popular single problem DAI architectures, as well as hierarchical (multiple problem) frameworks such as OPERA. More important, SOCIAL permits different elements of a complex DAI system to be implemented using ***different*** agent and agent organization Types. MetaCourier provides the substrate or "backplane" of distributed communication services that enables high-level integration and coordination. Developers can exploit SOCIAL's support of heterogeneity to implement application elements using the most appropriate strategies for control and cooperation.

Status: The MetaAgents language design specification has been finished. Kernel process control protocols have been implemented. Initial control Models, Gateways, and agent organization Types will be completed by mid-1990.

MetaViews: A Language for Accessing Heterogeneous Data Resources

SOCIAL's Gateway agent Type facilitates non-intrusive integration of databases and knowledge-based systems implemented using standard, commercial DBMSs and AI shells. Gateway interfaces and services for distributed communication and process control derive from MetaCourier and MetaAgents Models. Additional services are required for formulating and processing queries and commands. MetaViews will address this problem through interface Models that are specific to particular DBMS or AI shells. These Models will be comprised of two elements: high-level interface protocols and services for translating between the protocols and the tool language in question. The protocols represent SOCIAL's equivalent to a programming interface library.

Figure 5 depicts the functions performed by a MetaViews interface Model. The block on the left represents an application agent A embedded in a Receptionist; the right-hand box represents a database or knowledge-based system B embedded in a suitable Gateway. A issues commands for controlling or accessing B in terms of the functional protocols. A's Receptionist translator services convert those commands into an efficient canonical data represention, which are dispatched via MetaCourier. B's Gateway translator services converts canonical commands into the tool-specific language using the (invertible) protocol library. A's Receptionist and B's Gateway use MetaAgents services to manage concurrent messages.

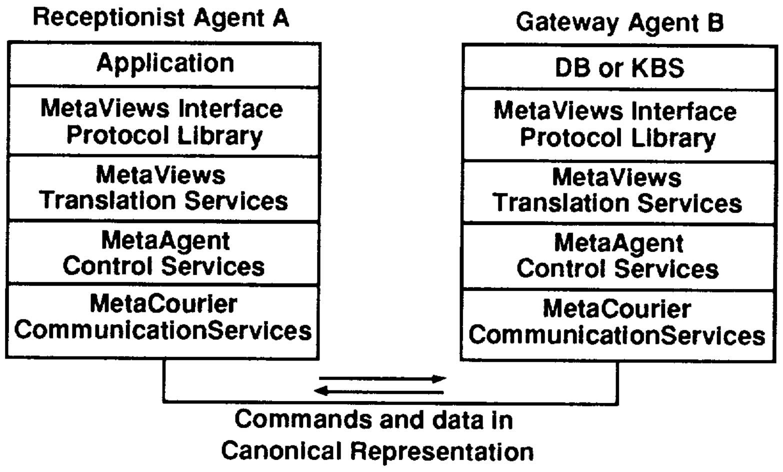
Receptionist Agent A
Application
MetaViews Interface Protocol Library
MetaViews Translation Services
MetaAgent Control Services
MetaCourier CommunicationServices
Gateway Agent B
DB or KBS
MetaViews Interface Protocol Library
MetaViews Translation Services
MetaAgent Control Services
MetaCourier CommunicationServices
Commands and data in Canonical Representation

Figure 5: Operational Model of MetaCourier Communication Process

MetaViews technology is extensible to integrate other kinds of information system tools, such spreadsheets, computer-aided design tools, data analysis libraries, and data acquisition software.

Status: The MetaViews language design specification has been completed. Initial versions of MetaViews Receptionist and Gateway Models for ANSI C and Common Lisp for Oracle and Sybase relational databases, KEE and CLIPS AI shells will be complete by mid-1990.

Conclusions

Operations support of complex systems exemplifies "multiple problem" Distributed Artificial Intelligence (DAI) domains. These domains are distinguished by their heterogeneity. Domain problems vary in difficulty and degrees of interdependence. Application software and data resources can differ substantially with respect to structure, complexity, intelligence, and interfaces. Software and hardware platforms are also typically heterogeneous. The central design concerns in such domains are: (a) to integrate these diverse elements non-intrusively; and (b) to supply flexible coordination models to allow intelligent applications to interact cooperatively as a coherent, unified system.

SOCIAL is a generalized tool for developing DAI systems. It simplifies design and maintenance by enforcing a clear separation between application-specific functionality and distributed services. Application elements access services through high-level interfaces to predefined agent and agent organization Types. SOCIAL's interfaces reduce complexity by concealing the mechanics of distributed communication and control across heterogeneous computing environments. "Standalone" applications, both intelligent and conventional, and data resources can thus be integrated non-intrusively. Moreover, SOCIAL allows intelligent applications based on different internal control schemes to be integrated within a single complex DAI system.

SOCIAL partitions distributed services into distinct object-oriented Models for: distributed communication (the substrate for all higher-level services); control services for managing processes and concurrency, and for coordinating agents on particular "single problem" DAI applications; and data translation. The SOCIAL architecture is open and extensible, with separate development interfaces to the library of generic agent Types and to the language-based Models that comprise them. These high-level tools free developers to concentrate on essential DAI architectural issues, such as designing strategies for coordinating intelligent subsystems.

Acknowledgments

The development of MetaCourier has been sponsored by the Department of Defense, U.S. Army Signal Warfare Center, under Contract No. DAAB10-87-C-0053. The development of SOCIAL has been sponsored by NASA under contract No. NAS10-11606.

References

[Ad89a] R. Adler, B. H. Cottman. " A Development Framework for Distributed Artificial Intelligence." *Proceedings Fifth Conference on AI Applications, Computer Society of the IEEE, Miami, FL, March 6-10, 1989.*

[Ad89b] R. Adler, A. Heard, and R. B. Hosken. " OPERA - An Expert Operations Analyst for A Distributed Computer Network." *Proceedings Annual AI Systems in Government Conference, Computer Society of the IEEE, Washington, D.C., March 27-31, 1989.*

[Ad89c] R. Adler. "A Distributed Blackboard Arhictecture for Integrating Loosely-Coupled Knowledge-Based Systems." *Intelligent Systems Review.* Association for Intelligent Systems Technology, E. Syracuse, NY, 1989.

[Ba88] S. E. Bayer, R. A. Harris, L. W Morgan, J. F Spitzer. *A Review of Space Station Freedom Program Capabilities for the Development and Application of Advanced Automation.* The MITRE Corp, Technical Report MTR-88D00059, McLean, VA, December, 1989.

[Bi89] K. Birman et. al. *The ISIS System Manual V1.2.* Department of Computer Science, Cornell University, Ithaca, NY, June 1989.

[Bo88] A.H. Bond and L. Gasser, eds. *Readings in Distributed Artificial Intelligence.* Morgan-Kaufmann, San Mateo, CA, 1988.

[Br89] M. R. Barry. "PX1: A Space Shuttle Mission Operations Knowledge-Based System Project." *Proceedings Annual AI Systems in Government Conference, Computer Society of the IEEE, Washington, D.C., March 27-31, 1989.*

[Bs87] R. Bisiani, F. Alleva, F. Correrini, A. Forin, F. Lecouat, and R. Lerner *Heterogeneous Parallel Processing: The Agora Shared Memory.* Carnegie-Mellon University, Computer Science Department. CMU-CS-87-112. March 1987.

[Er80] L.D. Erman, F. Hayes-Roth, V.R. Lesser, and D.R. Reddy "The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty." *ACM Computing Survey, 12, pp. 213-253, 1980.*

[Ga86] L. Gasser, C. Braganza, and N. Herman. *MACE: A Flexible Testbed for Distributed AI Research.* Distributed Artificial Intelligence Group, Computer Sci. Dept. USC, 9-Aug-1986.

[Gr87] P.E Green. "AF: A Framework for Real-TIme Distributed Cooperative Problem Solving." in [Hu87].

[Ha86] B. Hayes-Roth. "A Blackboard Architecture for Control." *Artificial Intelligence, Vol.262, pp.251-321, Mar,1986.*

[Ha88] F. Hayes-Roth, L. D. Erman, S. Fouse, J. S. Lark J. Davidson. "ABE: A Cooperative Operating System and Development Environment." in [Bo88], pp. 457-489.

[He87] A.E. Heard. "The Launch Processing System with a Future Look to OPERA." *Acta Astronautica, IAF-87-215.*

[Hu87] M. N. Huhns, editor. *Distributed Artificial Intelligence.* Morgan-Kaufmann, Los Altos, California, 1987.

[Ja88] V. Jagannathan, R.T. Dodhiawala, and L.S. Baum " Boeing Blackboard System: The Erasmus Version." *International Journal of Intelligent Systems, Vol 3 Number 3, Fall 1988, pp. 281-294.*

[Ja89] V. Jagannathan, R.T. Dodhiawala, and L.S. Baum, eds. *Blackboard Arhictectures and Applications.* Academic Press, San Diego, CA, 1989.

[Le83] V. R. Lesser and D. D. Corkill. "The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks." *AI Magazine, Fall 1983 pp. 15-33.*

[Mu89] J. F. Muratore, T. A. Heindel, T. B. Murphy, A. N. Rasmussen, R. Z. McFarland. "Applications of Artificial Intelligence to Space Shuttle Mission Control." *Proceedings Conference on Innovative Applications of AI, Stanford, CA, March, 1989, pp 15-22.*

[Ni86] H.P. Nii "Blackboard Systems: The Blackboard Model of Problem-Solving and the Evolution of Blackboard Architectures." *AI Magazine, pp. 38-53, Summer 1986.*

[Nu88] R.O. Nugent and R. W. Tucker. "An Architecture for Integrating Distributed and Cooperating Knowledge-Based Air Force Decision Aids." *Second Annual Space Operations Automation and Robotics Workshop (SOAR 88), Dayton, Oh, July 1988.*

[Pa88] R. C. Paslay. *metaCourier: A Language For Distributed Heterogeneous Communication.* Symbiotics Inc. Cambridge, MA March 1988.

[Pe89] L.L. Peterson, N.C. Bucholz, and R.D. Schlichting. " Preserving and Using Context Information In Interprocess Communication." *ACM Transactions on Computer Systems, 7,3, August 1989.*

[Ru88] K. S. Rubin, P. M. Jones, C. M. Mitchell, T. C. Goldstein. "A Smalltalk Implementation of an Intelligent Operator's Assistant." *Proceedings Object-Oriented Programming Systems, Languages, and Applications, September, 1988, pp 234-247.*