Declarative Flow Control for Distributed Instrumentation

B. Parvin, G. Fontenay, and J. Taylor
*Computing Sciences
Lawrence Berkeley National Laboratory
Berkeley, CA 94720

D. Callahan
Life sciences
Lawrence Berkeley National Laboratory
Berkeley, CA 94720

Abstract

We have developed a "microscopy channel" to advertise a unique set of on-line scientific instruments and to let users join a particular session, perform an experiment, collaborate with other users, and collect data for further analysis. The channel is a collaborative problem solving environment (CPSE) that allows for both synchronous and asynchronous collaboration, as well as flow control for enhanced scalability. The flow control is a declarative feature that enhances software functionality at the experimental scale.

Our testbed includes several unique electron and optical microscopes with applications ranging from material science to cell biology. We have built a system that leverages current commercial CORBA services, Web Servers, and flow control specifications to meet diverse requirements for microscopy and experimental protocols. In this context, we have defined and enhanced Instrument Services (IS), Exchange Services (ES), Computational Services (CS), and Declarative Services (DS) that sit on top of CORBA and its enabling services (naming, trading, security, and notification) IS provides a layer of abstraction for controlling any type of microscope. ES provides a common set of utilities for information management and transaction. CS provides the analytical capabilities needed for online microscopy. DS provides mechanisms for flow control for improving the dynamic behavior of the system.

1 Introduction

The current trend in telepresence research is to bring experts and facilities together from geographically dispersed locations [3, 5, 10, 6]. The natural evolution of this research is to couple declarative representation with object oriented techniques for maximizing reusability and flexibility, increasing abstraction, and reducing maintenance cost. A declarative approach provides a semantic that is concise and deep. For example, rule based systems provide the basis for dynamic interaction between the users and applications with an abstraction that is much like predicate logic and more understandable than traditional approaches. In this context, the logic (the declarative part) can be changed on demand, but the methods for applying the logic to the state of the system remain intact. In contrast, any change to the logic, in the present framework, requires deep changes in the procedural code that are difficult to maintain. Furthermore, different scientific experiments require different sets of logic. Thus, the design must be extensible.

The design themes are functionality, scalability, and performance. We are also interested in interactivity, which is achieved through the best effort with most commercial ORBs or Web servers. Functionality refers to what and how an instrument does something and how well system resources can be managed and accessed. Scalability refers to the number of instruments, vendor-specific desktop workstations, analysis programs, and collaborators that can simultaneously attach themselves to the system. Performance refers to how well system resources are being utilized. Our testbed includes several electron and optical microscopes that are located at Berkeley Lab (LBNL), Oak Ridge National Laboratory (ORNL), and the University of Illinois, with applications ranging from material science to cell biology. Our system is named DeepView, which has been installed at several institution. The interface to the microscopy channel and a listing of various instruments are shown in Figure 1. The channel is tightly coupled with the OMG-defined. Naming and Trading services for binding and resource discovery. From the user's perspective, we established a set of desirable requirements in terms of functionality, scalability, interactivity, safety, and security. From the designer's perspective, we abstracted these requirements into four categories of services: Instrument services (IS), Exchange services (ES), Computational services (CS), and Declarative services (DS). These services sit on top of CORBA, OMG defined services, and Web servers. IS provides a layer of abstraction for controlling any type of microscope or simulation software. Simulation aims at generating a representation based on physical properties of a system and its relationship with respect to an observation mode. ES provides a common set of utilities for information management and transaction. CS provides the analytical capabilities needed for online microscopy and problem solving. DS provides flow control and required XML-based features for improving the dynamic behavior of the collaborative infrastructure.

*This research is supported by Director, Office of Science, the Office of Biological and Environmental Research, and the Office of Advanced Scientific Computing Research, Mathematical, Information, and Computational Sciences Division of the U. S. Department of Energy under Contract No. DE-AC03-76SF00098 with the University of California. The publication number is LBNL-47408. E-mail: parvin@media.lbl.gov.

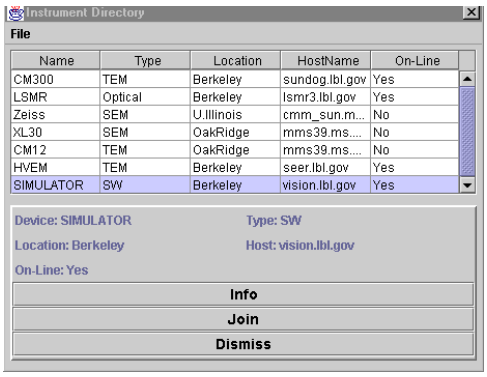


Figure 1: User's view of the microscopy channel.

CS offers an extensible array of tools for visualization, model recovery, and comparative analysis of observed and simulated data. Model recovery is an inverse problem-solving process that attempts to (a) link a specimen's behavior to external stimulation through feature extraction, archival, and data mining or (b) construct a 3D geometric model of an object through user interaction. In general, model recovery is a computation-intensive algorithmic process requiring the extensive support of high-performance computing and low-latency network infrastructure.

The next section of the paper summarizes recent related work in collaboratory computing. Section 3 describes software architecture, ongoing scientific experiments and their corresponding computational needs. Section 4 concludes the paper.

2 Related Work

Previous systems fall under two categories: telepresence [3, 5, 7] and collaborative frameworks [6]. Telepresence research has focused on remote functionality of the instrument and the necessary automation for large scale data collection and analysis. In general, these systems ignore many of the scalability issues that we have been advocating [6]. With respect to the collaborative framework, a taxonomy of existing systems is given below.

- UC Berkeley's MASH project [4] uses MBone tools in a heterogeneous environment to develop scalable multimedia architecture for collaborative applications in fully distributed systems.

- NCSA's Habanero project provides smooth management and simultaneous distribution of shared information to all clients in a component-based, centralized system written primarily in Java.

- Rutgers University's DISCIPLE uses a CORBA framework for distributed access in a service-based, centralized system for enforcing shared virtual space.

- Sun Microsystem's Java Shared Development tool kit enables collaborative-aware Java code to send data to participants within a communication session. It supports three types of transport protocols: TCP/IP socket, light-weight reliable multicast, and remote invocation method. In this framework, all objects are manageable and collaboration occurs within a session that includes channel, token, blobs, and listener.

- The University of Michigan's Upper Atmosphere Research Collaboratory (UARC) is a web-based distributed system (written mostly in Java) that collects data from over 40 observational platforms for space physics research for both synchronous and asynchronous collaboration. In this system, data suppliers publish their data on a data-dissemination server. Clients then subscribe to receive the desired information.

Our approach combines service-based distributed architecture with the declarative framework to maximize the use of commercial middleware and emerging new technologies. This is based on an OMG-defined CORBA framework [with an Internet Inter-ORB Protocol (IIOP)], Web Servers, and emerging XML-based specifications. CORBA provides virtual distributed containers for objects. These objects can then be implemented in any language; e.g., Java, C++. Furthermore, OMG has defined a number of enabling technologies for decoupled communication, object localization and resource discovery, and security. The main advantages of using CORBA are that (a) it is not restricted to the Java language, (b) it is available on multiple platforms, (c) it supports a rich class of enabling services, and (d) it supports real-time applications [8] under a newly adopted standard.

2.1 Flow control

Over the last 8 years, several work flow models have been proposed through various working group. The intent is to define an abstract peer-to-peer collaboration that can operate across the Internet. The first such specification by the Workflow Management Coalition (WfMC), defined a set of state model representing a process in terms of running, terminating, and completion. The standard defines the contents of requests and responses that are extensible and leverages well-known concepts such as property objects or java beans. WfMC was later used by OMG to define a framework for distributed workflow management. The framework defines object models for workflow requests, a registry, workflow process management, and workflow activities. The simple workflow access protocol (SWAP) and Wf-XML are the evolution of the OMG initiative. SWAP uses http protocol that renders interaction between workflow applications. The lesson learned from SWAP led to a Wf-XML specification with new features such as synchronous and asynchronous interactions, data typing, and timing. Our system uses a Wf-XML to specify the workflow model in a distributed environment. Wf-XML is the only specification that meets the

science requirements for annotating time series data with synchronous and asynchronous control.

3 Software Architecture

Our system uses an extensible object-oriented framework so that applications can be rapidly assembled, maintained, and reused. These objects may reside on any host and can be listed, queried, and activated in the system. The architecture illustrated in Figure 2 bridges the gaps between different services that may reside at any node in a distributed system.

Our system consists of three service categories that interact with the ORB, Adaptive Communication Environment (ACE), and Web servers: Instrument Services (IS), Exchange Services (ES), and Computational Services (CS). A brief review of Enabling Services is provided in this section.

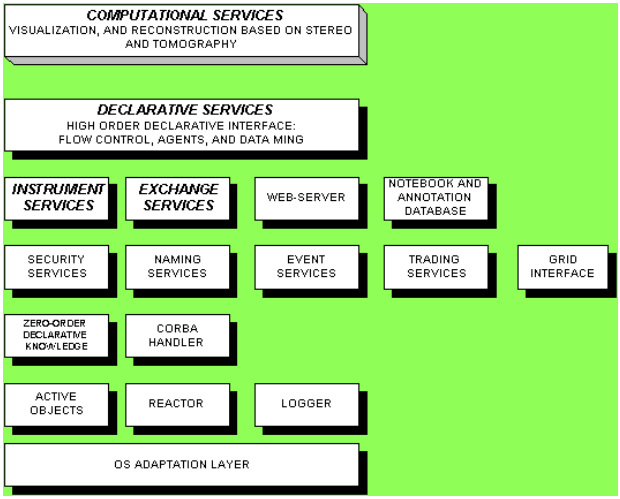


Figure 2: Architecture of DeepView

3.1 Enabling Services

Enabling services include OMG defined services (Naming, Trading, Security, and Notification) as well as Web based technologies such as Web Servers and JSP. The Naming Service binds a name to an object and allows that object to be found subsequently. It behaves much like the "White Pages." The names are resolved within a naming context that is organized as a graph. The naming context is an object that stores name binding for objects, and it is essentially a table. The Trading Service provides facilities for dynamic object discovery. The trader stores a description of the service along with object reference, and behaves much like the "Yellow Pages." It provides an advertisement service, policies, and a matching engine through an OMG-defined constraint language. The constraint is a boolean expression that is somewhat similar to an SQL interface. The constraint language provides boolean, arithmetic, and comparison operations to locate a particular object or resource based on its properties. The requirement for resource management and brokering has been well documented [1] through the use of either LDAP or relational databases. The CORBA Naming and Trading Services are an alternative approach that is reliable (for writing), and well supported by the commercial vendors. While the Naming Service is hierarchical (much like a UNIX file system), the Trading Service is flat. The Security Service is based on the secure socket layer (SSL), which provides authentication, privacy, and integrity for TCP-based connections. SSL uses RSA public key encryptography for authentication, where each application has an associated public key and an associated private key. In this context, data encrypted with the public key can only be decrypted with the private key, and data encrypted with the private key can only be decrypted with the public key. The Notification Service is a replacement for Event Services. Several communication models are supported by CORBA:

- The first model is based on the standard CORBA invocation model of two-way, one-way, and deferred synchronous interaction. Although this model simplifies distributed processing, it lacks asynchronous message delivery and does not support group communication, which can lead to excessive polling.
- The second model uses COS Event Services that provide decoupled communication between suppliers and consumers. The key concept in this service is the event channel, which can assume a variety of design patterns depending on the model of collaboration among different components [9]. The roles that the event channel can play include (a) a notifier for the push/push model, (b) a procurer for the pull/pull model, (c) a queue for the hybrid push/pull model, and (d) an intelligent agent for the hybrid pull/push model. Presently, only the push/push model with typed and untyped events is supported. This service allows clients to register with events of interest and filter incoming events. A unique feature of the Notification Service is that it supports quality of service (QoS) and various policies to enforce it. The underlying transport protocol can be either TCP or reliable multicast. Reliable multicast is supported by OrbixTalk, which provides assembly, sequencing, and ordering of IP multicast packets for enhanced network utilization.

In our system, all synchronous communications (for collaboration) are performed through the event channel, and all asynchronous operations are conducted with CORBA two-way and one-way communication. The Naming and Trading Services are used for object localization and its required resources. The Naming Service is organized as a hierarchical tree structure for modular organization of objects. These services are federated, with each physical site maintaining its own catalog of information. However, this view is hidden from clients. The key advantages of a federated organization are (a) improved reliability (when a single server becomes inaccessible), (b) improved performance (where different servers can work in

parallel), (c) improved scalability (where persistent information is distributed on multiple hosts), and (d) improved administration boundaries. Naming and Trading Services are a powerful mechanism for resource discovery, brokering, and subsequent load balancing.

The SSL handshake is initiated by a client sending a message to the server. The server responds by sending its X.509 certificate. The client extracts the public key from the certificate and encrypts a session key. The server uses its private key to decrypt the session key and application data. Additionally, the server requests the client certificate to resume a previously established handshake.

3.2 Instrument Services

The key to rapid integration of any instrument into the system is declarative annotation of instrument control and detectors. An instrument is partitioned into devices, each of which may have a number of properties. A DTD description was developed, and the corresponding XML information has been constructed and stored for each instrument. The devices and properties can then be queried through a Web interface. During initialization, the servers access the persistent storage to configure themselves for a particular instrument. In this context, Instrument Services provide a scalable means of collaborative instrument control and interaction through three objects: instrument, instrument factory, and an abstract action class. See Figure 3. An instrument consists of a set of devices (e.g., controller, detectors) that are advertised through the Naming Service. Each device has a list of properties. For example, the controller may include focus, shift, and tilt properties. These properties and their corresponding attributes can be queried and manipulated through instances of instrument and action objects. An action consists of three simple interfaces: get, set, and cando.

All actions occur within a managed session. When a client instantiates an action, it passes an object that uniquely identifies itself. By associating a particular user with each action, the server may queue and/or prioritize the processing of its services in the collaborative environment. The design of IS is partially influenced by the Object Property Service (OPS) as defined by OMG. OPS provides a mechanism to associate objects with typed name-value pairs. These objects can then be manipulated through set and get methods. In an instrument, the value of a control parameter can change either by natural drifts in the system or when it is set to a new value by a principal client. The design of the server is multi-threaded. One thread autonomously scans the property values (of each device) every n seconds. The second thread simply changes the value of a property through a set operation. In both cases, any changes in the state of the system are recorded and then broadcast to all clients. This design is scalable, since properties and their corresponding attributes are stored in a persistent configuration file. In other words, addition of new instruments will not require any changes to the CORBA IDL representation, and thus no changes will be needed to the clients (with the exception of instrument-specific changes to the GUI). Each new instrument needs to define a "plug-in" for the proposed architecture. Another utility of IS is its use as a front end to a simulation engine or modeling system. In this context, a physical instrument and a simulation engine behave similarly. They both have control parameters, and they both generate blobs of data.

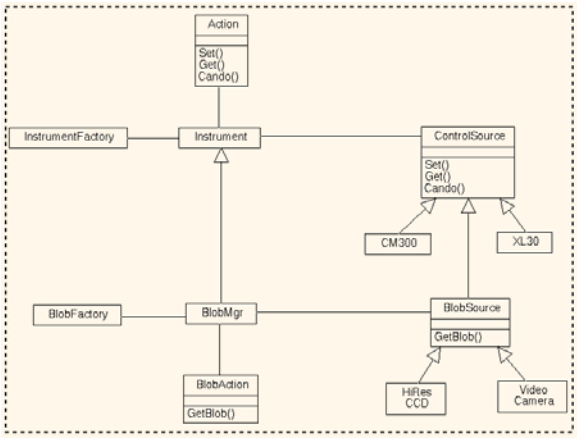


Figure 3: Relationship of key objects for Blob Manager and Instrument Manager. Each additional instrument requires a plug in for ControlSource and BlobSource. The details of ControlSource and BlobSource are hidden at the IDL level.

3.3 Exchange Services

Exchange Services provide a set of objects for information exchange between collaborators, instruments, and application programs. This set consists of:

- The session manager (SessionMgr) object, which provides a listing of active users and a policy for sharing an instrument among multiple collaborators. This policy empowers the current "principal" to pass the control to another user. The instrument has a local operator who can override current the principal by assigning the instrument to a third party. Each time a new user joins the system, his or her presence is broadcast to all the other clients. Likewise, when the user leaves, he or she is removed from the list of active clients.

- The blob manager (BlobMgr) object, which provides an efficient means of transferring bulk data between various objects. It uses the same IDL that is used by IS, but it is extended to handle blob data. On the server side, the detailed implementation of BlobMgr has a three stage pipeline for high throughput. The pipeline architecture has shown a throughput of 15 frames/sec for compressed data over the wide area network. Furthermore, the recent implementation of ORBs from Iona and TAO has shown zero memory copy with similar performance to the UNIX socket over the high-speed network. As a result, all the bulk

data transfer is implemented as a sequence data type. The server side of the blob manager is designed in such a way that the number of blob objects (detectors) are hidden from the IDL. As a result, each time a new detector (blob generator) is added to the system, no changes to the IDL are made. A client can query various blob sources in the system and register to receive data from a specific detector. The blob object is managed by the session manager.

- The shared space manager, which provides the necessary services for clients to exchange chat messages, graphics overlays, and images among multiple collaborators. Message sharing can be private or public. Public messages are broadcast to all collaborators, while private messages are sent to a subset of collaborators. This component is tightly coupled to the session manager for private messages.

3.4 Declarative Services

Declarative services refer to zero-th or higher order forms of representing information about a particular experience. This representation is generated by a user. Such a framework needs to incorporate several types of declarative notions. Our present focus has been on flow control, which is modeled after Wf-XML specification. The workflow engine provides interoperable functions in terms of operations. Each operation may pass a set of request parameters and return a set of response parameters. Operations are divided into different groups so that they can be identified by their context. There are three primary groups of operations, which are named *ProcessInstance*, *ProcessDefinition*, and *Observer*. The ProcessDefinition is a factory for creating an instance of a service, which can be referenced by interoperable services. The ProcessInstance corresponds to the actual invocation of ProcessDefinition and maintains its own identifier. The Observer is essentially a notifier pattern that informs other operations of any state changes. The interaction of these operations is shown in Figure 4.

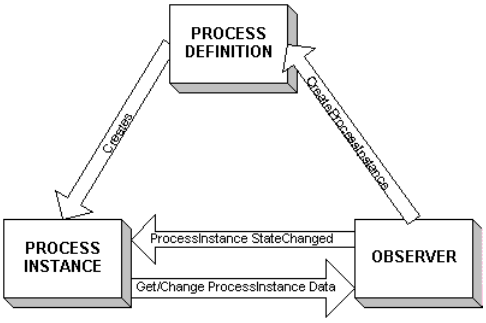


Figure 4: Interaction of operations in Wf-XML.

In our system, collaborators communicate with servers through OrbixWeb (a Java version of Iona's ORB), where remote GUI objects are implemented as Java beans. In theory, Java provides scalability on different types of desktops. However, some modification is needed for porting the GUI across multiple platforms. The GUI component aims to provide needed functionality for a particular type of experiment. A fairly detailed abstraction exists for in-situ and high-resolution microscopy. It also provides a logbook where the state of the instrument can be traversed to a previously known state. The GUI manager has two components: a generic interface for common instrument control, and a set of specialized components that are instrument specific.

The Java client is based on the model view controller (MVC) pattern. The implementation uses the Java event model. The model provides a proxy for a remote data source, and provides three kinds of interfaces: command, data, and notification. The command interface is for requesting changes to the model's data. The data interface is for requesting the model's data. The notification interface is for notifying listeners about changes in the model's data as a result of external events. The key models (event sources) in the DeepView client are BlobSource, ControlSource, MessageSource, and SessionSource. Each of these sources can generate user-defined events. The view components receive input from the user and forward it to an appropriate controller. It displays data to the user by requesting it from a model. additionally, the view receives notification from the model. The basic views in the system are BlobCanvas, Device and PropertyTable, and SessionPanel. The controller defines the relationship between input from the view and the action made against the model. It follows an observer pattern. The controllers use the three interfaces provided by the model to maintain a consistent view. The controllers are BlobPlayer, Instrument Manager, and Session Manager.

Figure 5 shows a specific GUI that is designed for a scanning electron microscope at the Oak Ridge National Laboratory. The output of the blob manager, session manager, shared space manager, and the instrument manager are presented to the end user. Figure 6 shows the interaction between clients and various services through the event channel. At the instrument site, there are three channels for Instrument State, Blob Manager, and Session Manager. The Blob Manager samples the output of the detector periodically and pushes a compressed image to the event channel. This is the most active channel, and it has been implemented with a three stage pipeline architecture. The other two channels simply notify the client application of any changes. This design is service based, decoupled, and distributed. The behavior of the event channel indicates that it broadcasts data at the rate of the client with the least amount of network bandwidth. Thus, to avoid penalizing clients with high network bandwidth, the system maintains a pool of event channels for the Blob Manager. Each channel has a different updating frequency, and its characteristic is registered with the Trading Service. The clients then measure their bandwidth and connect themselves to an appropriate channel with matching impedance. With the ex-

ception of event channels corresponding to the BlobMgr, all other event channels run in a secure mode.

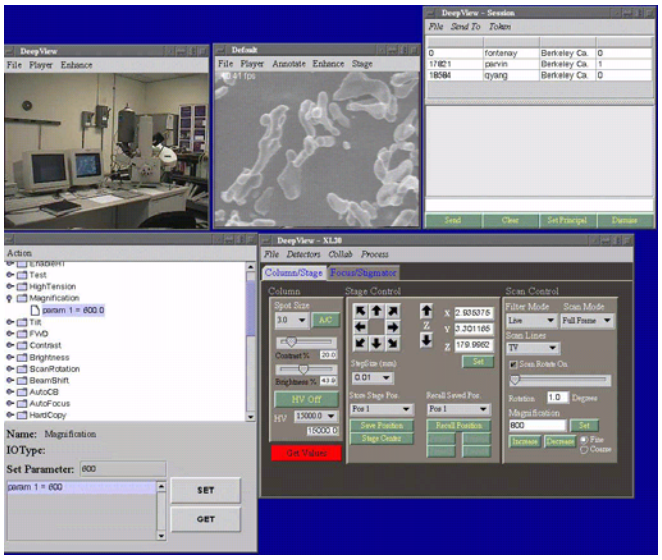


Figure 5: Collaborative view of the DeepView shows the shared view space, which includes image, session manager, and a listing of devices and properties at the instrument site. Changes in the properties are recorded and broadcast to all clients.

3.5 System Views

DeepView maintains three views of the system: the naming view, content view, and meta view. Naming view leverages the vendor-supported GUI for advertising object references and their corresponding hierarchy. Content view shows the current state of the instrument in terms of devices and their associated properties in the instrument control panel. Meta view uses XML to represent the relationship between devices, properties, and their attributes. This view is used to annotate actions performed on the instrument and for logging information into the archival system.

3.6 Computational Services

A number of computational components have been integrated to enhance instrument operation and science experiments through high performance computing [5, 2]. Two new computational features are included in this paper that address issues in material science as well as cell biology.

3.6.1 Stereo reconstruction

From a functional perspective, it is often difficult to examine 3D structural details that may be present on a specimen holder when observed with scanning electron microscope. We have developed an algorithm to view three dimensional

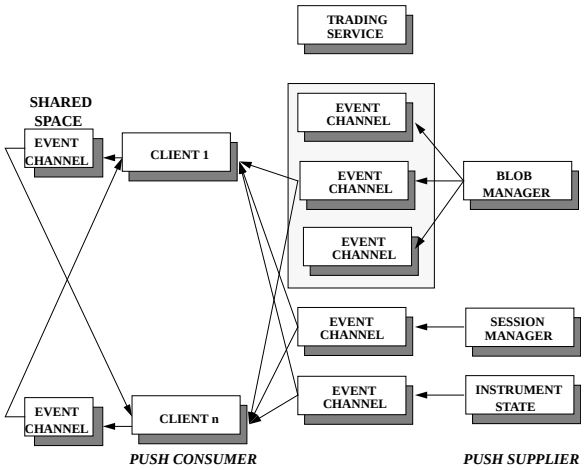


Figure 6: Interaction between producers and consumers through event channels. Each consumer measures its available bandwidth and connects itself to a blob manager event channel with matching impedance. The data rate for session manager and instrument manager is low and no impedance matching is needed. The underlying transport for the event channel can be either TCP or reliable multicast.

structures by tilting the specimen. The details of this approach are beyond the scope of this paper; however, the workflow model for this *operation* is included below as well as an example of 3D reconstruction (in Figure 8).

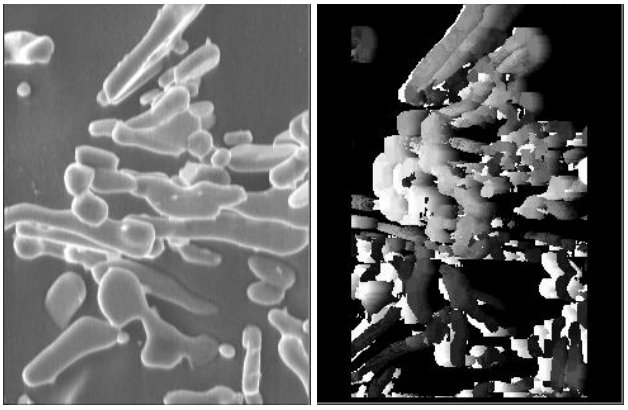
3.6.2 Kinetic uptake and retention factors at cellular level

An inverted optical microscope has been used to study the kinetics of uptake and retention in living cells. In this case, particular compounds of interest are injected into the cell environment under computer control. Our system can be programmed with user defined recipes that indicate concentration of various compounds (at different time points) being injected into the cell environment, sampling rate for collection of images, and various imaging parameters. The system uses a workflow model to capture images periodically, segmenting those images, measuring cellular responses for a field of several hundred cells, and logging those responses into an archival system. These responses allow direct measurements of kinetic uptake and retention factors for each cell line for subsequent comparison. The archival system can then be browsed and queried through a Web based interface. Examples of browsing raw data, segmentation results, and corresponding metadata (computed responses over a 2.5-hour period) are shown in Figures 9, 10, and 11.

```
<WfMessgae Version = ``1.0'' >
        <WfMessageHeader>
                <Request ResponseRequired= ``Yes''/>
        <Key> http://vision.lbl.gov/wfprocess/stereo </key>
        </WfMessageHeader>
        <WfMessageBody>
        <CreateProcessInstance.Request StartImmediately= ``true''>
                <Observerkey> http://vision.lbl.gov/wfprocessor
                </ObserverKey>
        <ContextData>
                <DisparityRange>
                        <Max> 24 </Max>
                        <Min> 0  </Min>
                </DisparityRange>

                <WindowSize>
                        <Max> 15 </Max>
                        <Scale> 2 </Scale>
                </WindowSize>
        </ContextData>
        </CreateProcessInstance.Request>
        </WfMessageBody>
</WfMessgae>
```

Figure 7: Process invocation by the client side of workflow control for stereo reconstruction. A workflow observer model runs on the remote server for reconstruction.



(a) (b)

Figure 8: Stereo Reconstruction: (a) one of the two views used for stereo reconstruction shows no apparent depth cue; (b) the 3D map shows depth cues for different pieces of specimen.

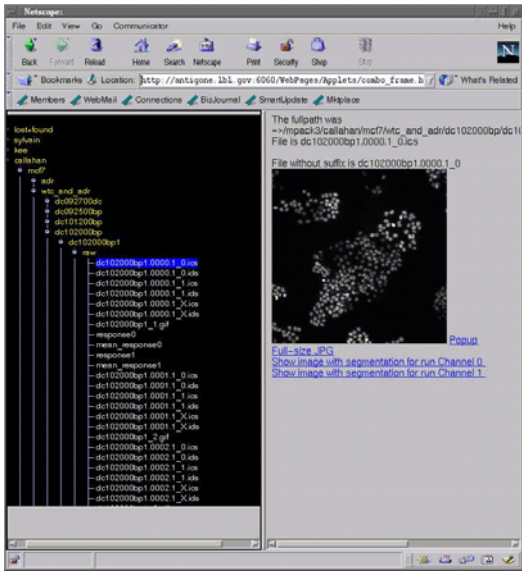


Figure 9: Tree structure representation of raw data and viewing of a particular image.

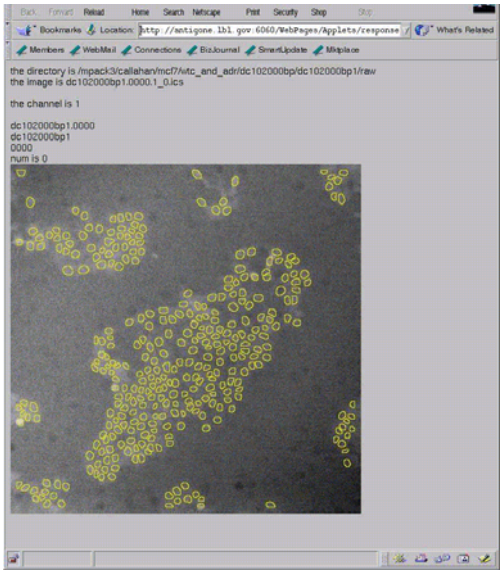


Figure 10: Segmentation results for a field of nuclei selected from Figure 9. This is an active page, where the user can click on a specific nucleus and observe its response over the entire experimental cycle (approximately 2.5 hours).

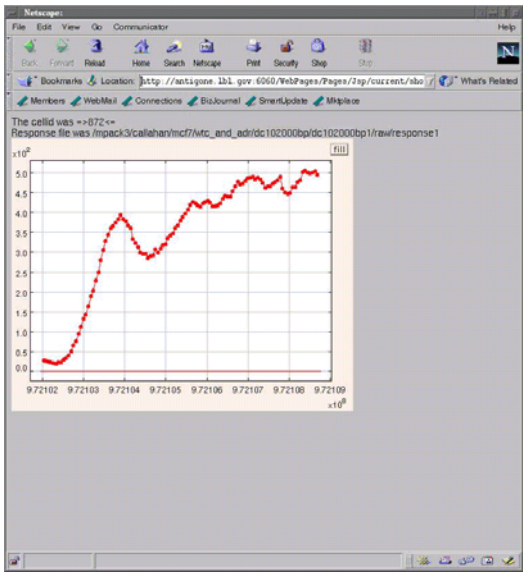


Figure 11: Response of an individual nucleus as a function of time as various compounds are injected into the cell chamber at different time points.

4 Conclusion

A channel for distributed microscopy has been implemented to meet the requirement for synchronous and asynchronous collaboration. We have leveraged OMG-defined services, Web servers, and XML based workflow specifications to construct four additional services for instrument control, collaborative management, and analytical capability. Our approach aims to leverage common off-the-shelf middleware software to exploit economies of scale. However, a key design feature of our system has been scalability. We view scalability not only as extensible software interfaces, but also in how experimental parameters can be varied through declarative workflow models. We have applied our design to problems in material science as well as in cell biology.

References

[1] C. Baru, R. Frost, R. Marciano, R. Moore, A. Rajasekar, and M. Wan. Metadata to support information-based computing environment. In *IEEE Conference on Meta Data Computing*, 1997.

[2] G. Cong and B. Parvin. Shape recovery from eqaul thickness contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:1055–1061, 2000.

[3] M. Hadida-Hassan and et al. Web-based telemicroscopy. *Journal of Structural Biology*, 125:229–234, 1999.

[4] S. McCanne. Scalable multimedia communication using ip multicast and lightweight sessions. *IEEE Internet Computing*, 3:33–44, 1999.

[5] B. Parvin, J. Taylor, D. Callahan, W. Johnston, and U. Dahmen. Visual servoing for on-line facilities. *IEEE Computer Magazine*, 1997.

[6] B. Parvin, J. Taylor, and G. Cong. A collaborative framework for distributed microscopy. In *IEEE Conf. on Super Computing*, 1998.

[7] C. Potter and et al. Leginon: A system for fully automated acquisition of 1000 electron micrographs a day. *UltraMicroscopy*, 77:153–161, 1999.

[8] D. Schmidt, D. Levin, and S. Mungee. The design of the tao real-time object request broker. *Computer Communications*, 21, 1998.

[9] D. Schmidt and S. Vinoski. Object interconections–the OMG event services. *SIGS C++ Report*, 1997.

[10] Young S.J. and etal. Implementing collaboratory for microscopic digital anatomy. *Int. Journal of Supercomputer Applications and High Performance Computing*, pages 170–181, 1996.