Designing and Implementing Asynchronous Collaborative Applications with Bayou

W. Keith Edwards, Elizabeth D. Mynatt, Karin Petersen,
Mike J. Spreitzer, Douglas B. Terry, Marvin M. Theimer
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
{kedwards, mynatt, petersen, spreitzer, terry, theimer}@parc.xerox.com

ABSTRACT
Asynchronous collaboration is characterized by the degree of independence collaborators have from one another. In particular, collaborators working asynchronously typically have little need for frequent and fine-grained coordination with one another, and typically do not need to be notified immediately of changes made by others to any shared artifacts they are working with. We present an infrastructure, called Bayou, designed to support the construction of asynchronous collaborative applications. Bayou provides a replicated, weakly-consistent, data storage engine to application writers. The system supports a number of mechanisms for leveraging application semantics; using these mechanisms, applications can implement complex conflict detection and resolution policies, and choose the level of consistency and stability they will see in their databases. We present a number of applications we have built or are building using the Bayou system, and examine how these take advantage of the Bayou architecture.

KEYWORDS: computer-supported cooperative work, asynchronous interaction, distributed systems, Bayou.

INTRODUCTION
Collaboration involves sharing: the sharing of data, artifacts, context, and ultimately ideas. The CSCW community has often categorized collaborative systems based on the temporal aspect of sharing: applications in which users share some "thing" at the same time are called synchronous. Applications in which the users share that thing at different times are called asynchronous.

Synchronous applications, typified by such systems as ShrEdit [15][18] and SASSE [1], are highly-interactive, "real-time" systems in which a group of possibly distributed users interact together to achieve some result. Much of the recent research into collaboration, with the exception of electronic mail [7] and occasionally group editing studies [17] has focused on new tools and techniques to support synchronous collaboration.

Asynchronous systems, however, present a number of unique challenges to designers and builders of collaborative systems, from both the human and the technological perspectives. Asynchronous systems are appealing because they allow their users to manipulate time and space to their own advantage—users can work when and where they please, without being constrained by the schedules or locations of others. This style of work, and the settings where asynchronous systems are deployed, have implications for the design of infrastructure and applications. Asynchronous systems must accommodate groups of largely autonomous users, perhaps only loosely connected to each other at any given time.

This paper explores design issues for collaborative systems in general, and asynchronous systems in particular. We examine the reasons that users opt for asynchronous interaction, and the implications of those choices for designers of collaborative infrastructure and applications. We also present a system, called Bayou, designed to support data sharing by groups of individuals working together.

Bayou is an infrastructure for supporting distributed and collaborative applications in which all user interaction involves reading and writing a shared, replicated database. Unlike many infrastructures for collaboration, Bayou is capable of operating over a range of connectivity parameters, from high-bandwidth and constant connectivity, to low-bandwidth and only occasional or unreliable connectivity, as in the case of mobile users. Bayou is a true distributed system—meaning that there is no single centralized location at which data is stored—with weak consistency among replicated data.

Bayou provides mechanisms for application builders to describe the semantic constraints of their applications to the system. These mechanisms allow applications to supply their own data-integrity constraints, conflict detection and resolution procedures, and data propagation policies.

In the following section, we discuss some of the characteristics of asynchronous work, and the properties of asynchronous work that make it desirable for many forms of collaboration. Next, we examine the impact of these characteristics on infrastructure and application design—of necessity, any system for supporting asynchronous work must be informed by the properties of such work.

Then, we describe the Bayou infrastructure. We detail the goals of the system, how it works, and the implications of Bayou for application builders. To demonstrate how Bayou supports the design of asynchronous systems, we describe a set of applications built on top of Bayou. These applications span a range of complexity and interactivity, and each presents a set of lessons for infrastructure builders and application writers.

CHARACTERIZING ASYNCHRONOUS COLLABORATION

Asynchronous collaboration is typically characterized as "different place/different time" collaboration. This characterization is often too simplistic, however. For many asynchronous systems, the defining characteristic is not the fact that the collaboration *doesn't* happen at the same time, rather that it *needn't necessarily* happen at the same time. This distinction is not simply a pedantic one—it has implications for designers of applications and infrastructure.

In an asynchronous setting, the reason that collaboration can happen at different times is because the users do not need to coordinate with one another interactively, and do not need to be notified in "real time" of each other's changes to the artifacts they are sharing. Certain collaborations may lend themselves to this style of interaction because of the nature of the task itself, the work practices of the participants, or the state of the technology at hand.

Tasks that are suitable for this style of work often require little interactive coordination and sharing of work. Collaborators typically can work independently for periods of time, and there is little need for instantaneous propagation of results.

Work practices that favor asynchrony are characterized by people exploiting time and space to work at their convenience and with limited disruption. Such practices may come about because of setting (time zones that prevent collaborators from working at the same time, for instance), or personal desire (minimization of interruption by letting telephone calls "roll over" to voice mail for example).

Technological constraints may also favor asynchrony. Common examples of these include limited network bandwidth that prevents fine-grained or timely sharing of information, and disconnected use (such as using a laptop on an airplane) that separates collaborators.

Independence is perhaps the key trait of asynchronous work. In asynchronous interaction, collaborators, while still operating on some shared set of data, context, information, or artifacts, do so largely independently of one another.

In such work, the need for coordination—communication *about* the collaboration—is lessened, or at least less frequent than it is in synchronous work. For example, collaborative paper writing—at least in the non-computer mediated case—typically involves fairly infrequent coordination. Authors work largely independently, "syncing up" only when necessary to integrate results, or to reaffirm goals or plans [17].

Further, asynchronous tasks that center around some shared artifact do not typically require that all participants immediately know about changes to that artifact. In fact, in some cases such knowledge may be detrimental because it disrupts individual efforts and may incur coordination overhead, when such operations may be more profitably deferred to later.

SUPPORTING ASYNCHRONOUS COLLABORATION

The properties of tasks, work practice, and technology that lend themselves to asynchronous interaction point to infrastructure traits that can support applications for asynchronous tasks.

Independence points to the need to "insulate" collaborators from the actions of others—collaborators should be able to operate with limited interference from or coordination with others. In particular, they should be able to *continue* working, regardless of the actions taken by coworkers. Replication of data is often a useful means for achieving independence of work. Replication can separate the actions of users from their colleagues, providing performance, fault-tolerance, and the ability to locally integrate changes before releasing them to the world at large.

One of the strongest forms of independence is the ability to work completely disconnected from the network and, by implication, other users. The desire to support disconnected use means that users must be able to view, update, and add to their own private replicas of data even when they are not on the network. This constraint requires us to support replicas that are only weakly consistent with one another. If we required strong consistency then all parties would have to be connected at all times, and users would lose a degree of independence from one another.

While eventual consistency of replicas is desirable, users also need to control when information is shared with other users. Applications such as word processing or software development might require explicit control over information propagation. For example, in the case of collaborative software development, users often wish to ensure that updates are withheld until a complete, coherent, and stable picture of the code is available.

Finally, since asynchronous interaction often relies on the fact that collaboration can be achieved even in the face of minimal coordination among users, support for automatic resolution of conflicts can help reduce the need for coordination. If we can mechanically deal with conflicts, we can relieve uses of the burden of "by hand" coordination about their shared artifacts. To be usable by a range of applications, the conflict facilities must be able to implement application-specific policies about how to deal with conflicts. Succinctly, applications must be able to provide their own semantics about how to resolve conflicts automatically.

In the following section we describe a system called Bayou that satisfies these requirements for supporting asynchronous collaboration.

BAYOU OVERVIEW

Bayou is a replicated, weakly consistent storage system designed to support collaborative applications in distributed computing environments with varying network connectivity [22]. A typical example of such an environment is a system with mobile hosts that may disconnect over periods of time, connect only through low-bandwidth radio networks, or connect occasionally with expensive cellular modems. Its model for replication and weak consistency—allowing disconnection of servers from the network—is designed to support extreme scalability, up to "world wide" applications. Bayou relies only on pair-wise communications between computers, which allows the system to cope with arbitrary network connectivity.

Bayou applications can read from and write to any available replica without the need for explicit coordination with other replicas. Every replica eventually receives updates from all other replicas through a chain of pair-wise exchanges of data. To handle the update conflicts that naturally arise in such a weakly consistent system, Bayou allows applications to specify how to detect and resolve these conflicts. In addition, Bayou allows applications to select or specify a number of other policies that control how and where read and write operations get executed.

These characteristics make Bayou well suited for building wide-area asynchronous collaborative systems.

The Bayou System Model

In Bayou, replication is managed by Bayou servers. Each server holds a complete replica of the data. The data model provided by the current implementation of Bayou is a relational database, although other data models could be used as well. We chose a relational model because of its power and flexibility. In particular, it naturally supports fine-grained, structured access to the data, which is useful for the application-specific conflict detection and resolution mechanisms described below. Higher-level application-defined data constructs can be created in terms of the data model provided by the relational database.

As mentioned above, Bayou replicas are weakly consistent. That is, at any point in time different servers may have seen different sets of updates and therefore hold different data in their databases. Weak consistency distinguishes Bayou from many of the replicated systems designed in the CSCW community [3][10]. Some collaborative and distributed systems infrastructures use fairly strong forms of consistency, usually based on pessimistic locking. That is, before data can be modified it must be locked to ensure that its access is serialized. Such strongly-consistent schemes ensure that applications always see a consistent picture of the data. However, they do not support weakly-connected applications, and do not scale to the global applications envisioned by Bayou.

Much like Lotus Notes [13], Bayou applications are free to read and update replicas at will, without locking. Bayou guarantees that the distributed storage system will move toward eventual consistency by imposing a global order on write operations and by providing propagation guarantees. Each write carries enough information so that a Bayou server can apply the writes it has received in the correct order without coordinating with any other server.

Bayou's Mechanisms for Application Semantics

One feature that distinguishes Bayou from previous replicated storage systems including Ficus [12], Coda [14][21], and Lotus Notes [13] is that applications can impose their own semantics on the operations executed at a replica. To this end, Bayou reads and writes are not the simple operations supported by most databases. Instead they include additional application-supplied information, which ensures that applications will receive the required level of service from the system.

Bayou's mechanisms for supporting application semantics fall into six categories:

- Application-defined conflict detection.
- Application-defined conflict resolution.
- Selection of session guarantees.
- Selection of committed or tentative data.
- Replica selection.
- Selectable anti-entropy (data propagation) policies.

Conflict Detection and Resolution. The first two semantic categories are provided through the Bayou write operation, and are designed to detect and resolve the conflicts that arise in a weakly-consistent system. In Bayou, a *write* consists of three components:

- Dependency Check
- Update Set
- Merge Procedure

The *dependency check* specifies a set of conditions that must hold so that the *update set* can be applied to the replica's database. A dependency check consists of a query to be performed at the database and the expected result of that query. If the actual result matches the expected result, then the update set in the write is applied to the database. The update set consists of insertions, deletions, or modifications of tuples in a relation.

If the dependency check fails, an application-specific conflict has been detected and the merge procedure is executed. The *merge procedure*, or "mergeproc" in short, is a fragment of code in a high-level interpreted mergeproc language intended to generate an alternate update set to be applied to the database. Mergeprocs support application-defined conflict resolution, meaning that conflicts are essentially handled through application code, even though that code is executed by the Bayou infrastructure itself. We shall see some examples of mergeprocs in our discussion of applications.

Bayou's use of mergeprocs differs from systems like Coda [14][21] and Ficus [12], which also support application-supplied conflict resolution, in that Bayou allows different resolution procedures to be associated with each individual write. Thus, Bayou provides applications with more fine-grained control over conflict handling. Furthermore, because the conflict resolution procedure propagates with the write it is available at each server when needed.

The mechanisms for automated conflict detection and resolution are important for supporting asynchronous collaboration, because they eliminate situations where users would otherwise be required to interact closely when faced with data conflicts. Hence, Bayou allows users to act more independently.

Session Guarantees. The session guarantees mechanism is used by an application to establish a required level of consistency for its own operations. That is, while a set of Bayou servers maintain data that is only weakly-consistent, a running instance of an application can request that its view of the world maintain a particular level of consistency. Different applications may have different requirements for their desired level of consistency, and Bayou supports a range of applications needs through this mechanism.

A *session* is an abstraction for a sequence of reads and writes performed during the execution of the application, and session guarantees are implemented by constraining the replicas that may be selected by the application during that session.

Four session guarantees are supported by Bayou:

- *Read Your Writes* ensures that the effects of any writes made within a session are visible to later reads within that session. In other words, reads are restricted to replicas of the database that include all previous writes in the session.
- *Monotonic Reads* permits users to observe a database that stays up-to-date over time. It ensures that reads are only made to database replicas containing all writes whose effects were seen by previous reads within the session.
- *Writes Follow Reads* ensures that traditional write/read dependencies are preserved in the ordering of writes at all servers. That is, at every replica of the database, writes made during the session are ordered after any writes whose effects were seen by previous reads in the session.
- *Monotonic Writes* says that writes must follow previous writes within the session. In other words, a write is only incorporated into a replica's database copy if the copy includes all previous writes from that session, and the write is ordered after these previous writes.

Session guarantees are described in more detail in [23], and are not intended to ensure atomicity or serializability. Instead, users of collaborative applications use session guarantees to maintain a self-consistent view of the database, even though they may read from and write to various, potentially inconsistent, replicas over time.

Stable vs. Tentative Data. Bayou provides a mechanism that establishes when a write is *stable* at a given server. That is, when no new writes will ever be received by the server that will have to be ordered before that write. When a write becomes stable at a server, its conflict detection and resolution mechanisms will not be executed again, which means that its final effect on the database is known. On the other hand, a write that is not yet stable at a server is deemed *tentative*. Tentative writes may need to be re-executed if other writes with earlier write-stamps are received by the server, and thus have a possibly changing effect on the database.

The distinction between tentative and stable data is important from the application's perspective. An application can be designed with a notion of "confirmation" or "commitment" that corresponds to Bayou's notion of stability. For example, color codes can be used in a graphical user interface to indicate whether a displayed item is tentative, that is, may change later because of conflict, or is stable and will not change due to conflict.

Bayou also allows clients to choose whether they will read from the database when tentative data has been applied, or only from the view of the database that corresponds to applying only stable writes. This ability allows clients to trade data availability for assurance of data stability—applications that can tolerate data that has not fully stabilized can read it immediately, without waiting for it to become stable.

Although stability does not equate with consistency, when a collaborative application reads only the results of stable writes, its users will perceive a different "sense" of consistency than if the application also reads tentative data.

Replica Selection. Another important feature that Bayou provides to an application is the ability to select which replica it will use for its operations. The ability to select from several replicas over the life-span of an application is particularly important to collaboration:

- A particular replica can be selected to optimize certain communication requirements. In particular, autonomous users with a disconnected laptop can run a server for a local replica on that laptop. Applications can choose this server, thus ensuring access to the database.
- Applications operating on behalf of different users on different machines can be connected to the *same* replica, which enables all the application instances connected to that replica to see updates as soon as they occur. In essence, the applications can work together in a tightly-integrated, strongly-consistent, synchronized fashion. The ability of applications to connect to a single replica, and later split apart and communicate with different replicas, can be used to support transitions between synchronous and asynchronous styles of collaboration.

Anti-entropy Policies. Anti-entropy is the pair-wise process by which the servers of two replicas bring each other's databases up to date. During the anti-entropy process two servers exchange the sets of writes known to one server but not the other [4]. For a more detailed description of the reconciliation protocol and its performance, refer to [20].

Although not fully implemented yet, the Bayou model supports client-supplied anti-entropy policies. Thus, clients can influence when to propagate their changes to the database to other servers. (Currently, anti-entropy is performed automatically at a set interval, or when manually requested by an application.) The ability to regulate when updates are propagated is important for applications like collaborative software development where users must ensure that a coherent picture of the code base is available at specific times.

IMPLEMENTING COLLABORATIVE APPLICATIONS WITH BAYOU

This section describes a range of collaborative applications we have built, or are building, on top of the Bayou infrastructure. Three of the applications below—a shared bibliographic database, a group calendar system, and a mobile electronic mail system—have been completed. The Bayou Project Coordinator system is still in the design stage. The final "application" is actually a higher-level collaborative toolkit. This toolkit exists currently but does not use Bayou. We are investigating porting the data storage portion of the toolkit to use Bayou.

All of these applications share the following characteristics: they are highly asynchronous, requiring few, or in some cases no, synchronous updates from other users. They can tolerate weakly-consistent data, and they can benefit from mechanized conflict detection and resolution.

We describe each of these applications, examine how Bayou benefits the applications, and how the applications have informed our designs and goals for Bayou.

Collaborative Bibliographic Database

BibDB is a multi-user shared bibliographic database that allows users to add and modify entries, and automatically generates citation keys that are used to refer to those entries. The system is conceptually similar to, but simpler than, bibliographic database systems like RefDBMS [9]. BibDB is perhaps an asynchronous application in its purest form: users of the system never "see" other users of the application. BibDB provides no awareness of others, even when several people are using the application at the same time.

Consider a situation in which Alice and Bob maintain a bibliographic database for their research project using BibDB. Their style of interaction is extremely asynchronous: even if Alice and Bob are updating the database at the same time, they have no knowledge that the other is using the tool. Further, the propagation and visibility of updates need not occur immediately: in most cases, Bob does not need to know immediately if Alice adds a new entry, although they will eventually need to know about duplicate entries. In other words, by its requirements, the system is tolerant of weak consistency and does not require updates to be globally visible immediately.

BibDB uses a simple algorithm to generate human-readable citation keys: the key is a few letters of the author's last name with a postfix consisting of the last two digits of the publication year appended, and possibly an extra character in the case of multiple papers by the same author from the same year. If two users add entries that would result in the same citation key, the conflict detection and resolution procedures will change the updates to ensure that keys are always unique.

This scenario is an example, albeit simple, of how the Bayou system can incorporate application-specific integrity constraints. "Application intelligence," in the form of a Bayou merge procedure, always ensures semantically-meaningful keys. Merge procedures are also used to detect and merge duplicate entries in the database. Note that conflicts can be resolved without the need for "manual" user intervention or coordination among users because of the mechanisms provided by Bayou.

Because of weak consistency and the fact that BibDB reads tentative writes, users must be aware that tentative citation keys may change until they become stable. So if a user refers to a newly-added citation key in a paper, he or she must check back once the update is stable to ensure that the key has not changed. Users who are well-connected may opt to only read stable data. But users can choose to view tentative data to maximize data availability when connection is poor.

BibDB is an example of a highly asynchronous application in which only loose artifact sharing is required. And, because of automated conflict detection and resolution, no user-level coordination is required. In other words, the application is an excellent match for Bayou.

Group Calendar

Like BibDB, Group Calendar helps users manage a shared resource, in this case, a shared calendar. One common usage example is conference room scheduling. This task has the following characteristics:

- Users may expect conflicts since they are negotiating the use of a shared resource.
- Awareness of other users is not critical since scheduling policies can be provided by the application.
- The application data, that is, dates and times, are structured, allowing the application to detect conflicts.
- The task supports specifying alternative appointment times for use when conflicts with other users occur.

As a typical scenario, imagine that Jane uses Group Calendar to schedule a meeting in the conference room from 10:30 am to 11:30 am on Monday. She also specifies Wednesday at the same time as an alternate. While working on the train, Kevin schedules a project meeting in the conference room from 10:00 am to 11:00 am on Monday. He also specifies Monday from 12:00 pm to 1:00 pm as an alternate time. When Kevin connects his laptop to the network, his modifications propagate through the system. As the writes are transmitted between the database replicas for the

```
Bayou_Write(
   update = {insert, Meetings, 12/18/95, 10:00am, 60min, “Project Meeting: Kevin”},
   dependency_check = {
      query = “SELECT key FROM Meetings WHERE day = 12/18/95
         AND start < 11:00am AND end > 10:00am”,
      expected_result = EMPTY},
   mergeproc = {
      alternates = {12/18/95, 12:00pm};
      newupdate = {};
      FOREACH a IN alternates {
         # check if there would be a conflict
         IF (NOT EMPTY (
            SELECT key FROM Meetings WHERE day = a.date
             AND start < a.time + 60min AND end > a.time))
               CONTINUE;
         # no conflict, can schedule meeting at that time
         newupdate = {insert, Meetings, a.date, a.time, 60min, “Project Meeting: Kevin”};
         BREAK;
      }
      IF (newupdate = {})    # no alternate is acceptable
         newupdate = {insert, ErrorLog, 12/18/95, 10:00am, 60min, “Project Meeting: Kevin”};
      RETURN newupdate;}
)
```

FIGURE 1: A Bayou Write for Group Calendar

conference room calendar, a conflict is detected. Kevin later receives a notification that due to a conflict, the conference room has been reserved at the alternate time he specified.

The Bayou write resulting from Kevin's input is shown in Figure 1. The write specifies that, given a conflict, if no alternative reservation can be found, the update is written to the error log. In the Group Calendar interface, items in the error log are accessible, enabling users to determine when their reservation requests have been unsuccessful.

Like BibDB, users must decide whether they want to only see stable writes to the calendar. Tentative writes can be color-coded in the graphical interface as shown in Figure 2.

Group Calendar typifies applications that can provide policies to minimize multi-user coordination. Since the experience of multiple people wanting to reserve the same thing is common, users are familiar with the strategy of providing alternate requests. The advantages of not having to wait for the approval of other users, as well as being able to work disconnected from the network, outweigh the cost of unresolved conflicts.

Two planned modifications to Bayou will improve the usability of Group Calendar. First, strategies for server selection and anti-entropy will help ensure that tentative writes stabilize quickly. Second, notification facilities for failed requests will remove the need for users to confirm their reservations.

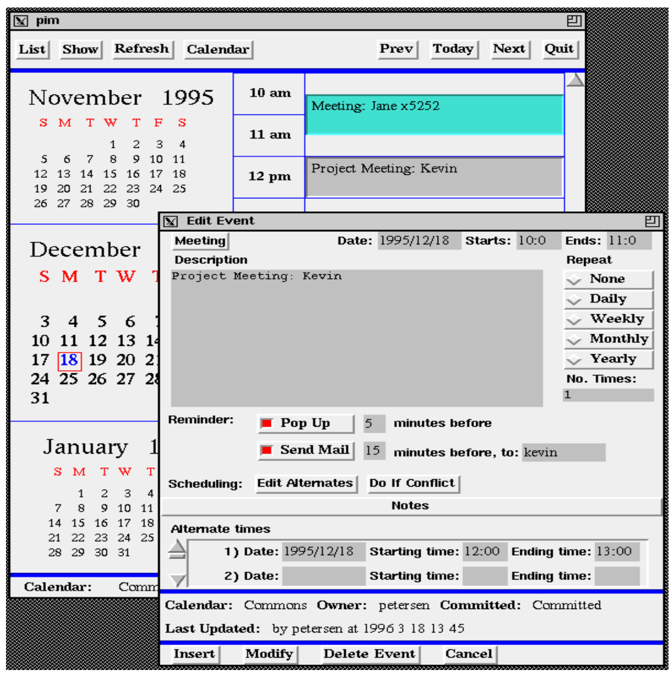

FIGURE 2: The Group Calendar Application

Mobile Electronic Mail

Electronic mail is often considered to be the “classical” asynchronous collaborative application. Even so, electronic mail has very different characteristics than the other applications examined here. Perhaps most importantly, there is very little shared state among participants, in the sense that when a message is “shared” with a collaborator, a copy of it is sent. There is typically no one single copy of a message that is simultaneously shared among collaborators.

But, even though messages are copied among collaborators using traditional (and existing) mail routing facilities, the state of a *particular* user's mail folders can profitably be stored and shared in Bayou. Thus, we have implemented a mail user agent called BXMH on top of the EXMH mailer. BXMH supports “mobile” access to electronic mail—a user can have access to his or her particular mail folders and messages, whether at a desktop machine in the office, a computer at home, or a laptop that is disconnected from the network. Even though replicas of the data are stored across multiple servers, changes made to any copy of the mail database will eventually be propagated to all other copies.

BXMH is implemented by replacing the file handling layer of EXMH with an interface to the Bayou relational database. Messages and mail folders are represented as sets of tuples stored in relations in the database.

A BXMH user will typically run a Bayou server—containing his or her mail—on each machine where mail will be read. Any machine with a Bayou server running on it can then be disconnected from the network. So, for example, a laptop machine running a Bayou server can be taken “on the road.” The user of this machine will still have access to all of his or her email. Further, changes can be made to the mail database while disconnected—filing messages, changing folder hierarchies, renaming folders, and so on. Later, when the machine is reconnected to other Bayou servers, the states of the mail database maintained at each server will be rectified. The Bayou anti-entropy protocols cause all servers to move toward a consistent state, and changes made while disconnected are propagated to the office and home machines. Note also, that the anti-entropy protocols can be run across low-bandwidth connections, including dial-in modems, infrared, or simply by exchange of floppy disks.

Inconsistencies can arise when a changes are made at multiple servers. One common example is when mail is automatically incorporated on a desktop machine. This new information must be merged with changes made by a user on a laptop. Conflicts can also arise when a user makes inconsistent changes on both a laptop and an office machine, perhaps filing a given message into two different folders at each machine.

Mergeprocs come into play to resolve inconsistencies that may arise between the two mail databases when updates are propagated between them. In BXMH, mergeprocs are used to “push” mail application semantics into the Bayou system.

BXMH defines a suite of mergeprocs that enforce particular policies about how the system should behave when certain inconsistencies arise. Common inconsistencies include situations such as when the user renames a folder on one machine but continues to file messages under the old name on another; the user deletes a folder on one machine while filing messages to it on another; the user disposes of one messages in different ways on different machines, and so on.

Rather than enumerating all possible choices to the user, BXMH provides a conflict policy UI that allows users to provide high-level guidelines about how to resolve conflicts. This interface allows users to favor one interpretation of an inconsistency over another. Conflict resolutions are recorded in a special mail folder in case the user wishes to know the details of what has transpired. Figure 3 shows a screenshot of the BXMH conflict configuration interface.

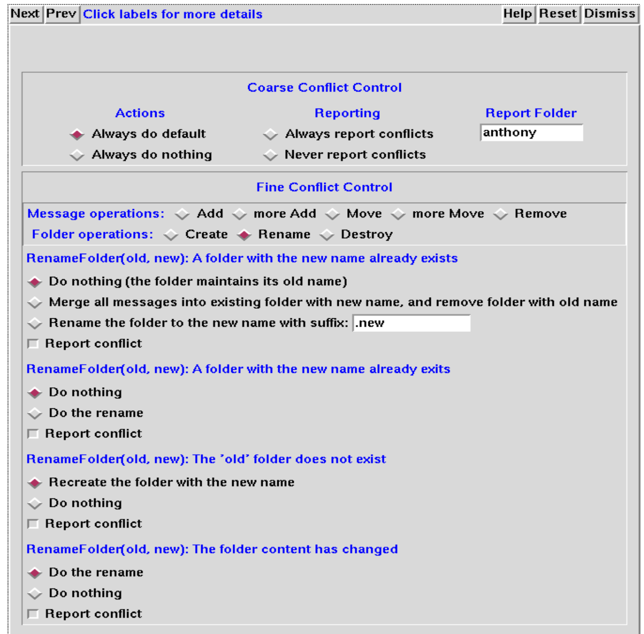

FIGURE 3: The BXMH Conflict Configuration Interface

Bayou Project Coordinator (BPC)

The Bayou project is a complex, multi-person effort requiring the management of many shared artifacts. Some of these artifacts are: the Bayou server binaries, various application databases and their replicas, laptops and modems, and per-application security certificates for users. We are currently designing the Bayou Project Coordinator to support the management of these resources. This application has two primary functions. First, it maintains dependencies between artifacts. For example, the system copies binaries for servers and client applications to new laptops. Second, it provides awareness of the activities of group members. Since a majority of tasks involve project artifacts, a representation of the activities of project members is culled from artifact use.

This application exercises Bayou’s conflict detection and resolution facilities in three ways. First, consistency among project artifacts is maintained without user intervention. For example, development platforms are kept up-to-date by distributing updated server kernels to Bayou machines Second, Bayou supports asynchronous interaction when user intervention is required. For example, when a new Bayou user is added, owners of specific applications asynchronously approve the creation of security certificates for the new user. Third, user interaction with Bayou servers triggers logging of user activity. By creating a dependency check that will always fail, the mergeproc will always be evaluated and can specify what activity information to record. The Bayou Project Coordinator can then use this information to summarize the activities of project members.

Consider the following scenario. Mike has been working for the past week on modifications to the Bayou server kernel. When he “checks in” changes that are ready for use by the other Bayou members, the modified kernel is propagated to their machines. During this time, Keith has introduced a new application and approved a set of users for this application. The BPC distributes copies of the application binary and security certificates to the set of approved users.

At the same time, Marvin has been traveling with a disconnected Bayou machine. When he reconnects his machine to the network, the results of Mike and Keith's work are transmitted to his machine. Observing performance differences in the behavior of the Bayou server, Marvin investigates the activity representations for group members in BPC. Noticing that Mike has spent the week working on the server, he is no longer surprised by the changes in server performance. Marvin also notices that Keith has introduced a new application. He decides to experiment with using the application while writing an email message to Keith with his initial impression.

BPC supports strong artifact sharing without requiring explicit coordination among users. Specifications about dependencies between project artifacts help maintain the integrity of the system including the propagation of new project artifacts. Given the complexity of the Bayou project, without the use of the BPC, project members need to constantly "baby-sit" the state of the project.

BPC also demonstrates using mergeprocs to trigger exploration of the shared data. In this case, the application summarizes changes to the data to provide awareness of the activities of project members. The same strategy could be used to trigger self-modifying data such as a word processor that automatically corrects spelling mistakes.

Timewarp

The Timewarp system is not an application, but rather another toolkit for building collaborative applications [6]. It is a "higher level" toolkit than Bayou, in the sense that it provides more functionality specifically designed to support collaboration. Timewarp provides mechanisms for awareness, coordination, multi-user access to data, and versioning.

The basic paradigm that Timewarp follows is that the history of a shared artifact is allowed to be divergent—that is, collaborators at multiple sites may "see" different versions of the artifact at any given time. Collaborators may work on these versions independently, perhaps reconciling them into one result periodically throughout a collaboration. The history of the artifact itself becomes a shared artifact that can be used to mediate the collaboration. So participants can "travel" through the parallel timelines of an artifact and make changes to the artifact at any point in its history.

The current Timewarp implementation uses a single centralized server to coordinate client applications that run at each collaborator's machine. We are investigating the use of Bayou as lower level infrastructure which would give us greater levels of independence, including the ability to disconnect from the network, that this style of collaboration favors.

Timewarp is implemented using Java and its Remote Method Invocation (RMI) system. We have created a Java-language interface to the client-side Bayou APIs. Porting Timewarp to use Bayou involves recoding the data structures used internally as tuples suitable for storage in a relational database.

The conflict management system used by Timewarp will have to be extended to take advantage of the disconnected operation permitted by Bayou. Currently in Timewarp, conflicts are brought to the attention of the user and (potentially) resolved as soon as they occur. Consequently, conflicts never "appear" in a timeline in which the user is not active. In a Bayou reimplementation of Timewarp, a given timeline will not be assured of being conflict-free until all of the data associated with it is committed. If a timeline depends on state that is still tentative, updates may still be received that will cause new Timewarp-level conflicts to occur. Mergeprocs will be used to integrate updates into timelines; this code will essentially translate Bayou database-level conflicts into Timewarp-level conflicts, and notify the Timewarp infrastructure that new conflicts exist and must be dealt with.

We believe that the facilities offered by these two toolkits—"high-level" versioning, awareness, and coordination by Timewarp, and true distribution, weak consistency, and disconnected use by Bayou—will complement each other when the two are combined.

SUMMARY AND PROJECT STATUS

Asynchronous collaborative systems present a number of challenging problems, from both the human design perspective and the technological perspective. This paper has investigated a number of the characteristics of asynchronous work, and the design challenges that must be addressed when building infrastructure for this space.

Asynchrony often arises when—whether through group work practices, technology, or simply the nature of the tasks at hand—collaborators need to work independently from one another. The human dynamics of asynchronous work have implications for designers of infrastructure and applications for asynchronous collaboration.

We have presented a system called Bayou that addresses many of these issues. Bayou has features that support both users and writers of asynchronous applications. Below we summarize some of the features of Bayou we feel are important for builders and users of asynchronous applications.

- **Efficient anywhere/anytime access to data.**

  Bayou supports weakly-consistent replication. Servers synchronize in a pair-wise fashion, supporting a range of work practices. For example, home and office machines can be synchronized through a laptop transported between locations; the home and office machines need never communicate directly with each other.

  This feature is used by BXMH to handle reading mail from any location, using exactly the same user interface even when disconnected. BibDB uses this feature for separation to support what is an intrinsically independent task.

- **Automatic management of conflicts.**

  Dependency checks and mergeprocs provide a way for applications to not only define for themselves what constitutes a conflict, but also to establish the procedures to take to resolve conflicts that occur. Thus, Bayou applications can often resolve most conflicts automatically, reducing the need for user intervention and coordination, and enhancing independence.

  All of the current Bayou applications use this feature, and most provide multiple resolution options to their users. BXMH allows users to set general conflict resolution policies. The group calendar lets users specify fallback times for calendars that will be used in the event of a conflict.

- **"Self consistency" and awareness of data status.**

  Session guarantees further support seamless transitions between servers. Clients can choose to see only the progression of activity, and not move back and forth between older and newer states. All of our example applications use this facility.

  Also, Bayou provides a means for applications to detect the status of data in a database—whether it is tentative or committed. This information can be presented to the user in a number of forms. In the group calendar, color is used to mark which entries will no longer change. In BXMH, the interface highlights which messages are in conflict and need attention from users.

  One weakness is that the current implementation does not notify applications (and hence users) when data changes—applications must poll the database to detect changes.

- **Flexible data model.**

  Bayou provides a robust and flexible data model for applications. The system supports any granularity of shared data. So writers can modify a field of a tuple (such as the time in a calendar entry), or entire sets of tuples (such as new versions of source code or mail folders) at once.

  While the relational data model may not be a "natural" fit for all applications, the model can be generalized for storages of other types of structured data fairly easily.

- **Fluid transition between synchronous and asynchronous modes of operations.**

  Multiple collaborators can connect to distinct servers for typical asynchronous operation, or connect to the same server for "tighter" synchronous operation.

  Users of the group calendar application typically connect to a centralized Bayou server to quickly share operations entered while at the office, therefore diminishing the opportunity for conflicts. But users can connect to local servers when disconnected, and still access and modify their calendars.

We have presented a number of asynchronous collaborative applications built or designed using the Bayou infrastructure that use these features. These applications span a range of interaction styles, both in terms of the amount of artifact sharing and the amount of coordination support they provide to their users.

The Bayou architecture outlined in this paper has been implemented and runs on Sun SPARCstations running SunOS 4 and 5, and on 486-based subnotebooks running Linux. The query language used in read operations and dependency checks is a subset of SQL. The mergeproc language is based on the Tool Command Language, TCL [19], augmented by SQL.

FUTURE RESEARCH

We expect that further exploration of the design of applications that transition between synchronous and asynchronous modes of operation will raise interesting questions, from both the interface and infrastructure perspectives. One key requirement for supporting synchronous applications is the ability for applications to request notifications when data at a server changes. In the current implementation, clients must poll the server to receive notification of changes, which makes the construction of synchronous applications difficult.

Issues we are planning to explore further in the context of the Bayou infrastructure include partial replication, policies for choosing servers for anti-entropy, server selection policies by applications, and fine grain access control.

Our most immediate design focus is on supporting partial replicas that contain subsets of a database. Partial replication is important for applications that run on laptops or PDAs, and raises a number of difficult problems ranging from characterizing a partial replica to resolving conflicts in a consistent manner across partial replicas.

We are currently examining the design issues surrounding porting the entire Bayou system to Java to enhance portability and ease-of-integration with applications. Such a system would most probably operate as a "replication layer" on top of an existing relational database management system.

We are also experimenting with wireless connectivity for servers and clients running on a laptop using the Metricom [16] wide-area radio network and point-to-point infra-red connections between laptops.

ACKNOWLEDGEMENTS

The design and development of Bayou have been a multi-year effort involving a number of people. Alan Demers, Carl Hauser and Brent Welch were invaluable participants in earlier stages of the Bayou design process. Mark Weiser, Craig Mudge, and John White, as managers of the Computer Science Lab, have been supportive throughout.

REFERENCES

[1] Baecker, R.M., Nastos, D., Posner, I.R., Mawby, K.L., "The User-centered Iterative Design of Collaborative Writing Software." *Proceedings of the Conference on Human Factors in Computing Systems*. Amsterdam, The Netherlands: ACM. 1993. pp. 399-405.

[2] Bly, S.A., Harrison, S.R., and Irwin, S., "Media Spaces: Bringing People Together in a Video, Audio, and Computing Environment." *Communications of the ACM*, Vol. 36, No. 1 (January 1993), pp. 28-47.

[3] Conklin, J., and Begeman, M.L. "gIBIS: A Hypertext Tool for Exploratory Policy Discussion." *Proceedings of the Conference on Computer-Supported Cooperative Work* (CSCW), Portland, OR: ACM, 1988, pp. 140-152.

[4] Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J. Shenker, S., Sturgis, H., Swinehart, D., and Terry, D. "Epidemic Algorithms for Replicated Database Maintenance," *Proceedings of the Sixth Symposium on Principles of Distributed Computing*, Vancouver, BC, Canada, August 1987, pp. 1-12.

[5] Dourish, P., and Bly, S., "Supporting Awareness in a Distributed Workgroup." *Proceedings of the ACM Conference on Human Factors in Computing Systems* (CHI'92), Monterey, CA: ACM, pp. 541-547.

[6] Edwards, W.K., and Mynatt, E.D., "Timewarp: Techniques for Autonomous Collaboration." *Proceedings of the ACM Conference on Human Factors in Computing Systems* (CHI'97), Atlanta, GA: ACM, pp. 218-225.

[7] Eveland, J.D., and Bikson, T.K., "Work Group Structures and Computer Support: A Field Experiment." *Proceedings, ACM Conference on Computer-Supported Cooperative Work*, Portland, OR: ACM, 1988.

[8] Galegher, J., and Kraut, R.E., "Computer-Mediated Communication for Intellectual Teamwork: A Field Experiment in Group Writing." In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, Los Angeles, CA: ACM, 1990, pp. 65-78.

[9] Golding, R., Long, D., and Wilkes, J. "The RefDBMS Distributed Bibliographic Database System." *Proceedings of Winter USENIX Conference*, San Francisco, CA, January 1994, pp. 47-62.

[10] Greenberg, S., and Marwood, D., "Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface." Proceedings of the ACM Conference on Computer Supported Cooperative Work, Chapel Hill, NC: ACM, Oct. 22-26, 1994. pp. 207-217.

[11] Grief, I., and Sarin, S. "Data Sharing in Group Work," *Computer-Supported Cooperative Work: A Book of Readings*, Irene Grief, ed. San Mateo, CA: Morgan Kaufmann, 1988, pp. 477-508.

[12] Heidemann, J.S., Page, T.W, Guy, R.G., and Popek, G.J. "Primarily Disconnected Operation: Experiences with Ficus," *Proceedings of Second Workshop on the Management of Replicated Data*, Monterey, CA, Nov. 1992.

[13] Kalwell, L. Jr., Beckhardt, S., Halvorsen, T., Ozzie, R., and Greif, I., "Replicated Document Management in a Group Communication System." *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, Portland, Oregon, September 1988.

[14] Kistler, J.J., and Satyanarayanan. "Disconnected Operation in the Coda File System." *ACM Transactions on Computer Systems* 10(1):3-25, February 1992.

[15] McGuffin, Lola, and Olson, Gary, "ShrEdit: A Shared Electronic Workspace," CSMIL Technical Report, Cognitive Science and Machine Intelligence Laboratory, University of Michigan, 1992.

[16] Metricom Inc. http://www.metricom.com.

[17] Neuwirth, C. M., Kaufer, D. S., Chandhok, R., and Morris, J. "Issues in the Design of Computer Support for Co-authoring and Commenting." *Proceedings of the Conference on Computer-Supported Cooperative Work,* Los Angeles, CA: ACM, 1990, pp. 183-195.

[18] Olson, J., Olson, G. Storrøsten, M., and Carter, M., "How a Group Editor Changes the Character of A Design Meeting as Well as its Outcome." *Proceedings of the Conference on Computer-Supported Cooperative Work*, Toronto, Ontario: ACM, 1992, pp. 91-98.

[19] Ousterhout, J., "Tcl: An Embeddable Command Language." *Proceedings of the USENIX Conference,* Winter 1990.

[20] Petersen, K., Spreitzer, M.J., Terry, D.B., Theimer, M.M., Demers, A.J. "Flexible Update Propagation for Weakly Consistent Replication." *Proceedings of the Sixteenth ACM Symposium on Operating System Principles* (SOSP), Saint-Malo, Franco, October 1997.

[21] Satyanarayanan, M., Kistler, J.J., Kumar, P., Okasaki, M.E., Siegel, E.H., and Steere, D.C., "Coda: A Highly-Available File System for a Distributed Workstation Environment." *IEEE Transactions on Computers* 39(4): 447-459, April 1990.

[22] Terry, D.B., Theimer, M.,M., Petersen, K., Demers, A.J., Spreitzer, M.J., Hauser, C.H. "Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System." *Proceedings Fifteenth ACM Symposium on Operating Systems Principles* (SOSP), Cooper Mountain, Colorado, December 1995, pp. 172-183.

[23] Terry, D.B., Demers, A.J., Petersen, K., Spreitzer, M.J., Theimer, M.M., and Welch, B. "Session Guarantees for Weakly Consistent Replicated Data." *Proceedings of the International Conference on Parallel and Distributed Information Systems (PDIS)*, Austin, Texas, September 1994, pp. 140-149.