Agent and Object Technologies for High-end Collaborative Applications *

Mustaque Ahamad, Raja Das, Karsten Schwan,
Sumeer Bhola, Fabian Bustamante, Greg Eisenhauer, Jeremy Heiner,
Vijaykumar Krishnaswamy, Todd Rose, Beth Schroeder and Dong Zhou

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
Email: {mustaq,raja,schwan}@cc.gatech.edu

Abstract

Complex distributed collaborative applications have rich computational and communication needs that cannot easily be met by the currently available web based software infrastructure. In this position paper, we claim that to address the needs of such highly demanding applications, it is necessary to develop an integrated framework that both supports high performance executions via distributed objects and makes use of agent based computations to address dynamic application behavior, mobility, and security needs. Specifically, we claim that based on application needs and resource availability, it should be possible for an application to switch at runtime between the remote invocation and evaluation mechanisms of the object and agent technologies being employed. To support such dynamically configurable applications, we identify several issues that arise for the required integrated object-agent system. These include: (1) system support for agent and object executions and (2) the efficient execution of agents and high performance object implementations using performance techniques like caching, replication, and fragmentation of the state being accessed and manipulated. We are currently developing a system supporting high end and scalable, collaborative applications.

1 Introduction

Over the last few years, the World-wide Web has emerged as a popular vehicle for developing distributed collaborative applications. For example, a Web browser enhanced with support for

*This work supported in part by NSF grants CDA-9501637 and ASC-9624987 and DARPA contract DABT-63-95-C-0125.

Java applets and Java remote method invocation (RMI) can be used to structure an application where many nodes can perform local computations and cooperate by invoking remote object methods. Although this approach is viable for applications that have limited computational and communication needs, new application are emerging that allow widely distributed users to share rich 3D environments (e.g., complex shared visualizations[2], VR based distributed games[3], etc.). To meet the requirements of such applications, it is necessary to provide support for their efficient execution and for low latency access to state shared by distributed application components.

We posit that object and agent technologies should be integrated in future middleware not only because this will result in a single programming model for application developers but more importantly, because these technologies provide complementary functionalities. Namely, distributed objects can offer high performance across wired platforms, using their invocation mechanisms for shipping data to remote nodes, assuming those platforms are able to run their compiled code. Agents, on the other hand, facilitate the use of a wide variety of platforms in both connected and disconnected operation, using their remote evaluation mechanisms via code shipped to those platforms and interpreted on them. Clearly, an agent running interpreted code cannot be assumed to carry out tasks with the same performance as a compiled object.

In this position paper, we propose a framework that can meet the quality-of-service (QoS) requirements of resource intensive and dynamic collaborative applications. Such a framework must include the following support:

- *Morphing objects/agents into agents/objects* – the remote evaluation mechanism that allows a node A to ship code to another node B (via Java applets or the more general agent paradigm) is valuable in dealing with the mobile and highly dynamic nature of application interactions. However, if significant computation needs to be performed (e.g., to generate a complex visualization or mine interesting data), interpretive execution of the agent code at node B may be inefficient. Thus, it is necessary to develop techniques that provide more efficient ways of code execution than what are currently available. Object representations of such code, when possible, could be one technique for improving execution performance. In addition, execution performance can be enhanced by combining into a single agent the executions of multiple, cooperating agents that have common computations (e.g., mining the same source of data or performing multiple operations on such data). Thus, a flexible framework must include support for both object and agent based executions and mechanisms for switching from one to the other as application requirements and available resources in the system change.

- *Flexible sharing of state* – the agent/object infrastructure addresses the problem of how the code of applications structured using objects and agents be executed flexibly. Once mechanisms are in place to allow an application's code to be executed as an object or agent at a node, the system must make the state accessed by the code available at that node. Furthermore, in a complex distributed collaborative application, the same state could be accessed and shared among multiple application components executing at different nodes. Therefore, to support the full flexibility offered by the integrated object and agent mechanisms, a state sharing system has to be developed. Such a system can employ a variety of techniques that include replication, fragmentation and caching of the shared state that is accessed by object and agent computations. Moreover, for such shared state, the object and agent computations must be able to express constraints concerning the manner in which this state is used in order to meet the service guarantees made to applications.

- *Runtime configuration via events and attributes* – the runtime management of mixed agent and object based implementations of dynamic distributed applications requires the availability of light-weight mechanisms that can aid in reconfiguring these implementations. Toward this end, we propose the use of *distributed events* which can monitor application behaviors and enact changes, by activating configuration services that perform tasks like object adaptation and agent composition. Configuration services rely on support which allows objects and agents to be configured via 'hooks' into their implementations, called *attributes*. Attributes express selected implementation detail and associated resource needs. Resource management services may be attached to such attributes by being encapsulated in *configuration objects*, which may be invoked synchronously with accesses to objects or agents, or as stated earlier, their execution may be triggered by events in response to external factors or to changes in object, agent, or execution platform state. Thus, events and attributes will allow us to adapt the dynamically changing resource needs of the applications.

The remainder of this position paper first outlines the infrastructure and the required runtime support and compilation techniques to support the development of applications using both object- and agent-based paradigms through a single API (see Section 2). In Section 3, it describes the representation, access and manipulation facilities that need to be provided to enable flexible information sharing between multiple program components. Thus, the paper documents our approach and ideas in developing a flexible system that can provide the infrastructure for high-end collaborative applications. Integration of the framework with the Web, application development, and evaluation will be topics of future papers. We present a brief description of the system being developed by us in Section 4. The relevant related work is described in Section 5. Conclusions and future directions appear in Section 6.

2 Infrastructure for Object and Agent Executions

Our research is exploring the integration of object and agent technologies, with the intent of leveraging the ubiquitous nature of agent-based computations while also being able to utilize the potentially high performance of object-based systems for complex, distributed, collaborations. Both methods of computation will be made available to end users by providing a single API, which therefore, will have to be "rich" enough to capture the various interfaces (calling sequences) required for both object invocation and agent execution. In addition, this API will also be used by online algorithms that undertake performance configuration and resource management tasks.

The integration of object and agent implementations will be achieved with a repository mechanism. The repository will contain (1) the actual code for the objects/agents used by applications and (2) information about currently existing objects and agents, enhanced with information about their alternative implementation. The same repository can be shared by multiple applications. Each repository will perform all necessary bookkeeping like registering new objects/agents, updating existing object/agent information, etc. Since multiple applications can use the same repository, the service request level at a repository can get very high, or the 'distances' between applications and repositories may become too large. In such cases, the configuration service keeping track of the request level may decide to clone and generate a new repository instance, and hand over the responsibilities for certain requests to the new repository. Such multiple repositories will interact because there will be a certain amount of overlap between the information they contain.

Each repository's API consists of two access windows: (1) an *information window* and (2) an *implementation window*. The information window lists all of its available objects and agents, the required calling sequences associated with them, status flags for their code(s), the associated *composition descriptors* (defined below), and other relevant information. The status flag for an agent/object entry in the repository contains information about whether its code is compiled or not and if so, for what architecture, or whether it is bytecode (e.g., generated by a Java compiler), or whether it is code written in a high-level language code (e.g., C, C++). Other information recorded for an object/agent may include the location (another repository) of an alternative implementation of the same object/agent which is highly optimized hence executes faster. Depending on QoS requirements, the configuration layer may choose to employ the services of the more optimized version of the object/agent implementation.

The implementation window provides a number of services. Its primary service is access to the object or agent code. In addition, it provides the interfaces for the composition of simple agents into more powerful ones. Agent composition is performed using a compiler, which is started by the repository when necessary.

Another feature of this integrated technology is the flexibility with which agents can be used as objects. Agents by nature are mobile. They move (user controlled) from site to site performing some specific computation. In our environment, depending on requirements, we will sometimes compile the agent code (usually written in some high-level or interpreted language) into machine specific code. Once such an architecture specific compilation has been performed for a particular agent instance, the instance loses its mobility and becomes equivalent to an object which provides the same functionality as the agent.

For ease of mobility between various architectures, most agents are written in interpreted languages (like Java, Tcl/Tk). Interpreted code runs up to 100 of times [17] slower than regular compiled code. To overcome the inefficiency of interpretation, just-in-time compilation systems [21, 15] have been proposed. Just-in-time (JIT) compilers generate native code on demand at runtime. Naturally, to keep the compilation time to a minimum, aggressive optimizations have to be avoided. Hence, if the agent is computationally intensive, JIT compilers do not help. In our approach, for computationally intensive agents, we will use either of the following two choices, depending on the agent code's representation: (1) if the agent is available in bytecode, we will use a bytecode compiler, to generate the efficient native code, or (2) if the agent is available in a high-level language like C/C++, we will use the native compiler for the machine where the agent will be executing.

2.1 Composition: A Mechanism for Generating Powerful Agents

Configuration techniques need to be developed for agents to address their potential lack of performance. Associated with each agent is a type of configuration object, called a *composition descriptor*, which is used to fuse smaller agents to generate large powerful agents. We can think of agents as specialized filters[1], thus allowing us to use novel compiler optimization techniques like filter fusion [7] to glue agents together. The agent composition compilers are part of the object/agent repository. *Distributed events* are used to recognize the simultaneous use of multiple agents to carry out a task, hence being prime candidates for fusion. A sample scenario in which such an action is highly useful is one in which several agents have 'found' a profitable source of information (e.g., a relevant sensor device) and are 'mining' it for future or remote use. In such cases, rather than continuing to use potentially inefficient agent realizations, a single, efficient new agent or even an object (using compilation in place or using a remote object repository) may be used to replace the ongoing inefficient cooperation between the multiple agents.

An agent composition compiler takes multiple agents as input and generates a single efficient agent. The composition descriptor associated with each agent is utilized by the composition compiler in its effort to glue the various input agents together. Various interesting issues being explored by us in building composition compilers are:

- Aggressive inter-agent analysis (this is like interprocedural analysis) will be performed to generate an efficient agent from multiple ones.
- Instead of spending substantial time doing inter-agent analysis, the compiler may decide to do minimum fusion between the agent codes, but make the various agents share certain large data-structures.
- How aggressively (if at all) can we compose agents written in different languages? In such cases, depending on the usage pattern, we may have to just connect the output of one agent to the input of another.

Most of the techniques that will be explored can be driven based on QoS requirements. We will also investigate techniques for the efficient migration of agents across heterogeneous platforms. For example, the amount of state being migrated can be controlled by the agent via stated QoS parameters (discussed next). Additionally, we will explore how compiler generated code can be used to efficiently capture execution state image when an agent needs to migrate.

3 Shared State

The object/agent infrastructure allows applications to be structured as collections of objects and agents. The execution of object methods requires access to the state of the object. Similarly, agent code needs to migrate to nodes where state accessed by them is available. Thus, the

[1] Filters are data manipulation abstractions which read data from a single source and write data to a single destination.

flexibility with which agent and object executions can be scheduled depends on where the state accessed by them can be made available efficiently. In this section, we argue that a state sharing system, coupled with the object/agent execution infrastructure, can provide rich resource management policies for executing highly dynamic applications.

The programming and execution performance of interactive applications could depend significantly on the interfaces provided for sharing state across application components and the implementations that enable the sharing. The problem of state sharing is to make available the state when and where it is accessed with minimum possible access latency and overhead. This problem is complicated by the fact that shared state gets updated at unpredictable times and future accesses at nodes cannot always be known in advance. For example, in a distributed game scenario, a node needs to access the state of an object only when the object is in its vicinity, and the accuracy requirements for the object's state also depend on the level of engagement.

Distributed and parallel computations can access shared state by using techniques that range from distributed shared memory to distributed objects. However, to obtain the desired performance guarantees for such accesses, the underlying system cannot simply react to the read and write requests when they are issued. We claim that applications must be able to communicate their requirements to the system which can exploit them to optimize its implementations. Our system will provide a QoS based API that can be used by applications to communicate their state sharing needs to the system when the application computations are structured using agents and objects. We are also investigating the associated configuration and resource management issues. Such issues include what fragments or copies of shared state should be stored at various nodes, their access methods (e.g., remote invocation via objects or by an agent) and the associated consistency protocols when shared state is modified.

3.1 A QoS API for Accessing Shared State

Sharing requirements can be characterized by a number of different parameters. Access response time and level of availability are two parameters that have been used in the past. We are exploring new and novel QoS parameters for information sharing in the object and agent environment. For example, to meet response time or availability requirements, it may be necessary to replicate and cache the objects that encapsulate shared state at multiple nodes. This introduces the problem of consistency, and the level of consistency is a QoS parameter that can be specified by the application. For example, in a distributed game, the distance between two objects can be converted into meaningful levels of consistency[3] for the copies of these objects. In the agent model of computation, a migrating agent carries its execution state with it. The amount of state that is carried by it can be controlled by QoS parameters such as MUST and MAY. Data that has the attribute MUST has to be moved as part of the migrating agent. On the other hand, if it is marked MAY, then the data may be moved depending on resource availability. If the data is not moved, the program will still execute correctly, though it might have to regenerate the MAY data.

We are developing an API that will allow applications to specify their sharing needs via QoS parameters. An analysis of the information access requirements for collaborative applications will help us define a concrete set of parameters and the calls that are used to specify them. Such parameters can be specified either when an object that encapsulates the state is created or when it is accessed. The level of availability for an object is naturally specified when the object is created. On the other hand, sometimes the consistency level of an accessed copy can be specified only when the object is invoked. Certain QoS parameters also need to be specified across a set of objects. For example, in addition to intra-object consistency, an application may also need consistency across a set of related objects that appear in a single visualization at the screens of multiple participants. Finally, since application needs can change dynamically, the API must allow applications to upgrade or downgrade their QoS requirements.

Although QoS can be specified for object invocations (e.g., BBN object QoS [9]), we define QoS at the level of state accessed by such invocations for two reasons. First, execution of the invocations is enabled by access to the state of the invoked objects and in distributed environments, invocation performance could greatly depend on the cost of accessing shared state. If access to state can be provided at the required level of performance, the object/agent mechanisms and the repository will allow us to schedule the computation's execution at one or more nodes. Second, information that is meaningful at this level (e.g., executing my request with a copy that may not have the latest updates is acceptable) is not easy to provide at a higher level but could have a significant impact on the resource usage in the system.

3.2 Configuration and Resource Management for Shared State QoS

The configuration and resource management issues for shared state QoS include where the shared state should be kept, how it is accessed, and how and when changes to the state made at one node are propagated to other nodes. To avoid high latency and communication costs and to exploit locality, multiple copies of shared state or its fragments need to be created at or close to nodes where it is frequently accessed. Our goal is to develop configuration and resource management algorithms, which take QoS requirements of a dynamic set of users and do the following:

1. Access Methods and Replica/Fragment Management: The state shared by application components must be stored at one or more nodes in a distributed system. The nodes where copies of such state need to be stored depends on the access patterns and locality of access. We will support both object and agent styles of access to shared state. In the former, the code that reads/writes the state is part of a local applications whereas in the agent style, remote nodes can ship code that accesses the state available at a node. The placement of copies or fragments of the shared state will depend on QoS needs. For example, if average response time (e.g., access time for most accesses) should be independent of the communication latency, a local copy can be created if permitted. However, creation of a large number of copies could increase the latency of certain consistency operations (e.g., update of all copies on a write). Thus, the problem is to determine the number and placement of copies while taking into account the QoS access parameters. The nodes that have copies can change as access patterns change.

2. Maintaining Consistency: Multiple copies of shared state that arise from replication, caching, and fragmentation need to be kept consistent. In particular, when a certain node updates such state, the effects of the update must be propagated to other nodes that access the state. Consistency models define the order in which updates can be viewed by nodes that access the shared state. The cost of maintaining consistency very much depends on the level of consistency required. We have explored several levels of consistency and their dynamic adaptation as application needs change. For example, in the simulation application, consistency level for the state of distant objects can be weaker. One based on causality in distributed system, can guarantee that the order in which updates are seen is consistent with the causal order. However, the effects could lag the actual occurrence of the updates because they can be propagated asynchronously. This can improve performance because updates can be batched and piggybacked on other messages. Synchronous updates to copies may be necessary at nodes that are in the immediate proximity of an object. This can be provided by employing a strong consistency protocol. Thus, multiple consistency levels, strong for objects copies that are in close proximity and causal consistency (or other forms of consistency) for distant objects can be used based on different QoS needs of different nodes.

Our work will explore multiple levels of consistency and how the consistency level associated with certain shared state can be changed in response to changed QoS requirements of the applications. The consistency levels will range from the traditional strong consistency to ones that exploit temporal and spatial requirements of applications. We are developing novel implementations of such consistency levels which will have the following properties:

1. We address a dynamic environment where copies of shared state can be created and deleted dynamically. We will develop light-weight protocols for creating and deleting copies. Currently existing protocols for such operations are expensive because they synchronously force all nodes to become aware of the membership change.

2. We support multiple access methods. For example, shared state can be read and processed by locally available native code or a remote node can send an agent to access the information. Depending on QoS needs and available resources, we can choose between the different access methods.

3. We develop consistency protocols that can use both *push* and *pull* based techniques for disseminating updates to shared state. In a *push* scheme, the node where an update is done is responsible for sending the updated state to other interested nodes. A *pull* scheme allows a node to fetch a consistent copy of such state when it needs it. These two schemes allow general resource management algorithms to be implemented when push can be done to nodes that frequently access the shared state and others can pull it on demand.

4. The access and consistency protocols need to move updated state as well as other control messages over the network. Thus, it is necessary that the communication system also provides a QoS interface which can be used to reserve sufficient communication resources or even configure them to meet the shared state QoS. We will couple our shared state protocols with network and processor QoS interfaces and configuration mechanisms.

The configuration and resource management problems can be viewed as a mapping problem from QoS parameter values to appropriate placement, fragmentation, access, and consistency protocols. We will implement a library of such protocols and experimentally evaluate the QoS that can be supported by them when shared state is accessed at a set of nodes.

The proposed QoS formulations and techniques address a key requirement of large-scale distributed collaborative systems: consistency of state shared by its entities expressed at a level meaningful to applications, formulated in terms of application semantics. The runtime use of these formulations will permit configuration algorithms to exploit resource tradeoffs in the implementations of object and agent based applications containing mixes of computationally intensive, interactive, real-time, and simulation components: (1) shared objects used by multiple interactive components, such as collaborating end users or real-time image streams accessed by battlefield displays, and (2) shared objects that maintain state of multiple federated simulations.

3.3 Distributed Events: Analysis and Transport

Events have been receiving renewed attention in the OS community as a useful structuring mechanism for distributed and parallel systems. Event services also form the basis of novel architectures for large-scale interactive systems now being developed in research and even in industry environments (e.g., CORBA events and IBM's Event Reaction Architecture at TJ Watson Laboratories). Clearly, for our effort, it is critical that we efficiently collect and transport events from their sources (e.g., from the machine on which two agents are jointly accessing and processing data from a single, shared source) to their destinations (e.g., to the configuration manager that realizes the composibility of both agents, then uses a nearby repository to do so), analyzing them 'on their way', at their destinations, or both. In addition, such distributed event services must interact cleanly with events captured from lower-level system components. For instance, when a failed QoS requirement is detected by a low-level communication protocol, then such a 'local' event must give rise to a transportable, globally meaningful event that may result in QoS renegotiation, changes in resources, or adaptations of software components.

Our system design (1) allows applications to attach 'actions' to low-level events, such as 'QoS requirement not met', then (2) can reflect such local events to higher level handlers (essentially an object-oriented version of upcalls), which in turn (3) can formulate globally meaningful events and deliver them to their intended (and dynamically varying) sources. Given these needs, we will develop a configurable and efficient user-level event services library. This library will be efficient in that it is layered on network transport services, rather than on remote invocations as is the case for most CORBA event implementations. It will be configurable in its use of multiple underlying transport services, depending on network connectivity, and in its ability to associate event analyses with their transport. One such analysis is described in [20], where the purpose of analysis is online hazard detection. A similar constraint-based approach can be used to generate and perform event analysis for the large-scale distributed applications.

4 System Development and Status

We have started development of a system that will support the integrated agent–object framework with the QoS based state sharing and distributed event systems. First, an IDL/C/C++ object environment is being developed with configurable remote invocation support. Object invocations will have associated *attributes* that can specify resource needs or express configuration actions to meet higher level QoS specifications. These attributes will be used to control invocation behavior and choose appropriate communication facilities, or configure them, at any time during a program's execution. Attributes may be used explicitly or via *configuration objects* offering higher level and QoS-based interfaces to programs.

Second, a distributed event handling facility is being put in place, with some of its mechanisms leveraging our considerable prior work in online program monitoring[22] and in lightweight data transfer mechanisms for dynamic, distributed applications[23]. Third, we are currently developing the repository mechanism which allows for the storage of both agents and objects. Agents can be written in both interpreted languages like Java, and in non-interpreted languages like C/C++. For our preliminary version, the repository itself is being developed in Java. At the time we were writing this paper, we were adding mechanisms to the repository that can be used to register and store agent code. When an agent is registered at a repository, the interface definition (calling sequence) has to be supplied and this is stored in the *information window* and the actual code is stored in the *implementation window*. The repository provides access mechanism that can be used to "pick up" copies of the residing agents. During the time of the workshop we will have a completed preliminary system with a datamining application developed based on the infrastructure provided by the system.

5 Related Work

Recent work in distributed object technology has led to the CORBA and OLE/ActiveX architectures and their implementations are being addressed in projects like Electra [11], BBN Corbus [10], and by CORBA-compliant systems from industrial vendors. Similarly, Java or Tcl-based agent technologies are being explored by Agent Tcl [12], Tocoma [13] and Rover [5] projects. Our research will advance such technologies along several dimensions. First, our integrated investigation of agent and object technologies is unique. We will explore how one technology can complement the other one (e.g., the use of agents while a platform is running in disconnected mode, following by the use of an object-like entity when that platform is wired). Second, for improved performance, we will explore the execution of agents with native code. Systems such as Emerald [14] have explored native code mobility but this work does not exploit compiler technology to improve agent execution performance. Third, we will explore how the implementations of objects and agents can adapt to changing application requirements. Fourth and perhaps most importantly, we will not simply develop innovative technology, but will understand and evaluate its deployment for realistic, large scale applications.

Interpreted languages like Java or Tcl provide a solution to the design of safe agent computations. The tradeoff of using interpreted languages to build agent-type computation is efficiency;

interpreted code executes hundreds of times slower than compiled code. Recognizing this fact, novel compiling techniques, like JIT [21, 15], bytecode compilation [16, 19, 17] and a combination of JIT and offline compilation [18] have been pursued. Our work will utilize and extend the existing technology. We will allow agents to be developed in both interpreted languages and high-level programming languages like C/C++. When agents are developed in interpreted languages we can utilize some of the novel compiling techniques but otherwise they are not applicable in our case. The major thrust of our compiler work will be in the area of agent composition i.e. building larger, powerful agents from smaller ones. This will require indepth inter-procedural analysis [1] of the agent code (bytecode or C/C++).

Distributed object and file systems that encapsulate shared information have been investigated in many contexts. Although systems like the BBN 'Quality of Service for Objects' have identified QoS [9] parameters for object invocations, we will explore (1) resource management policies for replication, (2) multiple levels of consistency among interacting objects that can be dynamically adapted, and (3) the joint use of object and agent technologies to meet application QoS needs. Multiple levels of consistency and their implementations have been explored in several systems for replicated data. For example, the Coda and Ficus systems, motivated by mobility concerns, have developed techniques that allow locally existing copies to be read and written. Such writes can lead to concurrent updates and their resolution has been explored by these systems. Systems like Bayou make aggressive use of replication and provide multiple levels of consistency (via the different session guarantees). Our work will benefit from past systems like Coda [6], Ficus [4], and Bayou [8], but in contrast to such work, we will explore the formulation and then the mapping of higher level QoS requirements to the lower-level algorithms that have been developed by these systems.

We are exploring collaborative applications based on distributed immersive environments for simulations, including notions of different 'levels of detail' and about physical or logical 'distance' from objects. For instance, for a shared 'target' object in a simulation, it is meaningful for end users to define multiple levels of detail at which such an object may be perceived. Each such level of detail may be directly mapped to a level of consistency supplied by the underlying software/communication infrastructure. In addition, there are meaningful formulations of 'distance' among objects that affect their visibility (ie., their consistency). For instance, in collaborative or immersive environments, two end users manipulating entities in a shared world may not need to see each others' manipulations when they cannot possible interact. Note that these issues are not already addressed by past work on distributed shared memory and object, with DSM work typically derived from hardware-near consistency formulations and distributed object work offering primarily technical solutions, but not policies, for such problems. Technical solutions we have been involved with or are aware of include object replication, caching, and fragmentation and of course, for DSM include various formulations and implementations of data consistency.

6 Conclusions

We believe that future distributed systems' software infrastructures will have to support high-end collaborative applications that will have significant computational and communication resource needs. This trend is already evident from current linkages between different data CAVEs at the National Laboratories and from national efforts addressing metacomputing environments. We have outlined an integrated agent–object middleware that can be coupled with existing infrastructures like the Web to meet the needs of such applications. Three key aspects of such middleware include (1) an integrated object-agent system for flexible and efficient executions of computations, (2) the efficient sharing of state between such computations using alternative representations of shared state and dynamic mechanisms for changing those representations, and (3) support for dynamic system configuration, consisting of object/agent repositories, configuration objects, composition descriptors, and attributes, lightweight notions and implementations of distributed events, and QoS management permitting online decision making and enactment.

Several components of the proposed system are already in place. For example, we have developed a repository for agent/object executions. Also, we have built a shared state system that provides multiple levels of consistency and evaluated its performance for various workloads. Similarly, reconfigurable objects via attributes and configuration objects have already been developed in our previous work, built on data and event transports. Last, we have already developed several collaborative applications and are in the process of integrating them with the object systems proposed in this paper. Our future work will integrate these systems and make them accessible from the Web to build large scale collaborative applications that share rich visualizations. We will then be able to evaluate the benefits offered by the combined agent–object execution mechanisms and the shared state policies developed by us.

References

[1] Gagan Agrawal, Joel Saltz, and Raja Das. Interprocedural partial redundancy elimination and its application to distributed memory compilation. In *Proceedings of the SIGPLAN '95 Conference on Programming Language Design and Implementation*, pages 258–269. ACM Press, June 1995. ACM SIGPLAN Notices, Vol. 30, No. 6. Also available as University of Maryland Technical Report CS-TR-3446 and UMIACS-TR-95-42.

[2] Yves Jean, William Ribarsky, Song Zou, Jeremy Heiner, Karsten Schwan, and Bobby Sumner. Collaboration and visual steering of simulations. In *Proceedings of the SPIE Conference on Visual Data Exploration and Analysis IV*. SPIE, 1997. to appear.

[3] Richard West, Karsten Schwan, Ivan Tacic, and Mustaque Ahamad. Expoliting temporal and spatial constraints on distributed shared objects. In Proc. of the *International Conference on Distributed Computing*, 1997.

[4] R. G. Guy, J.S. Heidemann, W. Mak, T. W. Page, G. J. Popek, and D. Rothmeier. Implementation of the ficus replicated file system. In *USENIX Conference Proceeding*, June 1990.

[5] A. D. Joseph, A. F. de Lespinasse, J. A. Tauber, D. K. Gifford, and M. F. Kaashoek. Rover: A toolkit for mobile information access. In *Proc. of 15th ACM Symposium on Operating Systems Principles*, December 1995.

[6] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *ACM Transactions on Computer Systems*, 10(1), February 1992.

[7] T. Proebsting and S. Watterson. Filter Fusion. In *Conference Record of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, January 1996.

[8] D. Terry, A. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. Welch. Session guarantees for weakly consistent replicated data. In *Proc. of International Conference on Parallel and Distributed Data Systems*, September 1994.

[9] J. A. Zinky, D. E. Bakken and R. Schantz. Overview of Quality of Service for Objects. *Proc. of the Fifth IEEE Dual Use Conference* May 1995.

[10] BBN Corbus. http://www.bbn.com/offerings/corbus/corbus.html.

[11] Silvano Maffeis. Adding Group Communication and Fault-tolerance to CORBA. In *Proc. of USENIX Conference on Object-Oriented Technologies* June 1995.

[12] D. Kotz, R. Gray and D. Rus. Transportable Agents Support Worldwide Applications. In *Proc. of ACM SIGOPS European Workshop* September 1996.

[13] D. Johansen, R. van Renesse and F. B. Schneider. Operating System Support for Mobile Agents. In *Proc. of Fifth Workshop on Hot Topics in Operating SYstems* May 1995.

[14] B. Steensgaard and E. Jul. Object and Native Code Thread Mobility Among Heterogeneous Computers. In *Proc. of 15th SOSP*, 1995.

[15] Symantec Corporation. Just-in-time compiler for windows 95/nt. 1996. http://cae.symantec.com/cafe/fs_jit.html.

[16] T. Keishiro. J2c java .class to c translator. 1996. http://www.webity.co.jp/andoh/java/j2c.html.

[17] Gilles Muller, Barbara Moura, Fabrice Bellard, and Charles Consel. Jit vs. offline compilers: Limits and benefits of bytecode compilation. Technical Report Publication Interne No. 1063, Institut De Recherche En Informatique Et Systemes Aleatoires (IRISA), 1996.

[18] M.P. Plezbert and R. K. Cytron. Does "just in time" = "better late than ever". In *Conference Record of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM Press, 1997. To appear.

[19] T. Proebsting. Juice : A java-to-c translator. 1996. http://www.cs.arizona.edu/sumatra/juice/.

[20] Beth Schroeder, Sudhir Aggarwal, and Karsten Schwan. Hazard detection using on-line analysis of safety constraints. Technical Report GIT-CC-97-01, College of Computing, Georgia Institute of Technology, 1997. http://www.cc.gatech.edu/tech_reports.

[21] Softway. Introduction to guava. 1996. http://guava.softway.com.au/introduction.html.

[22] Greg Eisenhauer, Weiming Gu, Karsten Schwan, and Niru Mallavarupu. Falcon – toward interactive parallel programs: The on-line steering of a molecular dynamics application. In *Third IEEE International Symposium on High-Performance Distributed Computing (HPDC-3)*, pages 26–34. IEEE, 1994. also available as technical report GIT-CC-94-08, College of Computing, Atlanta, GA.

[23] Beth Schroeder, Greg Eisenhauer, Karsten Schwan, Jeremy Heiner, Vernard Martin, Song Szou, Jeffrey Vetter, Ray Wang, Fred Alyea, Bill Ribarsky, and Mary Trauner. Framework for collaborative steering of scientific applications. *Science Information Systems Newsletter*, IV(40):19–23, 1997.