Managing Distributed Collaboration in a Peer-to-Peer Network

Michael Higgins, Stuart Roth, Jeff Senn, Peter Lucas, Dominic Widdows
{higgins,roth,senn,lucas,widdows}@maya.com

MAYA Design Inc.**

To appear in 14th International Conference on Cooperative Information Systems (CoopIS 2006), Montpellier, France, Nov. 2006.

Abstract. Shared mutable information objects called u-forms provide an attractive foundation on which to build collaborative systems. As we scale up such systems from small fully-connected workgroups to large, highly distributed, and partially disconnected groups, we have found that peer-to-peer technology and optimistic replication strategies provide a cost-effective mechanism for maintaining good performance. Unfortunately, such systems present well-known coordination and consistency problems. This paper discusses strategies for addressing those difficulties at different levels of the system design, focusing on providing solutions in the information architecture rather than at the infrastructure layer. Addressing problems at this higher layer allows greater freedom in design, and simplifies moving from one infrastructural base to another as technology evolves. Our primary strategy is to enable robust decentralized and asynchronous collaboration while designing architectures that do not rely on two users writing to the same u-form at the same time in different venues. Techniques are provided for simple messaging, collaborative maintenance of collections, indexing supporting rich query, and stand-off annotation and elaboration of third-party datasets. We outline the application of these techniques in a working collaborative system.

1 Introduction

This paper describes a variety of structures that enable robust peer-to-peer collaboration on many semantic levels. The strategy we pursue is to create information architectures that allow widespread replication but minimize the danger of conflicted data. This pattern pervades many of our techniques, from a simple asynchronous messaging protocol to the ability of many publishers to contribute content about the same phenomena.

For some time, MAYA Design and our collaborators have been building powerful collaborative applications by allowing users to share and modify mutable information objects called *u-forms* [1]. This sort of "shared memory" approach

** This work was supported by Defense Advanced Research Projects Agency grant SB031-008 for the Cluster-Based Repositories and Analysis project.

to managing collaboration is both elegant and powerful. As the breadth and reach of our systems has expanded, we have encountered a variety of challenges, both obvious and subtle. In brief:

- Systems with many users generate large amounts of storage and retrieval cost.
- Systems with many users are rarely completely connected. Users are mobile: they come and go, and need information even when they have little or no network connectivity.
- Systems with many users generate large amounts of disagreement. These disagreements range from mismatches between versions of specific information objects to high-level differences of opinion on certain topics. Many such disagreements cannot and should not be resolved automatically.

We have found optimistic peer-to-peer replication to be a useful tool for attacking the first and second problems. Such a replication model permits a very scalable infrastructure, does not require a major datacenter investment, and tolerates disconnected and poorly connected users [2]. The difficulty of managing an optimistically replicated system is well-understood; it was most famously articulated in [3].

However, this model does nothing to address the third problem, and, if anything, exacerbates it by allowing concurrent and inconsistent updates to the system. Book-keeping to detect such concurrency problems can become significant, and the management of trust and security in the system is complicated. While good work has been done on managing and automatically resolving conflicts (see e.g., [4]), those who have worked hardest on this topic acknowledge how difficult it is to provide automatic conflict resolution in general, because the semantics of conflict resolution criteria are always heavily application dependent, and may depend on several replicated objects, nnot just one.

Our experiences building working applications and a shared *Information Commons* [5] collaborative space on top of a particular replication system (called *Shepherds*) has led us to evolve techniques to minimize and in some cases avoid these difficulties. Instead of trying to create rules for resolving all conflicts, we have taken a quite different approach, relying on careful information architecture to enable direct collaboration while minimizing the danger that two users will concurrently update the same replicated u-form.

It is our belief that the assumptions that our system depends upon are weak enough that our strategies will be applicable to other systems. There are many exciting recent developments in peer-to-peer systems that could support such collaboration strategies. Important systems include Chord [6], Pastry [7], and OceanStore [8]. Chord and Pastry supply only lookup and routing technology; OceanStore is a more complete system providing replication and object storage. Other higher-level systems include CFS (built on Chord) [9] and PAST (built on Pastry) [10]. Several of these systems use cryptographic hashes to identify immutable data as do we.

This paper describes the Shepherds system, the relevant engineering and design assumptions, and some interesting applications and collaborative structures.

The paper is organized as follows. Section 2 discusses u-forms and the collaboration model they enable, as well as a brief survey of systems we have developed using this model. Section 3 discusses engineering and design issues, including the particular choices and assumptions that our system makes, and the costs and trade-offs entailed. This discussion includes details about conflict detection, shepherdable indexes, trust management and digital signatures, and some information about our underlying replication and storage infrastructure, elaborating on the problem areas raised in this introduction. Section 4 describes how we layer higher level data structures upon this foundation to support messaging, shared maintenance of information collections, and collaborative publication and annotation, without running afoul of the pitfalls above. In addition, by layering our solutions at a higher level of the system, we give ourselves the freedom to more readily change infrastructure technologies. Finally, Section 5 ties these techniques together by briefly describing a *distributed collaborative geographic information system* that makes use of every layer of the system.

We conclude with a fundamental observation: optimistic replication systems scale well and provide a wonderful opportunity for very large-scale collaboration. But the guarantees provided by such systems are necessarily weak. Instead of trying to ensure that conflicting updates to the same u-form are resolved before users notice, we try to create information architectures than minimize the possibility of such conflicts occuring in the first place — while still empowering users to collaborate in real time and to make contributions about definitive data. Clever layering in the design and engineering of the system allow us to support very powerful applications without burdening the entire system with the maintenance of low-level guarantees.

2 U-forms and the Visage Collaboration Model

All well-designed replication and collaboration systems depend on a fundamental unit of storage and replication, such as a file, a data table, or an individual data record. In Visage [11] and all its descendents, including our Shepherds system, this unit is called a *u-form* [12, 1]. A u-form is an extensible bundle of attribute-value pairs identified by a universally unique identifier or UUID. U-forms are mutable (though sometimes changes to a u-form can be recognized as evidence of unauthorized tampering and rejected, as discussed in section 3.3). Attribute names are unique within a u-form. Attribute values can be any data: lists, dictionaries, binary blobs, and, of course, UUIDs of other u-forms. A UUID appearing as the value of an attribute is called a *relation*[1] from one u-form to another (sometimes thought of as a pointer or a reference). The ability to store

[1] This use of the word "relation" carries the traditional philosophical meaning of a relation between two *objects* (c.f. Aristotle, *Categories* Ch. 7) rather than the mathematical meaning of a relation between two *sets*, the meaning used in relational database systems. These uses are closely related: a relation between sets can be thought of as a *subset* of a Cartesian product, while a relation between objects is an *element* of a Cartesian product.

a UUID as a value in a u-form is important: it allows us to construct complex structures of u-forms by reference. A simple example might be a u-form whose `members` attribute is a list of UUIDs. Such a u-form is called a collection. We will see more sophisticated examples exploiting u-form relations throughout the paper.

The name u-form is an homage to Michael Dertouzos' e-form [13]. UUIDs are used quite commonly in many areas of computer science, particularly distributed systems. One contemporary reference relating UUIDs and the Web's URN system is [14]. A good overview of u-forms is found in [1]. A formal resemblance exists between the (UUID, attribute, value) triple in the u-form, the (URI, predicate, object) triple of RDF [15]. The u-form is an abstract datatype, and u-forms can be serialized using many formats including XML, though our implementations commonly use a recursive bytecode format called the Visage Standard Message Format, which is considerably more efficient than XML in practice.

Although u-forms *per se* are completely schemaless, schemata can be layered on top of the basic u-forms by including relations to special *role* u-forms [16].[2] Roles are u-forms that reserve the use of particular attributes and explain their intended interpretation. Storing relations to roles inside u-forms helps in making u-forms self-describing and introspective, which contributes to the effectiveness of replication and enables a single u-form to be a useful member of many heterogeneous datasets.

Updates to u-forms are simply changes to the attribute-value set. Changes to a single u-form can be made atomically in the local venue. The entire state of a u-form is replicated when replication occurs. Because users may be disconnected for a long period, we replicate only the state of a u-form, not a log of all operations on that u-form (which could be quite large).

U-forms support collaboration because they are mutable but have universal identity, so the underlying replication system allows a user to "subscribe to" a u-form: any changes are then propagated to that user. In this way multiple users can be interested in a shared set of u-forms. As they modify this shared set, the u-forms act as a common collaborative space and all the interested parties observe the evolution of that space.

2.1 A Brief History of the U-form Style

It is useful at this point to give a quick overview of some of the systems we have built over the years using u-forms. This demonstrates the utility of a simple, sound foundation, and will give us a set of concrete touchstones to refer to during the development of some of the more abstract ideas later in the paper.

Visage and Visage2 We first used this collaborative style in an information visualization system called Visage (see [11] and [18]), so we have come to think

[2] This use of a single syntax to describe both data and metadata is reminiscent of XML Schema's use of XML as its definition language. See [17].

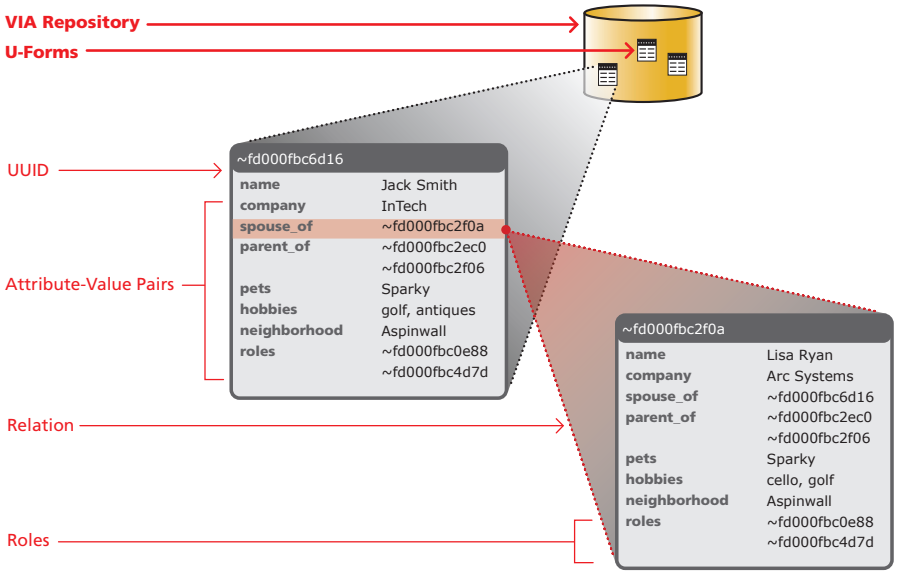

Fig. 1. U-forms and several related aspects of the system, including attributes, values, relations between u-forms, roles, and the VIA repository.

of it as the "Visage collaboration model." Visage pioneered an interaction model called information-centricity; the idea was to simultaneously provide:

- An easy-to-use direct-manipulation user interface.
- Fine-grained access to small units of information (scraps of text and rows of tables, not just whole documents).
- Data strongly separated from presentation to encourage multiple visualizations of the same data.
- Easy collaboration despite the fact that different users might prefer very different visualizations of the same data.

This last feature was added for Visage2, and cemented the definition of u-form that we use today. In particular, the fact that we use UUIDs to identify data objects (rather than locally scoped primary keys or location-specific URLs) enabled us to envision much more flexible collaboration scenarios [19]. Our commitment to the information visualization application domain made it critical that collaboration was mediated through changes to the data space, not simple sharing of the user interface (see Figure 2 and further discussion in [20]).

Visage2's implementation depended upon client-server technology, not peer-to-peer technology. Therefore replication was limited and controlled. It is the increase in scale, and corresponding relaxation of our controls on replication, that has led to much of the present work.

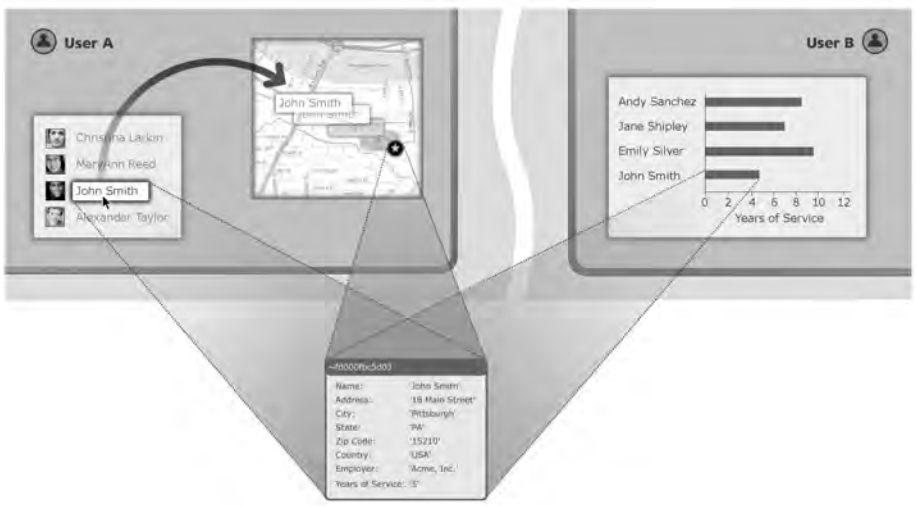

Fig. 2. Polymorphic visualization of a u-form.

CoMotion Success in the laboratory and in limited deployments of the Visage system encouraged us to commercialize the technology. In 1998 we spun off a company called MAYA Viz to develop a commercial product. The framework they created is called CoMotion. Products built on top of CoMotion are used by the U.S. Military for command, control, and logistics applications [21]. CoMotion applications are also used in the medical and energy industries. MAYA Viz was acquired by General Dynamics in 2005 and is now GD Viz [22].

The Geobrowser Meanwhile, we continued our research and became interested in very large scale, loosely coupled systems. This pushed us toward peer-to-peer replication as a scheme that can scale much better than client-server systems.

The Geobrowser [23] began as a collaborative GIS system built on our Shepherds peer-to-peer system, but has been evolving toward a more general information browsing and publishing system. The Geobrowser is discussed more thoroughly in Section 5.

The Community Directory and Buskarma U-forms are not limited to use in exotic information management domains. We have built a variety of useful web applications on top of the Shepherds architecture. These include the Community Directory family of websites [24, 25], winners of the Exemplary Systems in Government Award from URISA, an international organization of government information professionals. Another application is Buskarma, a public transportation website for the Pittsburgh area [26]. The underlying peer-to-peer nature of these sites is not very visible, since websites require a stable server to function, but the webservers themselves are connected through the Shepherds system to

the same shared information space that the Geobrowser uses. This information space is called the Information Commons [5].

2.2 Some General Observations about U-form Usages

U-forms are the unit of information in our system, but this same unit is often being exploited for a variety of purposes, including:

- Conceptual units, in 1-1 correspondence with objects in the world. This perspective is about semantics and the world.
- Collaborational units (e.g., data that users share and update). This perspective is about interaction design.
- Trust units, for digital signatures and encryption. This perspective is about social cooperation and trust.
- Basic data storage and transport units. This perspective is about engineering and system efficiency.

Each of these perspectives will be important as we explore collaboration techniques, and there are often trade-offs among them. For example, computing digital signatures imposes a non-trivial computational cost per u-form, so there are sometimes pressures to put lots of related data from the same publisher into a single u-form. On the other hand, an index to a dataset is often a single conceptual unit, but for effective use of resources, and to serve the purposes of a variety of users with different interests, this conceptual unit needs to be broken up across several related u-forms.

3 Design and Engineering Issues and Background

need to cite shepherds white paper

This section gives an overview of three basic computer science areas that are necessary ingredients for our collaboration architectures. These are the shepherds (replications agents) themselves, shepherdable indexes, and digital security issues. We do not intend to provide a detailed description of all aspects of the engineering of the Shepherds system. It would be beyond the scope of this paper, and would distract from the general applicability of the techniques described later. It is nonetheless important to give some sketch of how the Shepherds system works, in order to make clear the kinds of guarantees we require from the low level portions of the system.

One major theme of this section is that while our system allows u-forms to be updated anywhere in principle, in practice this is problematic for a variety of reasons. The techniques described later in the paper will show how we can perform collaboration while attempting to minimize the number of venues in which a given u-form is updated.

3.1 Shepherd Agents, Version Vectors, and Conflict Recognition

Shepherds is a large-scale peer-to-peer system that uses optimistic replication to facilitate collaboration through shared u-forms. Let us quickly examine some facets of the system.

The system contains a potentially very large number of venues. There is an extremely large number of u-forms; no venue is likely to contain a significant fraction of all u-forms. U-forms can be created at will by any user in any venue. U-forms can, in general, be updated in any venue (but we will discuss practical restrictions). The system uses peer-to-peer agents called *shepherds* to replicate u-forms amongst the venues. The shepherd agents are guided by application-specific business rules, the specification of which are the job of the designers of a particular system.

The precise details of the replication schemes used are beyond the scope of this paper, and, indeed, have varied significantly over the history of the system. One of the strengths of the techniques described here is that they do not depend strongly on the underlying replication schemes. This means that as new peer-to-peer techniques arise we can use them without major application-level changes.

The essential characteristics of the replication scheme are these:

- It is optimistic. That is, it does not defer replication or updates in an attempt to guarantee consistency. Instead, it replicates under the optimistic assumption that inconsistent updates are rare and can be detected and repaired when they are detected.
- If a user has subscribed to updates on a given u-form, she will eventually receive all relevant updates (though the order and timeliness of those updates is not guaranteed).

We use *version vectors* to detect concurrent updates [27]. A version vector is a list of counters: one counter is kept for each venue in which a given u-form has been modified. A venue's counter for the u-form is incremented when the u-form is updated in that venue. Therefore the storage required for version vectors scales with the number of venues in which a u-form is updated. Version vectors are a rather old technique; perhaps the original reference is [27] (though we believe it to have been independently invented many times). Version vectors are known to be a minimal representation for detecting violations of causal history [28]. An alternative mechanism for capturing causal history is the Hash History approach [29]. While it provides some interesting advantages over version vectors, it must be periodically pruned and can therefore lead to dangerous numbers of false conflicts in a large system.

If, upon replication of a u-form, we discover that one replica's version vector does not strictly dominate the other replica's, then we can conclude that concurrent modification has taken place. In some cases, the shepherd agents may be able to determine that two concurrent updates are not incompatible, so they are merged. Generally, however, concurrent updates result in a conflict. The conflicted u-form is annotated with pointers to the alternate versions.

Users may manually resolve conflicts, or applications may have specific automatic conflict resolution rules. Manual conflict resolution, while sometimes necessary, imposes a burden on the user, and automatic conflict resolution is only reliable in particularly well-understood application contexts (as also demonstrated by the designers of the Bayou system [4]). These problems grow worse as the system scales in size and as inter-dependencies between objects grow. This experience is consistent with the analytic results obtained by Gray in [3].

It is often better to avoid conflicts—in essence, to try to justify the optimism of our replication strategy through appropriate design. We will see u-form data structures that achieve this goal in section 4.

3.2 Shepherdable Indexes

Our repository infrastructure provides only very limited expressivity: given a UUID, one can update the associated u-form or subscribe to others' updates. There is no low-level provision for value-based search, only UUID-based lookup. This narrow expressivity gives us a great deal of freedom to choose different underlying storage and replication models, and improvements in storage and replication speed and reliability have a beneficial effect on all applications that use the Shepherds system.

Instead of hard-wiring a few value-based searches into the infrastructure, we build index structures that effectively enable the implementation of value-based searches by composing several UUID-based searches. An index is a data structure that efficiently maps a key or range of keys to one or more values (typically UUIDs).[3] We represent indexes by organizing u-forms into tree-like structures, and annotating the u-forms in the tree with key information dictating which sub-tree is relevant to the query at hand. For example, a typical "index node u-form" may contain information that an index reader will interpret as "follow relation u_1 for names beginning *A–L*, follow relation u_2 for names beginning *M–Z*."

A hallmark benefit of this approach is that index nodes are replicated on demand by the shepherds, just like any other u-forms. This means that once a user has performed a particular search while online, this search can be repeated in the future when offline. An index with this property is called *shepherdable*. For a detailed account of shepherdable index structures see [30]. We have implemented and routinely use B-tree style structures [31] for handling one dimensional queries and R-tree style structures [32] for multi-dimensional and geo-temporal queries.

One important use of indexes in collaborative systems is the support of what we call *virtual relations*. Recall that a relation is formed when a u-form contains a UUID (or set of the UUIDs) as the value for one of its attributes. It is not always desirable (for all the usual reasons: access control, scalability, etc.) to directly encode relations in one of the u-forms implicated. Instead, we can construct an index whose keys and values are both UUIDs. Such an index is a mapping from UUIDs to UUIDs: thus, a virtual relation. Virtual relations

[3] In some sense a collection is a trivial sort of index with linear query performance.

are used extensively in the Universal Genetics Database [33], a research project that uses the Shepherds system to represent and share publicly available genetic databases, and enables researchers to annotate and reuse particular genes from these databases for special purpose projects, without writing to the centrally administered database. We will use this technique in section 4.3 to support standoff annotation and commentary.

3.3 Digital Signatures, Security, and Trust

We have seen that modifying the same u-form in many venues increases the likelihood of conflicts, and increases the amount of book-keeping information we must store in the form of longer version vectors. These facts act as forces that encourage us to keep the number of venues modifying a particular u-form small.

In practical systems, another consideration also plays a major role. In many cases, not every user has the authority to modify every u-form in the system. It is not necessary for a user to consistently use the same venue, but it is common that a given user will only use a small number of venues. Therefore, if the number of users permitted to modify a u-form is small, the number of venues in which it will be modified is also likely to be small.

To protect u-forms from being maliciously or accidentally modified, we employ digital signatures. A good comprehensive introduction to technologies such as digital signatures, cryptographic hashing, and encryption can be found in [34]. Signatures are attached to u-forms as normal attribute values, and public key credentials are published in the system as u-forms. This allows shepherd agents and applications to verify that u-forms are only updated by their proper owner.[4]

Fully- or partially-immutable u-forms can also be created by using cryptographic hashes of the u-form content as part of the UUID for the u-form. These are of somewhat less interest in highly dynamic collaborative systems, but are very useful for storing certain kinds of data.

4 Collaborative Structures

Our experience with the Visage project demonstrated to us the value of using u-forms as a shared collaborative space that operates by allowing a set of users to update a shared set of u-forms. We wish to design publicly available systems that do not presuppose investment in major datacenters, and that also support collaboration between users with poor or intermittent connectivity. These requirements guided us towards investigating optimistic replication and a peer-to-peer approach.

[4] There remain a variety of potential attacks that we will not discuss in this paper. One is a spoofing attack that we call a "land grab" in which a malicious user creates a new u-form with a pre-existing UUID. This results in two apparently valid but competing signed u-forms. Such an attack must resolved through human arbitration or high quality automated reasoning. Another class of attacks involves denial-of-service, and is best addressed in the replication layer.

We have seen, though, three distinct pressures that require us to minimize the number of venues in which a given u-form is modified. One is the bookkeeping associated with version vectors and conflict detection; a second is the difficulty of lazily reconciling conflicts; and a third is the natural tendency to need to restrict write access to a subset of users. On the other hand, a legitimate worry is that restricting the original update-anywhere-anytime policy will impair collaboration.

In this section we examine techniques that help us to have our cake and eat it too. We describe a series of multi-u-form structures, along with conventions for using them, that let us perform many kinds of collaboration without needing to update the same u-form in many venues.

4.1 The Carrier Pigeon Protocol

One useful tool for collaboration is simple messaging: the ability for one user to send another user a message. Not only are the messages themselves handy for instant messenger or email style applications, but having such a technique available allows a distributed workflow: one user can request another user or automated agent to perform an action on his behalf.

As we have seen, our replication system does not presuppose that any two users are simultaneously connected. If each user is only intermittently connected, it may be that there is never transitive connectivity between them in the underlying network. However, our system does guarantee that replication eventually makes progress. Therefore the solution is to mediate messaging through u-form updates.

The most obvious way to do this is to have the two users who wish to communicate simply read and write to a shared u-form. This, however, leads to frequent conflicts.

Instead, we arrange matters so that each user has an outbox. User A writes to the A outbox, and listens to updates on the B outbox, while User B writes to the B outbox and listens to updates on the A outbox. We arrange the protocol as follows:

1. User A desires to send a message to user B. So User A writes an attribute into his outbox called `message_X`, where `X` is any locally unique token he chooses (a number is a reasonable choice). The content of the message is simply the value of the attribute `message_X`. User A also writes an attribute called `current_message_id` whose value is `X`.
2. Since User B is listening for updates on User A's outbox, User B will eventually see the message. She can then decide upon a response and write an attribute `response_X` into her own outbox, where `X` matches the message identifier token chosen by user A. The value of the attribute `response_X` is the response to User A's message.
3. Since User A is listening for updates on User B's outbox, he will eventually see the response. At that point, he can remove the `message_X` attribute from his outbox, keeping the outbox size reasonably small.

4. User B will eventually notice that User A has removed the `message_X` message. Consequently, User B can remove the `response_X` response from her outbox.

The important thing to notice about this technique is that it is guaranteed to eventually succeed as long as the underlying replication scheme—no matter what it is—can make progress. Moreover, no single u-form is modified by more than one user, so no conflicts occur, no security policies are violated, and no extra version vector housekeeping is incurred.

4.2 Collections and Recursive Collections

As we mentioned earlier, it is easy to construct a u-form that holds references to many other u-forms. We call such u-forms collections, and they are very useful for defining datasets, that is, sets of u-forms that a given user community is interested in.

Many of our visualization applications depend strongly on shared collections. For instance, a military commander may be examining a collection of his aircraft in a map visualization, while, simultaneously, a logistics officer is viewing the same set of aircraft in a chart showing fuel and ammunition supplies. It is very useful for these visualizations to be encoding the same collection u-form, so that if it is modified (say, to add or remove an aircraft), both visualizations are updated.

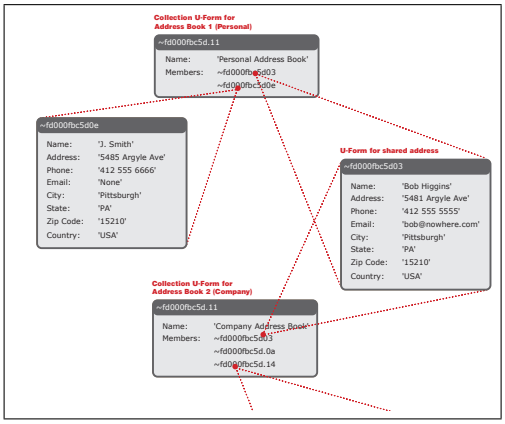

Fig. 3. Several collections

Collections, however, are very difficult to update consistently in a distributed fashion if many writers are involved. There is no unique "ground truth" method for producing a single collection by resolving several edits in different venues, especially if a globally consistent ordering is to be preserved. Moreover, the expense of version vectors is incurred and many users must have the authority to update the collection u-form.

One solution to the problem is to make one user the owner of the collection, and have any other user that wishes to modify the collection send a message (see section 4.1) to that user. This, in effect, converts our multi-master distributed system into a single-master client-server system *for this particular piece of data*. The nice thing about this approach is that any collection semantics can be supported, and we can assign ownership however we like for various pieces of data

and datasets. The downside is that if the owner of the dataset is unreachable then no updates can be performed.[5]

Another solution is to factor a single collection u-form into a *recursive collection*. A recursive collection is a collection u-form whose member u-forms are themselves collections, together with some annotation to distinguish between the intent of adding a single collection or adding all the *members* of that collection. We can assign ownership of each "child" collection to a different user, and we can interpret the recursive collection as containing the union of the members of its children.

This works well provided ordering is not crucial to the application, and provided no single user needs to be able to fully delete an item from the recursive collection (since she cannot delete it from other people's "child" collections, only her own). Version vectors are kept small and digital signatures are easy to manage, because there is a one-to-one relationship between users and the collection u-forms they modify. In some situations it can be argued that managing the security policy, the conflict resolution, and the large version vector on a single collection provides superior performance to tracking all the child collections in a recursive collection. It must be noted, though, that safety demands that version vectors be kept forever, whereas only currently interested users need maintain child collections.[6]

Setting up the recursive collection can be handled using a single-writer-via-carrier-pigeon approach, since changes to the set of interested users (who also need to write) are probably comparatively infrequent.

It is likely that recursive collections can be generalized with additional annotations to support approximate or limited guarantees on the ordering of elements, but we have not yet experimented with this concept.

4.3 The Publication and Annotation Mechanism

Virtual relations, described in section 3.2, arm us with a powerful tool. A user can associate one u-form with another without requiring write access to either u-form. Instead, the user simply needs write access to a virtual relation index that supplies the mapping or, more commonly, someone to add the virtual relation on the user's behalf.

We have established a convention for using such virtual relations to perform collaborative enrichment of information. (The application of this mechanism to annotation of linguistic corpus data is described in [35].)

We define an *author* to be a user who creates u-forms and may be interested in annotating existing u-forms.

---

[5] It should be noted, though, that the carrier-pigeon message approach will automatically queue messages until connectivity is restored, so updates are not lost, only deferred.

[6] Technically, it is possible to prune version vectors if a safe distributed transaction can be performed over every venue in the system. Unfortunately, for internet-scale systems with intermittent connectivity such a transaction is essentially impossible.

A *publisher* is some agent who vouches for the veracity of a piece of information. Each publisher also maintains a well-known virtual relation index. An author may be his own publisher, or may cooperate with a well-known publishing organization.

A *theme* is a u-form defining an area of interest for some community of users. These could be created in an ad-hoc fashion, or worked out in a standardization process. It depends on what the publishers and users find most convenient; the system doesn't care what mechanism is used to agree upon the meaning of themes—they are treated as unique labels.

An *annotation* is a u-form that comments upon another u-form with respect to some theme. (Because an annotation is itself a u-form, it might be the target of further annotations. The same goes for themes themselves.)

Each publisher maintains an index whose keys are UUIDs of u-forms that are being annotated (it is thus, by our earlier definition, an index of virtual relations). The values in the index are UUIDs of collections that store UUIDs of annotation u-forms, sorted by theme.

A user who wishes to publish an annotation on an existing u-form creates the annotation u-form, and uses a carrier-pigeon channel to ask the publisher to relate the new annotation to the existing u-form through the publisher's index[7]. To maintain the integrity of the index, a publisher will typically copy the user's annotation to u-form that is signed by the publisher, and publish this version, rather than publish the user's original annotation (this avoids "bait and switch" abuses).

Similarly, if a user knows of a publisher's index, and a source u-form of interest, she can efficiently find all the themes and annotations published by that publisher for that source u-form by looking up the source u-form UUID in the publisher's index.

In practice, this technique supports rich stand-off annotation and commentary without requiring users to modify each others' u-forms. We will discuss an existing application of this mechanism in section 5. An advantage to using stand-off annotation over direct modification of data is that competing viewpoints and opinions can be captured and expressed. This is a much more natural mode of discourse in many respects than a tug-of-war over a single mutable information object.

Stand-off annotation is only a recent description of the much older scholarly practice of citation with commentary. This technique can be found in ancient Greek writings and in early scriptural scholarship. The tendencies apparent in such practices are the same as those we see today: the more definitive a text, the more people wish to comment upon it, and the less likely they are to be able to edit the definitive version with inline annotation or markup.

[7] The publisher, of course, is free to pursue any policy for deciding whether or not to publish an annotation. And, of course, a user may self-publish if no publisher is willing to work with him.

5 A Comprehensive Application: Collaborative, Distributed GIS

Our Geobrowser application [23] leverages all these techniques to support collaborative, distributed GIS. What do we mean by that? A Geobrowser user can browse and explore a large shared dataset of u-forms describing political and physical features of the Earth. (This dataset, called the Information Commons Gazetteer, is quite complex and is built from many freely available sources. Its structure is described in detail in [36].) The user can also create her own data and datasets and share them. She can also enrich existing datasets through the publication mechanism.

Each Geobrowser contains a database node that is part of the Shepherds system. Consequently, the Geobrowser application works well whether or not it is currently connected to the internet.

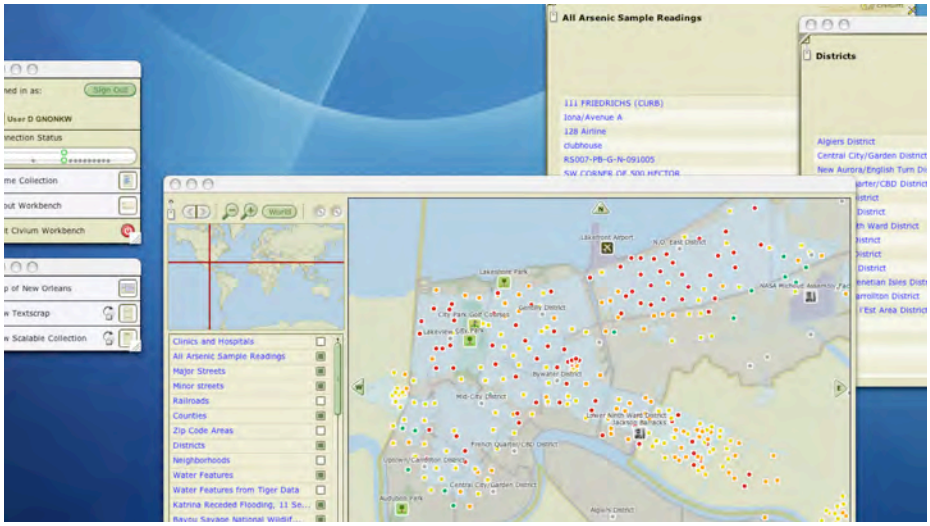

Fig. 4. The Geobrowser

Let's walk through some of the techniques we have discussed and identify how they support specific Geobrowser features.

- U-forms are used to represent all of the data and much of the implementation of the Geobrowser application. It is thus "self-updating" since the underlying replication mechanism will update relevant u-forms transparently.
- Digital signatures are created and checked automatically by the tools in the application using the active user's credentials.
- Carrier-pigeon protocols are used to support creating new Geobrowser users (a master index of users and public key credentials is maintained).

- Collections are used extensively to structure and organize data. Users can easily create and share new collections (as well as other sorts of u-forms).
- U-form based indexes are used to accelerate access to over 5 million physical and political features of the world. Both geo-temporal and string-search indexes are provided.
- The publication mechanism can be employed by users to provide free-text comments on any u-form in the system. The publication mechanism is also used to associate more structured data from multiple sources: for instance, data from the Wikipedia has been associated with the Information Commons Gazetteer through the publication mechanism.

The Geobrowser is in active development and is becoming a tool that reaches far beyond GIS. It is being used to as a content management system for the Community Directory websites [24, 25], and to support research and collaboration in the biomedical domain [33].

6 Conclusion

U-forms as a shared collaborative space are a valuable and powerful tool. Optimistic peer-to-peer replication allows us to increase the reach of u-form based systems by improving scalability, and by allowing us to operate in disconnected or poorly connected environments.

This capability comes at a price, however. Managing conflict-detection bookkeeping in the form of version vectors can grow expensive over time as u-forms are updated in many venues. Lazy conflict resolution increases in difficulty as the number of conflicts grows and as the complexity of u-form structures grows. As the number of users grows, the desire for access control to protect data from accidental or malicious tampering also argues against overpermissive editing policies.

To continue to support rich collaboration despite these challenges, we have developed a variety of higher-level structures and protocols on top of our basic u-form storage and replication system. The high-level techniques demand only very weak guarantees from the underlying system, allowing us the freedom to exploit technological progress as it becomes available.

These high-level structures support simple user-to-user messaging, multi-writer datasets, and rich multi-user annotation and elaboration of data. We believe that other researchers interested in large-scale collaborative systems may also find these patterns efficient and effective.

Future work will continue to improve the efficiency and utility of these structures without compromising their ability to scale. Simultaneously, we are continuing to explore new replication and storage technologies for the underlying system. We are also developing end-user applications and tools for a variety of domains, including bio-informatics and GIS.

References

1. Lucas, P., Senn, J., Widdows, D.: Distributed knowledge representation using universal identity and replication. Technical Report MAYA-05007, MAYA Design (2005)
2. Saito, Y., Shapiro, M.: Optimistic replication. ACM Computing Surveys **37** (2005)
3. Gray, J., Helland, P., O'Neil, P., Shasha, D.: The dangers of replication and a solution. In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data. (1996) 173–182
4. 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. In: Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP-15), Copper Mountain Resort, Colorado. (1995)
5. Lucas, P.: Civium: A geographic information system for everyone, the Information Commons, and the Universal Database. In: Vision Plus 10, Lech/Arlberg, Austria (2003)
6. Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A scalable Peer-To-Peer lookup service for internet applications. In: Proceedings of the 2001 ACM SIGCOMM Conference. (2001) 149–160
7. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science **2218** (2001) 329
8. Kubiatowicz, J., Bindel, D., Chen, Y., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., Zhao, B.: Oceanstore: An architecture for global-scale persistent storage. In: Proceedings of ACM ASPLOS, ACM (2000)
9. Dabek, F., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I.: Wide-area cooperative storage with CFS. In: Symposium on Operating Systems Principles. (2001) 202–215
10. Druschel, P., Rowstron, A.: PAST: A large-scale, persistent peer-to-peer storage utility. In: Proceedings of HOTOS (Hot Topics in Operating Systems). (2001) 75–80
11. Roth, S., Lucas, P., Senn, J., Gomberg, C., Burks, M., Stroffolino, P., Kolojejchick, J., Dunmire, C.: Visage: A user interface environment for exploring information. In: Proceedings of Information Visualization, San Francisco, IEEE (1996) 3–12
12. Lucas, P., Senn, J.: Toward the Universal Database: U-forms and the VIA Repository. Technical Report MTR02001, MAYA Design (2002)
13. Dertouzos, M.: What Will Be. Harper, San Francisco (1997)
14. Leach, P., Mealling, M., R.Salz: A UUID URN namespace. Technical report, The Internet Society (2004) Current draft, awaiting approval.
15. Manola, F., Miller, E.: RDF primer (2004)
16. Lucas, P., Widdows, D., Hughes, J., Lucas, W.: Roles in the universal database: Data and metadata in a distributed semantic network. Technical Report MAYA-05009, MAYA Design (2005)
17. van der Vlist, E.: XML Schema. O'Reilly (2002)
18. Higgins, M., Lucas, P., Senn, J.: VisageWeb: Visualizing WWW Data in Visage. In: Symposium on Information Visualization (Infovis), IEEE (1999) 100–107
19. Lucas, P.: Mobile devices and mobile data: Issues of identity and reference. Human Computer Interaction **16** (2001) 323–336
20. Bishop, D., Lucas, P.: Polymorphic collaboration: Beyond relaxed WYSIWIS in Visage-Link. Technical Report MTR-02007, MAYA Design (2002)

21. Project, D.: Command post of the future (CPOF) (2005) http://www.darpa.mil/ato/programs/CPOF/DT.htm.
22. General Dynamics: GD Viz (2005) http://www.gdviz.com/.
23. MAYA Design, Inc.: Civium Workbench (2002) http://civium.maya.com/.
24. Allegheny County Department of Human Services: HumanServices.net (2006) http://www.humanservices.net/.
25. A-Plus Schools: Pittsburgh After School (2006) http://www.pghafterschool.com.
26. MAYA Design, Inc.: Buskarma (2002) http://www.buskarma.com/.
27. Parker, D., Popek, G., Rudisin, G., Stoughton, A., Walker, B., Walton, E., Chow, J., Edwards, D., Kiser, S., Kline, C.: Detection of mutual inconsistency in distributed systems. IEEE Transactions on Software Engineering **SE-9** (1983) 240–247
28. Charron-Bost, B.: Concerning the size of logical clocks in distributed systems. Information Processing Letters **39** (1991) 11–16
29. Kang, B.B., Wilensky, R., Kubiatowicz, J.: Hash history approach for reconciling mutual inconsistency in optimistic replication. In: 23rd IEEE International Conference on Distributed Computing Systems (ICDCS'03). (2003)
30. Higgins, M., Widdows, D., Balasubramanya, M., Lucas, P., Holstius, D.: Shepherdable indexes and persistent search services for mobile users. In: 8th International Symposium on Distributed Objects and Applications (DOA 2006), Montpellier, France (2006)
31. Sedgewick, R.: Algorithms in C. Addison-Wesley (1990)
32. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: Proceedings of SIGMOD. (1984) 45–47
33. Widdows, D., Barmada, M.: The Universal Genetics Database: Information sharing in genetics and beyond. BioTech International **18** (2006) 11–13 (Byline article).
34. Schneier, B.: Applied Cryptography. 2nd edn. John Wiley and Sons (1996)
35. Balasubramanya, M., Higgins, M., Lucas, P., Senn, J., Widdows, D.: Collaborative annotation that lasts forever: Using peer-to-peer technology for disseminating corpora and language resources. In: Fifth International Conference on Language Resources and Evaluation (LREC 2006), Genoa, Italy (2006)
36. Lucas, P., Balasubramanya, M., Widdows, D., Higgins, M.: The Information Commons Gazetteer: A public resource of populated places and worldwide administrative divisions. In: Fifth International Conference on Language Resources and Evaluation (LREC 2006), Genoa, Italy (2006)