Sharing Objects over the Internet: the Mushroom Approach

Tim Kindberg, George Coulouris, Jean Dollimore and Jyrki Heikkinen

Department of Computer Science, Queen Mary and Westfield College,
University of London, Mile End Road, London E1 4NS, UK
Email: Tim.Kindberg@dcs.qmw.ac.uk

Abstract

This paper reports on the Mushroom project[1]. The project is developing a software framework for collaborative working and user interaction on the Internet. The paper illustrates some of the required features in the context of an application scenario and outlines those aspects of the Mushroom system architecture that support the sharing of information. The project's motivation is to address the problems of coordinated user interaction, distribution and privacy. *Mrooms* are interactive environments for groups of collaborating users. They provide a shared space in which users are aware of one another while working on shared objects. Their boundaries provide a triggering mechanism for consistency and privacy checks. In contrast to other systems with room-based metaphors, Mushroom focuses on a scalable and flexible system architecture using replicated state, group communication and event-based updates.

1. Introduction

The World Wide Web was designed for information browsing, but recently the interest of the Internet community has begun to focus on interaction. There is an increasing requirement for an infrastructure to enable user interactions and collaborations based around mutual goals and shared data. Geographically distributed organisations require this support for their internal operations as do groups of organisations that collaborate. Moreover, Internet users in general could benefit from being able to 'meet' when they access the same data, allowing them to interact and form collaborations based around their mutual interests.

Mushroom is a software framework that supports the dynamic creation and management of *Mrooms*, which are shared environments for collaborative work and containers for shared objects. Users access Mrooms through links on World Wide Web pages and they navigate between them by following links within Mrooms. Access control is applied, providing task-related privacy and integrity constraints. In Mrooms, users can share applications and information objects such as documents and whiteboards. Users also share tools for awareness of, and communication with other users in the Mroom. Mrooms are normally persistent, and users can interact synchronously (in the same room at the same time) or asynchronously (users occupy the room at different times, but observe one another's changes to information objects). Further motivation is given in [1].

Mrooms contain representations of collaborating users, the information objects on which they are working and links to other Mrooms. The Mushroom user interface offers a consistent view of each Mroom to all the users in the Mroom and provides operations for altering the contents of Mrooms. The view is desktop-like by default, and it includes iconic representations of the users in the Mroom and of the shared objects and links that it contains. Objects may be 'dragged' from one Mroom to another, with the default semantics that a copy is placed at the destination[2]. Objects may exist in more than one Mroom; this is achieved through a different user interface option which creates an alias to an object and places it in an Mroom. The iconic representations of users and objects are dynamic – changes in their colour and contents are used to exhibit their current attributes, e.g. user active in Mroom, object in use.

The Mushroom architecture supports the collaborative use of conventional interactive tools, but for synchronous working an event-based synchronisation mechanism is supported. For example, a Mushroom aware Web browser would handle *go to page* events, enabling a user to demonstrate a set of pages to a group of other users. Each user's instance of the browser would show the pages selected by the demonstrator.

A key feature of the Mushroom model is the exploitation of the room model to implement privacy, integrity and concurrency control mechanisms. Attempts to transfer objects or users across any Mroom 'boundary' trigger a variety of validation procedures. These checks may be specified on a per-Mroom basis, or by default when the Mroom is created from a template. They may be automatic or they may involve the intervention of users. The user interface supports a 'doorstep view' of users and objects awaiting user intervention before entering a room.

The Mushroom framework is currently under development in Java. We provide a helper application for Web browsers, so that Mrooms can be linked to and accessed from conventional Web pages. At the system level, the Mushroom framework models Mrooms and the shared objects that they contain by a distributed object replication scheme. Significant changes to the state of Mrooms and their contents are communicated to affected users' workstations with appropriate message ordering and reliability constraints to ensure consistent views of Mrooms and their contents.

In Section 2 we illustrate some features of the Mushroom model and user interface through the presentation of an application scenario developed collaboratively with an industrial partner. In Section 3 we state the system requirements for Mushroom and discuss the use of object replication, events and naming in the sharing of real-world information objects. In Section 4 we place Mushroom in the context of other work and describe its current status.

[1]The work reported here is funded in part by the UK Engineering and Physical Science Research Council under grant GR/K73674.

[2]This is analogous to the Macintosh and Windows desktop semantics in which documents are copied when moved to a new disk volume.

a. Presentation Mroom – for pre-contract presentation and negotiation

Shared objects	*Properties/behaviour*
Web pages	Viewed with a 'Mushroom aware' Web browser. Changes to pages are performed outside the meeting Mroom using a conventional single-user editor, then dragged back into the Mroom.
Other documents	Viewed with a 'Mushroom aware' viewer; or a conventional word-processor, in which case the users would be responsible for maintaining the consistency of their views. Access restrictions can be specified at the time of introducing a document and will be enforced by Mushroom.
Whiteboard	A standard Mushroom object type.

b. Project Mroom – for private working by contractor staff

Shared objects	*Properties/behaviour*
Client requirements (alias)	This refers to the document in the Progress Meeting Mroom.
AMS Web standards	Can be changed only by Project Leader.
Project deliverables	Each item (HTML page, etc.) is the responsibility of a team member. They may replace it with a new (validated) version.

c. Progress Meeting Mroom – for joint working by client and contractor staff

Shared objects	*Properties/behaviour*
Client requirements	Viewed by all participants. Changed only by an authorised client representative. Changes are logged.
Project deliverables (alias)	Representing the current state of the product. These would be accessed as in the Project Mroom, but client representatives might be allowed some limited update capabilities. This requires concurrency control to ensure that their updates are synchronised with those of AMS staff.

Figure 1. Shared objects and their properties in the Web site design scenario.

2. Scenario

In this section we illustrate the use of shared objects in Mrooms in the context of a scenario for the task of cooperatively designing Web sites. Our scenario is based on the experience of an industrial partner in the Mushroom project, Active Media Solutions (AMS), whose business includes the design of Web sites for clients.

Currently, the task of designing a Web site involves a variety of meetings at which collaborative work occurs between Web developers, graphic designers and the client. The initial stage involves a meeting between one or more AMS staff and a similar number of client representatives; AMS aims to convey the potential of a Web site to the client and to suggest some options. In the next stage the client representatives provide AMS with an initial specification of their requirements. The AMS developers formulate an initial structure and content for the design and a graphic designer is brought in to do screen layout and add graphics. Subsequent meetings are held between AMS, the client and the designer to demonstrate and discuss the product under development.

We now suggest how this task could be carried out in meetings using a set of Mrooms containing shared objects. This does not preclude additional face-to-face meetings, e.g. after the initial meeting, in order to establish a better social *rapport*. The collaborative environment reflects the client–contractor relationship, allocating different Mrooms for pre-contract presentation and negotiation, private working by contractor staff and joint working between client and contractor staff.

In Figure 1, we list the shared objects that would appear in each Mroom and note the most significant aspects of their properties and behaviour. To save space, we have listed objects that appear in all of the Mrooms only in Figure 1a. In addition, all Mrooms contain voice and possibly other communication channels to which each user may connect.

Presentation meeting: Using a Web browser, each AMS client representative follows a link from the AMS Web page to an AMS Presentation Mroom. In the Mroom the meeting participants are represented by icons – labelled photographic images. Admission to the room is controlled by the Meeting Organiser who invites users to the meeting by dragging copies of their icons from their home Mrooms to the Presentation Mroom in advance of the meeting. If a late participant joins the meeting, the organiser can validate them on the 'doorstep' and admit them if they meet appropriate criteria. Users are assigned to 'roles' by the organiser. For this task, the roles would be Meeting Organiser, AMS Staff and Client Staff. Roles determine the rights of users with respect to the shared objects in an Mroom.

On entry to the Presentation Mroom, each user's iconic representation is highlighted to show that they are 'present'. If they have switched on their audio channel to that Mroom, they can hear and be heard by the other users in the Mroom and this is indicated in their icon. If necessary, users who do not join the meeting promptly can be reminded to do so by using a Pager object to 'bleep' and leave an urgent message for them on the workstation screen.

Once the meeting has convened, an AMS representative demonstrates the features of some example Web pages and discusses them with the client representatives. The Web pages can be updated on the fly by the demonstrator or her assistant, enabling her to customise the demonstration to the client's needs. The client representatives may bring documents and other information about their organisation into the Mroom and refer to them or show them to the AMS representatives. A simulated whiteboard is used by all participants to assist in the discussion of potential page layouts and graphics.

When the meeting closes, the meeting organiser may decide to retain the current configuration of the Mroom for future meetings with the same client. The user representations and shared objects would remain in the Mroom with suitably modified icons to show their currently passive status. Unlike rooms in the real world, new Presentation Mrooms for other projects can be created at will.

Project work: AMS technical staff work with a graphic design consultant to produce the Web site required by the client. The main problem here is to maintain consistency between the portions of the project that are developed by different members of staff. Staff develop Web pages and other documents 'off-line' using conventional tools and deliver them to the Project Mroom. The Project Mroom contains only 'approved' versions of Web pages and other project information. This is achieved by enforcing a validity check whenever an object is dragged into the Project room. The checks may be automatic (e.g. an HTML syntax check), or manual (e.g. the Project Leader is notified that an object is on the 'doorstep' of the Project Mroom, and she 'eyeballs' it before authorising its entry to the room.

Progress meetings: A Progress Meeting Mroom is used for regular project meetings with a subset of the participants from the initial presentation meeting. This could be set up by creating a new Mroom from a standard template and copying users and objects from the Presentation Mroom as required.

3. System support for object sharing

The application scenario given in the previous section indicates the need for a powerful and flexible underlying distributed system architecture. The main system requirements and issues for Mushroom are:

Object replication: An important goal of the Mushroom design is scalability to many users distributed across the Internet sharing many objects. The system architecture is based on object replication [2, 3], in which shared objects are replicated at users' workstations for efficient access. This also enables users to continue with some work even when disconnection occurs. Objects with a high availability requirement are also replicated at several servers.

Persistence: Many objects in Mushroom (in particular, objects representing Mrooms themselves) are required to persist even when no users are accessing them, and to survive machine crashes. Therefore servers must keep copies of these objects on persistent storage.

Event management: An *event* is a set of attribute values which describes a new state of affairs. It is used to describe a change that is to be made to the set of replicas of an object; or, it is used to convey information to users about a change that has occurred, for example, the addition of an object to an Mroom. Our requirement is to deliver events to just those sites that need them, and to deliver them with reliability and ordering semantics that are appropriate to the consistency semantics of the objects affected.

Naming: We require human-readable names for Mrooms and the objects within them (including users) so that users can sensibly discuss shared objects and inform one another about them. Users can read objects from outside Mrooms (subject to permission) and so we require absolute (global) names for objects, as well as names used in the context of an Mroom. Mrooms can fall under the administration of different sets of organisations as they evolve, and a further requirement is to be able to continue to access objects using the same names. This feature is noticeably absent from URLs [4].

Integrity and concurrency control: We need to be able to enforce concurrency and integrity constraints on a per-class or per-object basis, supporting both optimistic and pessimistic forms of concurrency control, and general integrity constraints.

Openness: Users can install new types of shared objects in Mrooms, as Java applications. We also support the import of traditional, non-group aware objects such as word-processing documents, for dissemination via Mrooms.

Security: Users require varying degrees of privacy of information and secure control over updates to objects. Threats posed to these requirements derive from the exposed communication channels of the Internet, over which we transmit object state and events; the potential for a user to make illegal updates to state stored on her computer; and the loopholes and lack of security guarantees in Java as it is currently implemented [5]. Also, there may be varying levels of trust between users inside an Mroom.

Space allows us only to describe our approach to the first four of the requirements defined above.

3.1 Replication and persistence

To provide high availability, the current state of shared objects is replicated at users' workstations wherever the objects are in active use. To support persistence, shared objects are replicated at a set of servers whose function is to maintain persistent copies of their current authoritative state. Persistent copies are maintained at multiple servers for increased reliability and performance. Thus the Mrooms and objects associated with a cooperative task will be replicated at a small, stable set of servers and at a set of workstations whose membership changes as users come and go.

We use the term *session* to refer to a set of Mushroom processes that are maintaining replicas of some collection of objects in an Mroom. Each Mroom has a *main session* which manages the Mroom's root directory and representations of the users in the Mroom. In general, other sessions are associated with an Mroom, in which a subset of the users are using a subset of the objects in the Mroom. Figure 2 shows a session with a group of four *peer* processes, which run at users' workstations, and two server processes that constitute the session's *persistence domain*. The collection of shared objects includes a directory which maps the objects' names to their attributes, and an encryption key for privacy and secure group membership.

Messages describing events that result in changes to shared objects are transmitted from the process initiating the event to all the members of sessions maintaining replicas of those objects. The event messages are delivered atomically to processes in the persistence domain, but only on a best-effort basis to the peer processes. Thus we avoid the expense of atomic delivery to a potentially large group of peers with rapidly changing membership. But we guarantee that peers can obtain lost events and authoritative state from a member of the persistence domain, although it may also sometimes be obtained from a local peer for performance reasons. Event delivery is source-ordered (FIFO-ordered) by default, and also totally ordered if required. Total ordering is required for events describing updates to the session's directory; but it is not always necessary for other

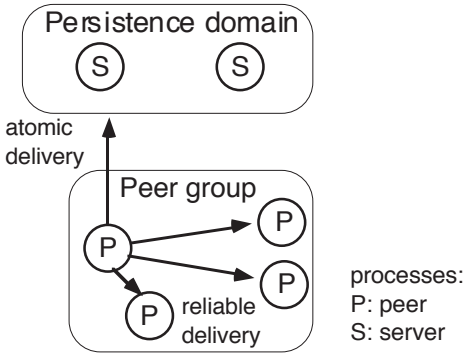

Figure 2. Updates in a session involving the peers and the servers in its persistence domain.

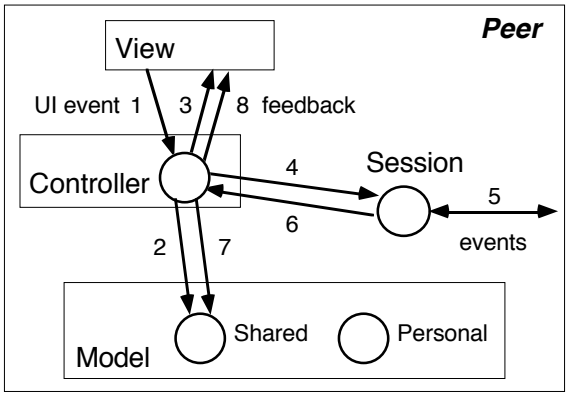

Figure 3. Model-view-controller with replication.

objects in the session. For example, if an object such as a whiteboard applies floor control, then source-ordering will suffice.

Event messages are delivered by protocols based on IP multicast. To implement total ordering a member of the persistence domain acts as a sequencer. We can choose dynamically which member this is, according to the network location of the majority of peers, and according to the sequencing workload on the servers due to other sessions [6]. The disadvantage of a sequencer is that it may become a bottleneck and it makes recovery from failure difficult. The protocols used to implement atomicity and total ordering are still under investigation.

3.2 Processing of events

When a user performs an action on a shared object, we need to provide her with rapid feedback in the user interface and to propagate the corresponding event to the replicas. Due to the latency of the multicast protocol, it is not feasible to delay the user feedback until the propagation is completed. Therefore the user is given immediate tentative feedback, and is eventually shown the correct shared state of the object when the multicast is completed. For example, if two users apply simultaneously to control the floor in a conference, then their positions in the queue could appear shaded at first and solid when the ordering is decided.

The Model-View-Controller architecture [7] separates the user interface into three types of component: a model (a collection of objects that are to be viewed and or changed); one or more views (objects that provide visual representations of the model); controllers (objects that deal with keyboard and mouse interactions).

We have adapted this architecture to allow for the replication of objects [8]. The model comprises both shared and personal objects as illustrated in Figure 3. The controller is responsible for user interface events (1). It has to be able to distinguish between shared and personal objects in the model and, in the case of shared objects, to propagate events to peers in the session. In the case of shared objects, the controller obtains the object state (2), provides tentative feedback through the view (3) and propagates events to replicas via the session (4, 5).When the multicast is complete (6) the controller applies the corresponding operations to the shared object (7) and provides feedback (8). Incoming events caused by the actions of other users are delivered to the controller, applied to the objects and displayed in the view.

Some events need not be handled by every process in a session. For example, when a user drags a new version of an object into an Mroom, an integrity check must take place. To save redundant processing it could be performed locally. This is satisfactory only if everyone else trusts the user. The alternative is for a trusted server in the persistence domain to make the check. This scheme has the disadvantage of loading servers with extra work and potentially delaying many sites as a result. However, the load could be balanced between the servers, for example, by choosing a server at random. We believe that both peer-centred and server-centred processing should be provided.

Some events can affect several sessions. For example, when a user drags an object from one session to another, both should be notified. Each session has an event propagation service, which receives subscriptions to events from other sessions. Whenever an event is received over the session, this service forwards it to any sessions that have registered interest in it. The service runs at the members of the session's persistence domain.

3.3 Naming and locating objects

We introduced above the requirement for Mrooms to have names which can survive changes in the set of organisations that have a stake in those Mrooms. For example, it would be unfortunate if an Mroom for users involved in the development of Web technology had to change its name when MIT was replaced by another university as the prime academic body behind the Web.

We have considered two solutions to this problem. The first is to establish a space of organisationally transparent names, akin to the USENET name space. Unfortunately we do not have an acceptable model of binding names into such a name space. Users vote for USENET names (except in the *alt*.* sub-tree). The difficulty is political rather than technological: it concerns who has control over name binding.

We now prefer a second option, in which an Mroom may have several names. A name would commence with the DNS name of a server where the remainder of the name can be resolved (c.f. URNs [9]).

A user or group with an interest in a given Mroom has a link to that Mroom in their own Mroom. The attributes of an Mroom link – as with any type of object in Mushroom – include a Mushroom Resource Locator, MRL. Unlike URLs, MRLs are system identifiers not meant for human eyes. We cannot use

physical locations in MRLs except as hints; but it would be undesirable to have to locate an object from a pure identifier. We have therefore divided the problem of locating an object (or a missing event) into locating the object's session, and then presenting the object's identifier to the session. The MRL for an object includes:

- a globally unique identifier for the object
- the globally unique identifier of the session that contains the object (this is a hint, since the object may migrate).

A session identifier is resolved into a hint containing:

- the set of servers in the session's persistence domain
- the session's IP multicast address
- the set of local sites that are members of the session.

To join a session or to access an object in the session, a peer contacts a member of the session using the session identifier in the relevant MRL. The session identifier is resolved into a session hint which is then used to contact a member of the session.

To access an object, a peer multicasts its object-request locally, using the multicast address in the hint. To avoid more than one local session member replying, the peer specifies in its request a list of local sites from which it would prefer to receive the object, and the protocol is designed so that only one recipient replies. The requesting peer's list is constructed according to such considerations as trust and closeness. Random ordering of equivalent sites helps to balance load. This scheme is similar to the approach taken in the SRM protocol to obtain missing messages [10]. But our approach of using a preference list specified by a requesting peer has an advantage with respect to load balancing and trust.

4. Discussion

Several systems employ similar room-based metaphors (e.g. Worlds [11] and BSCW [12]). Mushroom differs in its concern with the integrity of shared information objects, and in its focus on a scalable and flexible system architecture through using replicated state, group communication and event-based updates. We are currently engaged in a feasibility study to investigate the utility of the Mushroom user interface abstractions of Mrooms and boundary crossing for sharing objects. We are developing a prototype which we shall also use to investigate the extent to which existing protocols for managing shared state are suitable for interactive applications over the Internet. We need to minimise the latencies induced by communication delays, but our approach recognises that they will remain significant and that the user interface must be designed to accommodate them.

Many system issues remain to be resolved in the design of Mushroom, including the details of the group communication and naming schemes we have outlined. The number of servers needed in Mroom persistence domains and the scalability of our proposed architecture are as yet poorly understood. These are partly political problems, because an added server improves availability for one group of users, at the expense of increased costs for other groups. While we believe our object location scheme is promising it raises many issues, including how session hints are updated. Security for group working is under investigation [13]. Behaviour under partitions is addressed by [14], but we believe that more investigation is needed into user requirements for dealing with partitions.

Our investigations into application scenarios such as the Web site design task convinces us that proper support for interactive object-sharing will greatly enhance organisations' productivity and benefit users. Other promising application domains include education, collaborative software development and software support.

5. References

1. Kindberg, T., Mushroom: a framework for collaboration and interaction across the Internet. Proc. CSCW and the Web, 5th ERCIM/W4G workshop, Busbach, U., Kerr, D., and Sikkel, K. (eds), GMD, pp. 43-53, 1996.
2. Liskov, B., Day, M. and Shrira, L., Distributed object management in Thor. In Distributed Object Management, Oszu, M., Ed., Morgan Kaufman, 1994.
3. Makpangou, M., Gourhant, Y., Le Narzul, J.P., and Shapiro, M., Fragmented objects for distributed abstractions. Readings in Distributed Computing Systems, IEEE Computer Society Press., Jul 1994.
4. Berners-Lee, T., Masinter, L., McCahill, M., Uniform Resource Locators (URL). RFC 1738, Dec 1994.
5. Dean, D., Felten, E. and Wallach, D., Java security: from HotJava to Netscape and beyond. Proc. IEEE Symp. on Security and Privacy, May 1996.
6. Kindberg, T. A sequencing service for group communication (abstract). Proc. Principles of Distributed Computing 95, ACM, p. 260, 1995.
7. Krasner, G., and Pope, S. A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80. JOOP, vol 1, no 3, pp 26-49, August/September, 1988.
8. Kindberg, T., A stake in Cyberspace. Proc. 7th. ACM SIGOPS European Workshop, Connemara, Eire, Sept. 1996.
9. The URN Interoperability Project (TURNIP). http://www.dstc.edu.au/RDU/TURNIP/.
10. Floyd, S., Jacobson, V., Liu, C., McCanne, S., and Zhang, L., A reliable framework for light-weight sessions and application level framing. Lawrence Berkeley Laboratory tech. report, ftp://ftp.ee.lbl.gov/papers/wb.tech.ps.Z., 1995.
11. Tolone, W., Kaplan, S. and Fitzpatrick, G., Specifying dynamic support for collaborative work within WORLDS. Proc. ACM Conference on Organisational Computer Systems, pp. 55-65, August 1995.
12. Bentley, R., Horstmann, T., Sikkel, K., and Trevor, J., Supporting collaborative information sharing with the World-Wide Web: The BSCW Shared Workspace system. 4th International WWW Conference, Boston, Dec.1995.
13. Rowley, A. and Dollimore, J., Replicated secure shared objects for groupware applications. Tech. rep. 716, Dept. of Comp. Sci., Queen Mary & Westfield College, U. of London.
14. Cosquer, F., and Verissimo, P., Large-scale distributed support for cooperative applications. Proc. European Research Seminar on Advances in Distributed Systems, pp. 105-110, 1995.