

Unifying User-to-User Messaging Systems

Unification of user-to-user messaging systems facilitates message exchange independent of time, place, protocol, and end-user device. This article describes an approach to unification that is based on introducing a middleware layer instead of employing gateways. It entails a single system that provides common services such as email, fax, and short messaging, but that can also enable novel services that current messaging systems can't support. The authors also describe how the model can be efficiently implemented on a global scale.

**Jan-Mark S. Wams
and Maarten van Steen**
Vrije Universiteit, Amsterdam

Although user-to-user messaging systems such as Internet-based email have become virtually omnipresent – letting users reach each other with devices ranging from i-mode phones to answering machines – interconnectivity between different platforms remains limited. Users must select an appropriate messaging system each time they want to send a message, taking into account the sender's and receiver's locations and what systems are available to both parties at the time. Misjudgments can easily delay or derail messages, even when the recipient could have been reached instantaneously. For an important message, a user might even want to use several messaging systems in parallel to increase the delivery's odds of success.

Although users have adapted to the lack of interconnectivity between systems, this situation is clearly unsatisfactory. Rather than having to select a messaging system, users should be able to follow one

simple approach to send messages. Unification is the key word here – of both *data representation* and *data transport*. Many researchers have focused on unifying data representation, as exemplified by such platform-independent standards as ASCII, MIME, and XML. Instead, we focus on data-transport unification, a vast and chiefly unexplored research area.

Building gateways to interconnect existing messaging systems seems an obvious approach to unification, but this is not the approach we have decided to follow. If done at the level of the underlying messaging systems instead, unification could enable a single system to provide the same messaging services that all current messaging systems provide separately, as well as facilitating combinations of services that can't currently be supported. In this article, we present a novel unified messaging system that supports services such as those offered by email, fax, short messaging, instant messaging,

and voice mail. We also describe an accompanying middleware system design to show that such a unified model can be implemented efficiently on a worldwide scale.

Unification through Middleware

Our proposed middleware solution is based, in part, on the fact that pair-wise integration of n different platforms would take $O(n^2)$ effort. In contrast, unification by means of a common message-transport system (namely, middleware) reduces these efforts to $O(n)$. A single user-to-user messaging system makes it much easier to support new combinations of messaging services. For example, users could change an email-like thread into a NetNews-like group.

In designing a unifying messaging system, we must create a single messaging model that captures the concepts underlying current messaging services. Table 1 shows the taxonomy we developed, characterizing existing messaging transport systems' capabilities along four different dimensions.¹

As an example of how this taxonomy can be applied to existing systems, Internet email never removes messages, provides only unidirectional communication, lets users send messages to anybody in the world (with an email address), and supports multicasting (that is, it can handle multiple recipients). NetNews, on the other hand, removes messages after they expire, uses a single communication channel to exchange news items (that is, newsgroups are duplex), allows only subscribers to a specific news group to be potential receivers, and broadcasts messages to recipients.

Our taxonomy establishes an important requirement: unified messaging should permit every useful (time, direction, audience, address) tuple. This effectively means that the unified messaging system should support any type of messaging service that the taxonomy can classify.

In addition to satisfying this requirement for *maximum adaptability*, we set out to build a unified messaging middleware layer that could scale well in terms of number of messages and users, and in the dispersion of users and resources across the Internet (and other networks).²

In contrast to existing approaches, our system puts the receiver in control. This choice comes from another simple but important requirement — namely, that a unified messaging system should hinder unsolicited messages, lest it become unusable. On the other hand, we feel some things are best handled in an end-to-end fashion. We thus

Table 1. Taxonomy of messaging transport-system capabilities.

Dimension	Issue
Time	How long does the system store messages?
Direction	Is message transport unidirectional or bidirectional?
Audience	Who are potential recipients?
Address	Does the system support unicast, multicast, or broadcast?

decided not to explicitly support presence, secrecy, authentication, or nonrepudiation, but have also been careful to use mechanisms and policies that will not hinder their support.

System Architecture

Our system is based on two kinds of messaging objects: *targets* and *targeted immutable short messages* (TISMs). A target is a generic name for an entity such as a mailbox, newsgroup, message channel, or chat room. It can best be thought of as a storage place for messages. As we'll discuss, individual users can have many targets.

TISMs are messages that are directed at targets rather than users. Targets, in their role as storage providers, hold on to TISMs until they expire. Once they pass into our messaging system, TISMs can no longer be modified. In other words, they are immutable by design. Both object types have associated expiration times. Every object has a fixed *home*, which always keeps a copy until it expires.

What sets our messaging objects apart is that each can have an associated *replication strategy*. In other words, our messaging objects encapsulate not only data and operations but also strategies that dictate how to migrate, distribute, and replicate them in the messaging system. We carried this design choice forward from our previous work on the Globe distributed infrastructure.³ By allowing replication at the object level — avoiding the need for a fixed, system-wide replication strategy — we achieve maximal adaptability, which we need in order to mimic existing messaging services' behaviors. Equally important, this approach provides an excellent means to achieve true scalability through differentiated strategies, as we've demonstrated previously for Web documents.⁴

Assigning each object its own replication strategy requires that special mechanisms are built into the middleware. We decided to keep matters as simple as possible. Every target or TISM object is created by a *user agent* (UA), which plays a role comparable to a UA in an email system. A collec-

tion of *target agents* (TAs) handles message storage, transfer, and replication.

By its nature, a UA can be online and offline sporadically, but ensuring that TAs can access TISMs and targets requires high availability. In practice, this means TAs should always be online. (To compensate for a TA's potentially low availability, a user can assign a replication strategy to a TISM or target object, which will increase its availability.) In this respect, our scheme somewhat resembles the Internet email system, in which an Internet service provider (ISP) runs a mail server and takes responsibility for its high availability. Also like the ISP model, every UA has one *primary* TA, through which all communication flows; this becomes the *home TA* for the objects the UA creates. Note that numerous UAs can share a single primary TA.

A TA will act as a home TA only for a target or TISM authenticated by a UA with the proper access permissions. The UA also handles end-to-end encryption, which enhances the system's overall security and relieves the TA from access control. Instead, a TA's main role is replication: if a UA or another TA requests a copy of a target or TISM object, the TA simply provides one. This approach is safe because UA-to-UA encryption generally renders a copy of an object useless without the proper decryption key.

Replication

To understand replication's importance in our system, consider the following simple scheme for message transportation: a TA could simply store each new object it received from any known UA and give out a copy of any requested object. Because each TA holds on to those objects for which it is the home, a UA that needed an object could contact the object's home TA to retrieve a copy.

This simple scheme produces a "working" messaging system, although it is highly susceptible to TA failures and might not scale very well (due to the impact of high latencies, for example). We can easily improve the design by adding caching capabilities to the TA. A UA could thus contact its primary TA to request a given object; the primary TA would start with a cache lookup and then request a copy from the object's home TA if it couldn't find one locally. Upon receipt, the primary TA would forward the object to the requesting UA, caching the object to allow the TA to service subsequent requests (for example, by another UA) from its cache. Note that this scheme silently assumes a relatively efficient connection between the UA and

its primary TA, but this is not unreasonable given that TAs are usually provided by ISPs or through organizations' LANs.

Piggybacking replication data on objects would enable replication developers to devise advanced retrieval schemes that further enhance system performance and robustness. Replication can allow UAs and TAs to preemptively send objects – to update targets, for example. (We will elaborate on target replication later.)

There are at least three ways to implement per-object replication. First, we could let an object run arbitrary code written in Java or some other language. This would make the target and TISM objects into full-fledged agents, but it would also raise many security issues and make the objects relatively heavyweight because they would need to transport code along with the data. To circumvent these problems, a second option would be to design a compact special-purpose replication language for use in expressing replication strategies. Given that one of our design goals was simplicity, however, we opted for a third alternative: we program several predefined replication routines in each TA and assign each object a replication-strategy index number to indicate which should be executed. This creates a flexible system in which an object can also carry arbitrary replication data under the replication strategy's full control.

Our design choices provide users considerable freedom in selecting replication strategies. Because each object has a home TA that is responsible for its availability, there is no need for hard guarantees concerning timely or reliable message delivery through replication. When a TA can't find an object locally, it can always contact the object's home to retrieve a copy.

The TISMs' immutability lets us escape many problems related to updating replicated data. For example, we avoid various consistency and synchronization issues because there are no write-write or read-write conflicts to resolve: writes simply can't occur after a TISM is inserted into the messaging system. Targets are mutable, but in a special way: they allow UAs only to add TISMs. In this sense, they behave similarly to append-only logs in file systems (although the order in which TISMs are added is not important for targets). Furthermore, targets (and TISMs) have expiration times, after which the system automatically removes them. These two properties make replication much easier to handle.

The immutable and add-only characteristics

also simplify the TA's cache strategy because they eliminate the notion of stale data. An object's availability through its home TA allows for any form of cache pruning. In other words, TAs needn't ensure that at least one copy of an object remains available in the system; the object's home TA takes care of that.

TA and UA Design Details

As Figure 1 illustrates, the TA's structure is relatively straightforward. The TA runs one or more threads per replication strategy, plus *fetch* threads to collect objects from networks. Each replication thread has a queue from which it takes objects to be processed. The fetch thread puts each object in the proper queue. For output, a TA also has queues and output threads.

Furthermore, one or more replication threads looks at objects in the storage and cache and, if need be, puts them on a replication thread's queue. This guarantees that objects will be processed for replication. Finally, a cleaning thread deletes expired objects and cleans the cache. Replication-strategy threads can store or cache objects and then send them out by putting them at the tail of the output queue. To allow for instant-message exchange and presence notification, the TA needs a notion of how to handle UAs going online on a target; it might need to track which UAs are currently communicating through a given target. Replication strategies supporting instant messaging can also insert objects at the head of the queue for immediate dispatching.

Target and TISM Objects

Target and TISM objects are similar. Each contains a system-wide unique ID, a replication-strategy index, a creation and expiry date, a short text, a payload, and replication data. The TA composes the unique ID from the object's home address and a locally unique index.

Targets and TISMs are represented by sets of distributed local objects, which at minimum carry the unique ID and two dates. Given the unique ID, the system can retrieve any other part of the object from its home TA.

For a TISM object, the payload contains the actual message, and the short text contains the subject line. Targets can also have subject lines, which can help users with organization. A target's payload is, basically, a list of TISMs.

Because targets are mutable, the system must compare and update them. They can become so

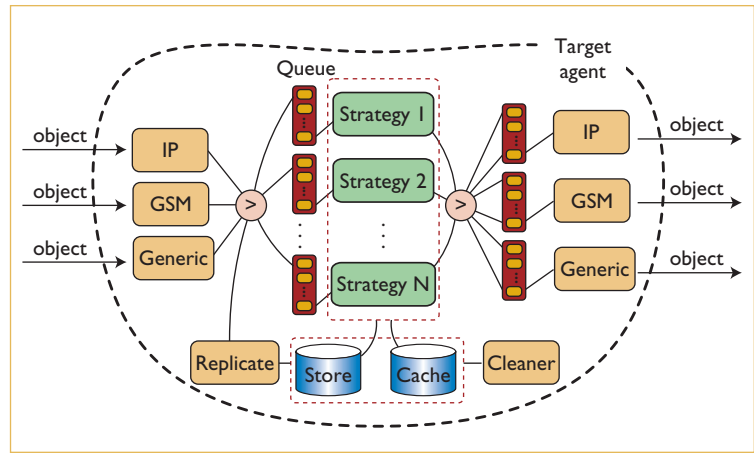


Figure 1. A target agent's internal design. Objects come in through various communication channels and are queued for further processing according to their associated replication strategies. Threads process objects and forward them across different communication channels. Stored and cached objects can also be appended to input queues.

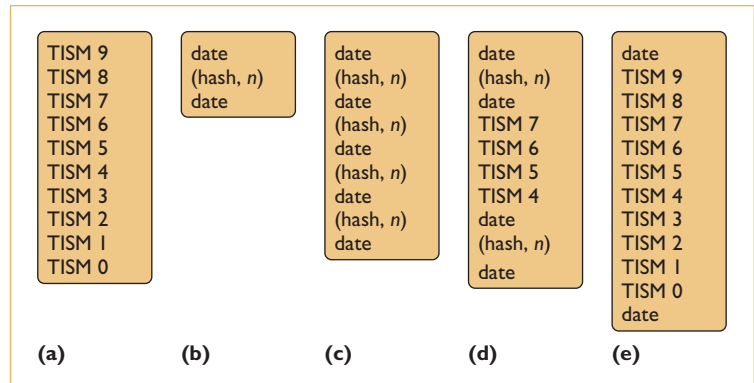


Figure 2. Five ways to represent a list of targeted immutable short messages (TISMs). (a) A plain list of TISMs. (b) A minimal-sized content list. (c) A content list containing only hash values. (d) A content list containing hash values and TISMs. (e) A TISMs-only content list.

arbitrarily large, however, that it can be impractical to exchange target objects even using minimal representations of TISM objects. We therefore designed a scheme to cut down the target object's memory footprint: by ordering the TISM list by creation date in the target payload (as shown in Figure 2a), we make the "old" part of the list more predictable and, thus, easier to compress for performance reasons. (Imagine a real-time messaging system sending a list of 1 million TISMs per second over the network for comparison with some other list to check for anything new.)

To facilitate efficient list comparison and updating, we introduce a *content list*. Between

Related Work in Unified Messaging

Unification of user-to-user messaging, also referred to as “unified messaging,” is becoming increasingly important given the various non-interoperable ways that users can send and receive messages. Because most user-to-user messaging is either Internet- or telephony-based, messaging developers and providers sometimes narrowly define unification as Internet-based integration of telephony and data services.¹ Such thinking drives the development of commercial unified-messaging offerings, which typically provide a centralized place for users to collect all messages — although usually limited to email, voice mail, faxes, and phone-text messages.²

Current approaches unify messaging by defining a single message box for each user in which to collect messages from different

services. Systems unify messages at a fixed endpoint on the receiver’s side.

In contrast to these methods, we propose implementing unification at the system level — that is, at the level of message transportation. In fact, our system has no collect-endpoint. Instead, it stores messages at their starting points and makes them available to recipients through replication.

Our approach solves a problem inherent to endpoint unification schemes, in which the user finds out who the sender is only after receiving a message. In our system, senders target not recipients but one of their many specialized message boxes, which take the roles of chat room, news group, and so on. Recipients can create multiple message boxes to give different access rights to various people and groups,

thus maintaining fine-grained control over who they receive messages from. In particular, a user (the recipient) can give another user (the sender) a separately created target so that the latter can post messages.

In short, we strive not merely to connect but to unify most existing user-to-user messaging systems into one. We believe this necessitates a multiplatform user-to-user messaging middleware layer, which no existing unified-messaging proposal offers.

References

1. H.J.Wang et al., “ICEBERG: An Internet-Core Network Architecture for Integrated Communications,” *IEEE Personal Comm.*, vol. 7, no. 4, 2000, pp. 10–19.
2. C.K.Yeo et al., “Unified Messaging: A System for the Internet,” *Int’l J. Computers, Internet, and Management*, vol. 8, no. 3, 2000, pp. 1–14.

each pair of creation and expiry dates in this descending list appears either a list of TISMs, a hash of a TISM list, or the number of TISMs in the hash. A content list of a particular TISM collection of can range in size from a few bytes to just over the size of that TISM collection when each TISM is listed explicitly.

Consider an example in which TA_A and TA_B both have local instances of the same target T . Let’s say that T ’s replication strategy prescribes that the lists should be unified. If we assume that TA_A has some TISMs that TA_B does not, the update proceeds roughly as follows.

First, TA_A sends a minimal-sized content list to TA_B (Figure 2b). TA_B ’s executed replication code calculates the hash of its TISM list and compares it with the hash from TA_A . If there were no differences, no further action would be needed. In this example, however, TA_B ’s executed replication code calculates several hashes of partitions of its TISM list and sends them in a content list to TA_A (Figure 2c). Upon receipt, TA_A executes the replication code that calculates the hashes on its TISM list and creates a new content list in which hashes that differ are replaced by the actual TISM sublist from that period (Figure 2d). TA_A sends this expanded content list to TA_B , which checks the hashes, finds no differences, and merges the new TISMs into its own list. TA_B does not need to send any reply to TA_A once it has updated its TISM list

as both parties know they now have the same TISMs. Note that the TAs don’t keep track of sessions; they react to each request separately, as in connectionless services.

Security

As we mentioned, UAs handle encryption in our approach. One (*post*, *read*) key pair is associated with each target, and a posting UA uses the *post* key to encrypt every TISM belonging to the target. (Actually, the TISM is encrypted with a random symmetric key, which is encrypted with the *post* key.) To decrypt the TISM, a UA needs the matching *read* key.

By controlling the distribution of keys for reading and posting TISMs, users control access to the targets they create. Because the UA can create targets at will, users can discard and replace targets, if the *post* key is abused in any way.

Users distribute keys out-of-band, just like email addresses. The messaging system itself will likely be a primary vehicle for key distribution. Imagine, for example, that a user puts a target ID and corresponding *post* key in some public place (a Web page, for example), and another user uses them to post a message in the advertised target, containing a reference to some newly created target with its own (*post*, *read*) key pair. The users could then communicate using this “dedicated” target; under normal circumstances, no other user would be able

to join the discussion. An email address on a Web page gives away much more control.

To prevent malicious and malfunctioning servers from filling a target with nonsense messages, a (`private`, `public`) key pair is also associated with each target. The user distributes the `private` key along with the `post` key, and posters use it to sign the encrypted TISMs. The target's home TA stores and gives out the `public` key, which other TAs can use to verify signatures and to filter out denial-of-service data.

Message Flow

To illustrate how messages flow through our system, consider the following scenario. Assume user A has just given a subject and message text to UA_A , which has the `post` key and ID for a target T . UA_A builds a TISM object with the encrypted subject as short text and the encrypted message text as long text. It then inserts the number associated with a “hold this” strategy, effectively requesting that the TA be this TISM's home, and adds a signed secure hash as replication data and as a second replication strategy. UA_A sends this TISM to A's primary TA, TA_A .

Upon receiving this TISM, TA_A checks the signed hash to ensure that this object is indeed coming from UA_A . It fills in the `creation date` field, generates a unique ID for the TISM, and then replaces the message's “hold this” strategy with the second strategy from the replication data. TA_A puts the resulting object in the store and back on the input queue for further replication. It then creates a copy, signs it, and sends it back to UA_A , which now knows that its primary TA will act as the TISM's home TA.

To finally post the message to the target T , UA_A also creates a local target object based on target T 's ID and an “update list” strategy, telling T to update its content. It then creates a simple content list that contains the TISM's unique ID. UA_A sends this target object to TA_A .

TA_A will send this target object on to other TAs, as dictated by the target's replication strategy, which the TA is likely to know about from the copy of that target in its store or cache. If the TA doesn't have a copy of the target, it will have to retrieve one before it can send the target object to other TAs. At some point, another UA, say UA_B , will request target T from TA_B . If user B so wishes, UA_B will retrieve the encrypted short text or message text via TA_B . If UA_B has the `read` key for target T , it can display the message to user B.

We discuss these processes in greater detail elsewhere,² particularly regarding the replication between TAs, which depends on the replication strategy chosen by target T 's creator.

Mimicking Existing Systems

Our messaging system can mimic existing systems like email and instant messaging. To mimic email, for example, the UA could distribute a newly created target's `post` key to the general public, effectively creating a mailbox that anyone can post to.

The UA could create a chat-room-like target by distributing `post` and `read` keys and assigning the target an “instant forward” replication strategy. Users would simply request the target from their primary TAs in order to go online in it.

Our unified messaging system can also mimic other existing messaging systems such as Net-News and Web logging,⁵ but the model offers much more. Some might find it surprising that such a simple architecture could facilitate novel and complex forms of messaging; with proper key distribution, however, our system can implement almost any type of messaging. Let's consider a few examples.

If there were one target for which all users had `post` and `read` keys, it would create a form of “say all, hear all” (SAHA) messaging that would make it hard to deny individual users access. Because TISMs are stored at the poster's home TA, our model makes this type of high-volume target feasible without a huge central server.

To add *moderation* to such a system, we could let one user (the moderator) create a new target and post its `read` key (and ID) in the SAHA target. This moderator would forward a selection of the TISMs in the SAHA target to the new target.

Of course, sharing the new target's `post` key with a select set of people would let a small group moderate the original SAHA target. The beauty of this scheme is that any user can start a moderated target based on the SAHA target. Even if the moderated target contained many TISMs, the required resources would be modest because there is no need to copy all the TISMs from the unmoderated target; we would need to store only the meta information. As we describe elsewhere, much more elaborate schemes are also possible.¹

Users will generally have many targets from which to receive TISMs, including family members, bosses, company mailing lists, hobby clubs, government agencies, and so on. A user can ignore

any such messages independently without further ado. As mentioned, the sender rather than the receiver is initially responsible for resources, thus avoiding many of the problems related to unsolicited messaging.

Future Work

We are currently separating our system between a messaging-independent *micro-object layer*, a messaging middleware layer on top of that, and a graphical user interface. (The latest source code is available for download from www.cs.vu.nl/ums/.) This separation will let us investigate replication strategies in both formal and practical settings. The lightweight micro-object layer will let us run a large-scale emulation on our wide-area cluster of workstations. Finally, the middleware layer implementation will be connected to Internet email and NetNews, and possibly to existing instant messaging systems.

Others might also use the micro-object layer as a testbed for different replication strategies. The unified messaging middleware interface should prove to be useful for testing those replication strategies.

More work is needed on the topic of immutable objects – not just in the context of middleware for user-message distribution, but also in the context of middleware layers for distributed file systems, distributed directory services, distributed databases, and so on. □

References



1. J.M.S. Wams and M. van Steen, "Pervasive Messaging," *Proc. 1st Int'l Conf. Pervasive Computing and Comm.* (PerCom), IEEE CS Press, 2003, pp. 495–504.
2. J.M.S. Wams and M. van Steen, "A Flexible Middleware Layer for User-to-User Messaging," *Proc. 14th Int'l Conf. Distributed Applications and Interoperable Systems*, LNCS 2893, Springer-Verlag, 2003, pp. 297–309.
3. M. van Steen, P. Homburg, and A. Tanenbaum, "Globe: A Wide-Area Distributed System," *IEEE Concurrency*, vol. 7, no. 1, 1999, pp. 70–78.
4. G. Pierre, M. van Steen, and A. Tanenbaum, "Dynamically Selecting Optimal Distribution Strategies for Web Documents," *IEEE Trans. Computers*, vol. 51, no. 6, 2002, pp. 637–651.
5. P. McFedries. "Blah, Blah, Blog," *IEEE Spectrum*, vol. 40, no. 12, 2003, p. 60.

Jan-Mark S. Wams is a PhD student at the Vrije Universiteit, Amsterdam. His research interests include efficient large-scale replication and domain-specific compression. Wams is the inventor of "lazy evaluation" compression. He received a master's degree in computer science from the Vrije Universiteit. He is a member of the IEEE and the ACM. Contact him at jms@cs.vu.nl

Maarten van Steen is professor at the Vrije Universiteit, Amsterdam. His research concentrates on large-scale distributed systems. He co-authored the text book *Distributed Systems* (Prentice Hall, 2002). He is a member of the IEEE and the ACM. Contact him at steen@cs.vu.nl.

IEEE Transactions on Mobile Computing

A revolutionary new quarterly journal that seeks out and delivers the very best peer-reviewed research results on mobility of users, systems, data, computing information organization and access, services, management, and applications. *IEEE Transactions on Mobile Computing* gives you remarkable breadth and depth of coverage ...

Architectures

Support Services


Algorithm/Protocol Design and Analysis

Mobile Environment

Mobile Communication Systems

Applications

Emerging Technologies



To subscribe:
<http://computer.org/tmc>
 or call
 USA and CANADA:
+1 800 678 4333
 WORLDWIDE:
+1 732 981 0060