Unifying User-to-User Messaging

J.M.S. Wams and M. van Steen • *Vrije Universiteit Amsterdam* • {*jms* |*steen*}@*cs.vu.nl*

Abstract

Unifying user-to-user messaging systems is becoming popular as it allows message exchange independent of time, place, protocol, and end-user device. Building gateways to interconnect existing messaging systems seems an obvious approach to unification. We argue that unification should take place at the level of the underlying messaging systems. Such a unification results in one system delivering the same messaging services that all currently existing messaging systems deliver, as well as different combinations of those services that cannot be currently supported. We present a novel unifying messaging system. Our approach supports services such as offered by e-mail, fax, short messaging, instant messaging, netnews, and voicemail. To substantiate the claim that such a unified model can be implemented efficiently on a worldwide scale, we present the design of an accompanying middleware system.

User-to-user messaging platforms are not well connected. Unifying these platforms will provide huge benefits for users.

There are many user-to-user messaging systems of which Internet e-mail is probably the most famous.. User-to-user messaging platforms are omnipresent with many different devices to reach someone, ranging from i-mode phones to answering machines. However, when considering interconnectivity between different platforms, one can only conclude that it is rather limited. What we see today is that users have adapted to this lack of interconnectivity.

Each time a person wants to send a message, she will select an appropriate messaging system, taking into account what systems are available at that time to both sender and receiver, as well as the location of both parties. Unfortunately, misjudgments may easily lead to delayed or even undelivered messages, even if the recipient could have been reached instantaneously. For an important message, a user might even decide to use several messaging systems in parallel to increase the odds of successful delivery.

This is not at all a satisfactory situation. A user should not be burdened with having to select a messaging system, but should preferably be offered a single and simple approach to sending a message. Unification is the keyword here. Unification of user-to-user messaging has two parts: unification of data transport and unification of data representation. Much research has been done on unification of data representation, as exemplified by several platform-independent standards like ASCII, MIME, and XML. In this paper, we focus on the unification of data transport, a vast and chiefly unexplored research area.

Related Work

Unification of user-to-user messaging is also referred to as "unified messaging." A simplified form of unification is the ability to have a single mailbox for receiving fax, voicemail, e-mail, and so on [7]. Since most user-to-user messaging is either Internet based or telephony based, unification is also sometimes narrowly defined as Internet-based integration of telephony and data services [6]. In line with these approaches are the commercial offerings of unified messaging, providing a centralized place where a user can collect all her messages, although usually limited to e-mail, voicemail, faxes, and phone-text messages.

Current approaches unify messaging by defining, for each user, a single message box to collect messages from different services. Note that in this approach, unification takes place at the receiver's side. Our approach is different in that we propose unification at the system level, that is, at the level of message transportation, rather than unifying the messages at a fixed end point. In fact, in our system there is no collecting end point. Instead, messages are kept at their starting point and are made available to the recipient through replication. In other words, messages are stored at the sender's side from where they can be copied to the receiver's side.

Our approach solves an important problem inherent to end-point unification, where it is only after reception that a user can find out who the sender is. In our system, a sender targets not the recipient, but a message box provided by the recipient. The recipient can create many message boxes and she can give different access rights to different people and groups, thus having fine-grained control concerning whom she wishes to receive messages from.

In short we strive, not merely to connect, but to unify most existing user-to-user messaging systems into one system. In our opinion, this necessitates a multi-platform user-to-user messaging middleware layer, which none of the existing proposals to unifi ed messaging offer.

Unification through Middleware

Our proposal for a middleware solution is partly based on the fact that pairwise integration of n different platforms will take $O(n^2)$ effort. In contrast, unifi cation by means of a common message transport system (i.e., middleware) will reduce these efforts to $O(n)$. Furthermore, by offering a single user-to-user messaging system, it becomes much easier to support combinations of messaging services that have not been realized so far. For example, with middleware it becomes possible to change an e-mail-like thread into a netnews-like group.

When designing a unifying messaging system, it is important that the concepts underlying existing messaging services can be captured in a single messaging model. To this end, we have developed a taxonomy along four different dimensions, characterizing various capabilities of existing messaging transport systems, of which the details are described in [4]:

Dimension	**Issue**
Time	How long the system stores a message
Direction	Is message transport unidirectional or bidirectional
Audience	Who are potential recipients
Address	Is there support for unicast, multicast, or broadcast

As an example, Internet e-mail never removes messages, it provides only unidirectional communication, it allows a user to send a message to anybody in the world (with an e-mail address), and it supports multicasting (i.e., it can handle multiple recipients). As another example, netnews removes messages after they expire, a single communication channel is used to exchange news items (i.e., newsgroups are duplex), only a subscriber to a specifi c news group is a potential receiver, while messages are broadcast to the recipients.

This taxonomy sets an important requirement for unifi ed messaging: every useful *(time,direction,audience,address)* tuple should be permitted, effectively meaning that any type of messaging service that can be classifi ed by our taxonomy should be supported.

In addition to this requirement (more formally referred to as *maximum adaptability* [5]) we set out to build a unified messaging middleware layer that could scale in terms of the number of messages that it needs to process, the number of users, and the dispersion of users and resources across the Internet (and other networks).

Also, as we already mentioned, in contrast to existing approaches, our system places the receiver in control. This choice comes from a simple, but important other requirement, namely that a unified messaging system should hinder unsolicited messages, lest it would become unusable. On the other hand, we feel that other things are best handled in an end-to-end fashion. Therefore we decided to accommodate nor hinder presence, secrecy, integrity, authentication, and non-repudiation.

Architecture

Our unified messaging system is based on two kinds of messaging objects. The first type of object is a **target**. A target is a generic name for entities such as mailboxes, news groups, message channels, chatrooms, etc. It can best be thought of as storage place for messages. As we shall explain below, a single user can have many targets.

The second type of object is a **TISM**, which stands for a **targeted immutable short message**. TISMs are messages that are directed to targets—not users. Furthermore, once a TISM has been passed over to our messaging system, it can no longer be modified. In other words, TISMs are immutable by design: what has been sent can never be modified again. Finally, TISMs are short messages, reflecting what is common in current user-to-user messaging systems.

Both type of objects have an associated expiration time. Until that point in time, every object has a fixed **home**, which will keep a copy.

What sets our messaging objects apart from the common notion of objects is that each can have its own associated **replication strategy**. In other words, our messaging objects do not only encapsulate data and operations, but also a strategy that dictates how that object is to be migrated, distributed, and replicated in the messaging system.

This design choice is inherited from our previous work on Globe [3]. By allowing replication at the object level, that is, avoiding the need for a systemwide replication policy, we obtain maximal flexibility needed to mimic the behavior of existing messaging services. Equally important is that this approach provides excellent means to achieve true scalability through differentiation of strategies, as we have demonstrated for Web documents [2].

Every target or TISM object is created by a **user agent** (**UA**). Message storage, transfer, and replication is handled by a collection of **target agents** (**TA**s). By its nature a UA can be sporadically online and offline, and in a sense is comparable to the role that a user agent has in e-mail systems.

To ensure that TISMs and targets can be accessed, we require high availability for TAs, practically meaning that a TA should be online. Note, however, that to compensate for possible low availability of a TA, a TISM or target object can have an associated replication strategy that increases its own availability. Our scheme resembles the Internet e-mail system somewhat in this respect, where an Internet service provider (ISP) runs a mail server and is responsible for its high availability. And also like the ISP model, every UA has one **primary TA** through which all communication flows. This TA will become the **home TA** of the objects created by the UA. Note that many UAs can share one primary TA.

A TA will act as a home TA only for a target or TISM that has been authenticated by a UA with the proper rights. The UA also handles end-to-end encryption. This enhances the overall security of the system, but equally important, it relieves the TA from access control. The TA's main role is to take care of replication. If another TA or UA requests a copy of a target or TISM object, the TA will simply provide one. This approach is safe, because due to the UA-to-UA encryption a copy of an object is generally useless without the proper decryption key.

Replication

To understand the importance of replication in our system, consider first the following simple, but workable scheme for message transportation. A TA could simply store each new object it receives from any known UA, and give out a copy of any object when requested. Furthermore, each TA only holds on to those objects for which it is the home. If a UA needs an object, it would then contact the home TA of that object to subsequently retrieve a copy. Note that this simple scheme produces a "working" messaging system, although it is highly susceptible to TA failures, and arguably does not scale very well as it can, for example, easily suffer from high latencies. This scheme can easily be improved by adding caching capabilities to a TA. When a UA requests an object, it can simply contact its primary TA and request it to retrieve the object. The primary TA can do a cache lookup, and if the object is not found, request the object's home TA for a copy. Upon reception, the primary TA forwards the object to the UA that requested it. If the object was cached by the primary TA, a new request (e.g., by another UA) could be serviced from the cache. Note that this scheme silently assumes that a UA has a relative effi cient connection with its primary TA. This assumption is not unreasonable, because a primary TA will usually be provided by an ISP or be available on the LAN of an organization. Replication can also lead to pre-emptive sending of objects. Most notably targets have a constant need to be updated. We will elaborate on target replication below. By piggybacking replication data on objects, it is clear that advanced retrieval schemes can be devised that further enhance system performance and robustness.

There are at least three ways to implement per-object replication. First, an object could be allowed to run arbitrary code written in Java or some other language. This would make the target and TISM into a full-fledged agent. There are many security issues involved, in addition to making the objects relatively heavyweight as code would need to be transported along with the data. To circumvent the problems of carrying code along with the objects, a second alternative would be to design a compact special-purpose replication language. This language would be used for expressing replication strategies. Third, we could simply program a number of predefi ned replication routines in each TA and have the object carry a replication strategy index number, indicating the replication strategy to be executed. Note that this is a flexible system because an object can also carry arbitrary replication data under full control of the object's replication strategy. Since one of our goals in designing the system was simplicity, we opted for the last alternative.

Our design choices provide much freedom in the selection of replication strategies. There are two reasons for this. First, since each object has a home TA, which is responsible for its availability, there is no need for hard guarantees in the case of, for example, a cache miss. When an object cannot be found locally, a TA can always contact the object's home to retrieve a copy. Second, because TISMs are immutable, we avoid many problems related to updating replicated data. For example, there are no write-write or read-write conflicts to resolve by which we avoid various types of consistency problems and accompanying synchronization issues. Targets are mutable, but of a special kind. In particular, they allow TISMs to only be added and in this sense behave similarly to append-only logs in fi le systems, although the order in which TISMs are added is not important for targets. Furthermore, targets (as well as TISMs) have an expiration time, after which they are automatically removed. These two properties make it much easier to handle replication.

The immutable and add-only characteristic also simplifi es the TA's cache strategy, because there is no longer a notion of stale data. The availability of an object through its home TA allows for any form of cache pruning. In other words, there is no need for a TA to ensure that at least one copy remains available in the system, as this is taken care of by the object's home TA.

TA and UA Design Details

These choices lead to a relatively straightforward structure of the TA (see Figure 1). The TA runs a number of threads. There are fetch threads to collect objects from

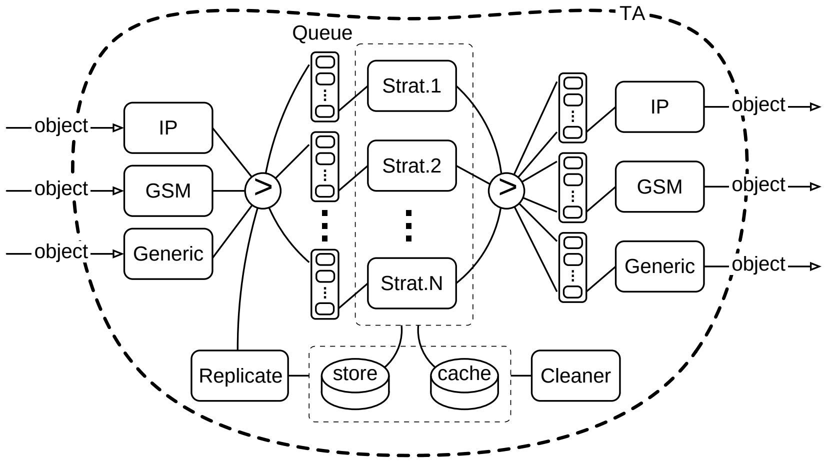

Figure 1: Internal design of a target agent.

the networks, and one or more threads per replication strategy. Each replication thread has a queue from where it takes objects to be processed. The fetch threads put the objects in the proper queue. For output there are also queues and output threads.

Furthermore, one or more replication threads look at objects in the storage and cache and puts them on the queue of a replication thread if need be. In this way, objects are guaranteed to be processed for replication. Finally, there is a cleaning thread to delete expired objects and to clean the cache. Replication strategy threads can store or cache objects, and send out objects by putting them at either end of the output queue. Also, to allow for instant message exchange and presence notifi cation, the TA needs a notion to handle UAs going online on a target, that is, it may need to keep track of which UAs are currently communicating through the same target.

Target and TISM Objects

The target and the TISM objects are similar. They contain a systemwide unique ID, a replication strategy index, a creation and expiry date, a short text, a payload, and replication data. The unique ID is a composition of the object's home address and a locally unique index. A target or a TISM is represented by a set of distributed *local objects*, which minimally carry the object's unique ID and its two dates. Given the unique ID, any other part of an object can be retrieved from the home.

The payload contains the actual message in case of a TISM object, the short text contains the subject line. A target can have a subject line too, this will help users organize their targets. The payload of a target is basically a list of TISMs.

Targets can become arbitrary large. Even with a minimal representation of the TISM objects, it may become unpractical to exchange target objects. Since targets are mutable objects, the system will have to compare and update these objects. We therefore designed a scheme that allows us to cut down the memory footprint of the target object. A target's payload is a list of TISMs ordered by creation date, as shown in Figure 2(a). We decided on this ordering, because it makes the "old" part of the list more predictable and easier to compress. This compression is needed for performance reasons. Imagine a real-time messaging system sending a list of 1 million TISMs over the network every second for comparison with some other list to see if there is something new.

For effi cient comparison and updating of lists, we introduce a **content list**. A content list is a list of descending dates. Between each two dates either a list of TISMs or a hash of a list of TISMs and the number of TISMs in the hash, as seen in Figure 2(b)–(e). A content list of a particular TISM list can range in size from a few bytes to just over the size of that TISM list.

Let us consider an example where TA-A and TA-B both have local instances of the same target T. Let us assume that the replication strategy that belongs to T prescribes that the lists should be unifi ed. Assuming that TA-A has some TISMs that TA-B does not, the update proceeds roughly as follows.

First, TA-A sends a minimal content list to TA-B [Figure 2(b)]. The executed replication code of TA-B 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 taken. However, as there is a difference, the executed replication code of TA-B calculates

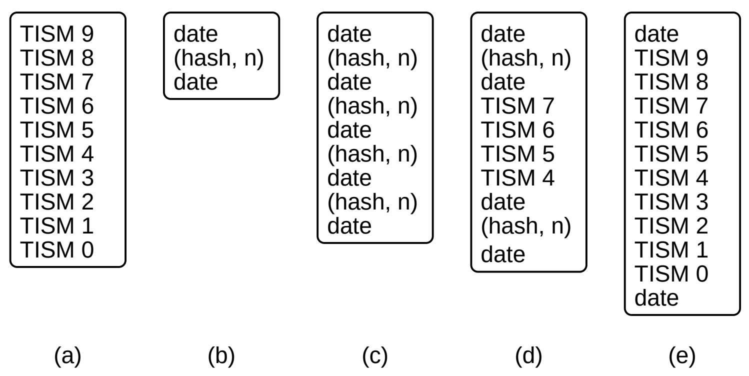

Figure 2: An ordered list of TISMs in five different formats, (a) plain TISM list, (b) minimal content list, (c) hash-only content list, (d) mixed content list (e) TISM-only content list

a number of hashes of partitions of its TISM list, and sends them in a content list to TA-A [Figure 2(c)]. Upon reception, TA-A executes the replication code that calculates the hashes on its TISM list and creates a new content list where hashes that differ are replaced with the actual TISM sublist, as shown in Figure 2(d). This expanded content list is sent to TA-B. Upon reception, TA-B checks the hashes, finds no differences and merges the TISMs into its own list. TA-B does not send any reply to TA-A. TA-B now has updated its TISM list. Note that TA-A and TA-B do not keep track of a session, but react to each request separately as in connectionless services.

Security

As mentioned before, encryption is handled by UAs. Associated with each target, there is one *(post,read)* key pair. Every TISM belonging to the target is encrypted with the post-key. (Actually, the TISM is encrypted with a random symmetric key, which is encrypted with the post-key.) In order to decrypt the TISM, the matching read-key is needed.

By controlling the distribution of keys for reading and posting TISMs, the user controls access to targets she created. Since the UA can create objects at will, users can replace objects in case of an infringement.

Keys are distributed out-of-band, just like e-mail and IM addresses. The messaging system itself will be most likely a primary vehicle for key distribution. For example, imagine a user that puts a post-key and a target ID in some public place (e.g., a Web page). Another user uses them to post a message in the target, containing some newly created target with its own (post,read) key pair. The users can now communicate using this "dedicated" target. No other user could, under normal circumstances, join the discussion. An e-mail address on a Web page gives away much more control.

To prevent malicious and malfunctioning servers from filling a target with nonsense messages, a second (private,public) key is associated with each target. The private key is distributed along with the post-key, and is used to sign the encrypted TISMs. The public key is kept by the home TA of the target and can be used to verify the signature, and to filter out Denial-of-Service (DoS) data.

Message Flow

To illustrate how messages flow through our system, consider the following scenario. Assume a user A has just given a subject and message text to UA-A. UA-A has the post key and ID for a target T and proceeds as follows. It builds a TISM object with the encrypted subject as short text and the encrypted message text as long text. UA-A then specifies a "hold this" strategy, effectively requesting the TA to be the home TA for this TISM, and adds a signed secure hash as replication data as well as a second replication strategy. This TISM is sent to A's primary TA, TA-A.

Upon reception, TA-A checks the signed hash. It fills in the creation date field and generates a unique ID for the TISM. It then replaces the message's "hold this" strategy with the second strategy from the replication data, and puts the resulting object in the store and back on the input queue for further replication. TA-A then creates a copy, signs it and sends it back to UA-A. UA-A now knows that its primary TA will act as a home TA for the TISM.

To finally post the message into the target T, UA-A also creates a local target object based on the ID of target T and an "update list" strategy. It then creates a simple content list consisting of twice the creation date with the TISM's unique ID in between. This target object is sent to TA-A.

On its turn, TA-A will send this target object on to other TAs, as dictated by the replication strategy of the target. This replication strategy is known to the TA, because it most likely holds a more complete copy of that target in its store or cache. If not, the TA will have to retrieve the replication strategy first. At some point in time, another UA, say, UA-B, makes a request for target T from its primary TA, TA-B. If user B so wishes, the encrypted short text or message text will be retrieved via TA-B. Now, if UA-B has the read key for target T, the message can be displayed to user B.

Some details have been glossed over, particularly the replication from TA to TA, but this replication is dependent on the replication strategy chosen by the creator of target T.

Mimicking Existing Systems

Our messaging system can mimic existing systems like e-mail and instant messaging (IM). To mimic e-mail for example, the UA could distribute a post-key of a newly created target to the general public, effectively creating a public mailbox that everybody can post into.

To mimic an IM system, the UA would distribute the post and read key of some newly created target, creating a chat room like target. Users could go “online” in that target, by setting the strategy of this target to an “instant forward replication strategy,” and requesting that target from their primary TA.

The UMS can be used to mimic other existing messaging systems such as netnews and Web logging [1], but the model has much more to offer. Proper distribution of keys allows our messaging system to implement almost any type of messaging. Some might find it surprising that with such a simple architecture, some novel and complex forms of messaging can be facilitated. Let us consider a few examples.

If there would be one target for which all users had a post and read key, it would be a form of say-all, hear-all (SAHA) messaging. It would be hard to deny access to individual users. Note that in our model, it is feasible to have a high-volume target like this without a huge central server, because TISMs are stored at the poster’s home TA.

Moderation could be added as follows: one user, called moderator, creates a new target and posts only 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 by sharing the post key for the new target with a selected number of people would allow a small group to do the moderation. The beauty of this scheme is that any user can start its own moderated target based on the SAHA target. It is worth pointing out that even if the moderated target would contain many TISMs, the required resources for the moderator would be modest for there is no need to copy all the TISMs from the unmoderated target, only the meta information would have to be stored. Much more elaborate schemes are also possible, as described in [4].

Note that users will generally have many targets to receive TISMs from: their spouse, their boss, their mother, their company's mailing list, their hobby club, their government, and so on, each of which can be ignored independently without further ado. As mentioned above, the poster is initially responsible for resources, not the receiver.

Conclusions

Our approach towards unified messaging shows that much can be gained in comparison to existing systems. In particular, by unifying at the systems level, we allow for various advanced models including models that correspond to current messaging systems. By putting the receiver in control and effectively letting the sender pay for messages, we avoid many of the problems related to unsolicited messaging.

We are currently in the process of completing a first implementation of our system (which is available through our Web pages). Future research concentrates on devising specific replication strategies, and providing a user-friendly interface for the UA.

References

[1] P. McFedries. "Blah, Blah, Blog." *IEEE Spectrum*, 40(12):60, Dec. 2003.

[2] G. Pierre, M. van Steen, and A. Tanenbaum. "Dynamically Selecting Optimal Distribution Strategies for Web Documents." *IEEE Transactions on Computers*, 51(6):637–651, June 2002.

[3] M. van Steen, P. Homburg, and A. Tanenbaum. "Globe: A Wide-Area Distributed System." *IEEE Concurrency*, 7(1):70–78, Jan. 1999.

[4] J.-M. Wams and M. van Steen. "Pervasive Messaging." In *Proc. 1st International Conference Pervasive Computing and Communications (PerCom)*, Mar. 2003. IEEE Computer Society Press, Los Alamitos, CA.

[5] J. Wams and M. van Steen. "A Flexible Middleware Layer for User-to-User Messaging." In *Proc. 14th International Conference on Distributed Applications and Interoperable Systems*, Lecture Notes on Computer Science, Nov. 2003. Springer-Verlag, Berlin.

[6] H. J. Wang, B. Raman, C. Chuah, R. Biswas, R. Gummadi, B. Hohlt, X. Hong, E. Kiciman, Z. Mao, J. S. Shih, L. Subramanian, B. Y. Zhao, A. D. Joseph, and R. H. Katz. "ICEBERG: An Internet-core Network Architecture for Integrated Communications." *IEEE Personal Communications*, 7(4):10–19, Aug. 2000.

[7] C. K. Yeo, S. C. Hui, I. Y. Soon, and G. Manik. "Unified Messaging: A System for the Internet." *International Journal on Computers, Internet, and Management*, 10(3), Sept. 2000.