Scaling and Selectivity: From NSTP to SGAP

Mark Day

Lotus Development Corporation
55 Cambridge Parkway
Cambridge, MA 02142, USA
Mark_Day@lotus.com

Abstract

We previously designed the Notification Service Transfer Protocol (NSTP) as a general-purpose coordination mechanism for building synchronous groupware applications. We initially thought that a colleague awareness tool (colloquially, a "buddy list") was just another synchronous groupware application. However, as users requested more capabilities, we discovered that awareness applications have an underlying model of interaction that differs from our other, quasi-meeting, applications. In particular, we ran into problems of scaling (NSTP's model required too much unnecessary information to be distributed) and selectivity (NSTP's model did not allow fine enough control of the way information was exposed). Accordingly, we designed the Simple General Awareness Protocol (SGAP) to address the problems that arise in sophisticated awareness applications. The problem is broader than NSTP alone: the international standard T.120 suffers from the same limitations, as does any other infrastructure built around a quasi-meeting model.

1. Introduction

We are interested in composing a particular kind of distributed application: namely, those in which the resulting application gives two or more people the impression that they are working together at the “same” time. Such systems are generally called *synchronous collaborative applications* or *synchronous groupware*. Although such applications can be constructed from scratch, we are also interested in two ways that such a system can be created by modifying previous systems: either by incorporating synchronous collaboration into an existing distributed application supporting asynchronous collaboration, or by enabling the synchronous multi-party use of an existing single-user (non-distributed) applications.

There is a considerable body of existing work and technology in this area, including a number of commercial products based on the ITU T.120 data conferencing standard [8]. However, all of these products suffer from a common weakness when applied to the problem of *colleague awareness*.

What is colleague awareness? In a network, people and systems may be available or unavailable, and may want to expose a variety of descriptive information or contact addresses (telephone number(s); mood; quote of the day; email address; and the like). We refer to this information as *presence information*. The basic problem of colleague awareness is to expose an entity's presence information to other interested entities. An early example of a system with this sort of functionality is Zephyr [6]; more recent commercial examples include systems such as *Instant Messenger* from America Online; *ICQ* from Mirabilis and *Ding!* from Activerse. A common colloquial term for these products is “buddy list;” however, that name is currently claimed as a trademark by a vendor, so we avoid its further use.

We have found in user studies and focus groups that colleague awareness as implemented by these products is only a simplified version of what people want. The more advanced form of colleague awareness includes a notion of *selective dissemination of information*. That is, the full colleague awareness problem is not to distribute one version of presence information to interested parties; rather, it is to distribute multiple versions of presence information appropriately to interested parties. Unfortunately, selective dissemination is at odds both with the structure of T.120 and with the underlying sharing models of the published protocols of colleague awareness tools of which we are aware.

This paper explains how we confronted this problem as we built our experimental colleague awareness tool, called Prairie Dog, and why the problem of selective dissemination led us to design a new protocol (SGAP) complementary to our existing protocol (NSTP). We believe that any colleague awareness tool whose designers did not anticipate selective dissemination will need to undertake a similar redesign of their protocol infrastructure.

2. Notification servers

We have worked [2,4,5,9] on supporting synchronous groupware. (For an overview of the general issues in groupware, see [1,7]). We proposed the use of *notification servers* that would rapidly communicate changes in small items of state to the clients participating in a collaboration.

The first protocol we designed for communicating with such a server was the Notification Service Transfer Protocol (NSTP). We intended that NSTP should be useful for the communication and coordination aspects of a wide collection of collaborative applications.

An NSTP server's universe consists of Places and Things. A Thing is a named byte array, with some associated immutable attributes governing access rights. The value of a Thing may be set to an arbitrary sequence of bytes. In general, the value of a Thing is entirely uninterpreted by the server. A single Thing may thus

- encapsulate the state of several user-visible entities, or
- be only part of the state of a singe user-visible entity, or
- be used for internal coordination among clients and have no relationship to any user-visible entity in the application.

In particular, clients can use arbitrarily complex transformations and encodings of their shared state (such as encryption or compression) without affecting the operation or utility of the server. Each Thing also has an associated latch that can be used to ensure mutual exclusion and which can be used to build more sophisticated concurrency control mechanisms.

A Place serves three distinct purposes:

1. It is a naming scope for Things. Each distinct Thing has a unique immutable name within its Place, but Things in different Places can have the same name.
2. It is the granularity at which clients can control the server's delivery of notifications. A client that is in a Place receives all notifications about changes to the Things in that Place, while a client that is outside a Place receives no notifications about those Things.
3. Places are part of the access control mechanism for Things. Some Things are visible (or even writable) for clients outside the Place, while others require that the client be inside the Place to see them or modify them.

Our protocol is built around this model of Places and Things and three additional principles:

1. *Notifications are symmetric:* the client causing the notification receives the notification just as every other client in the Place does.
2. *Responses are sufficient:* a client knows enough from a response to a request to continue correctly without waiting for a notification.
3. *Notifications are local:* no client outside a Place receives notifications from that Place.

Our experience suggests that this model is sufficiently powerful and flexible for a diverse set of applications. More importantly, perhaps, it is simple enough so that it is easy for application writers to see how to map their application into this framework, and to remap it in different ways to achieve different objectives.

NSTP has been broadly useful. We have built a variety of interesting applications using the protocol and servers, and applications using NSTP remain in daily use at Lotus. However, we have concluded that NSTP is not well-suited for colleague-awareness applications.

3. Limitations of NSTP

In simple systems for small groups, it suffices to broadcast all the presence information collected. However, as organizations grow in size, two new requirements arise: scaling and selectivity.

Scaling: As the number of people in a system grows, the amount of presence information to be distributed also grows. Roughly speaking, the distribution problem grows as n^2. However, we know that large organizations are not fully-connected: it is not the case that every individual communicates with every other individual. Indeed, even among parties that do communicate occasionally, there is not necessarily an identical need to have awareness of each other. We might reasonably guess that instead of growing as n^2, the actual awareness needed by an organization grows at a rate closer to linear ($n \log n$ might be a reasonable approximation). We would like to distribute only as much information as is really needed, rather than requiring an n^2 "virtual meeting" of the entire organization simply to distribute awareness information.

Selectivity: In a large organization, a person may choose to expose different presence information to different people or groups. The following are examples of the ways that people ask to control their visible presence information:

- they want to be available to co-workers in the same organizational unit but unavailable to those in other organizational units.
- when working on an *ad hoc* task force with a tight deadline, they want to be unavailable to usual co-workers while being available to the members of the task force.
- they want to provide different levels of detail in the information available to different groups. When travelling, current team members should get telephone contact information; other colleagues should get at least a return date; persons not otherwise identified should know only that the traveller is unavailable.

NSTP's access control mechanisms allow only for Things to be accessible or inaccessible: there are no mechanisms that would allow different clients to see different values of the same Thing. Indeed, NSTP is based on a different notion of synchronous applications, in which the goal is to provide all the clients a common view of shared data.

We call NSTP's approach a *quasi-meeting model*: the systems built around it are like physical meetings in that participants (mostly) see the same things.

The requirements for scaling and selectivity lead to the design of our new protocol, the Simple General Awareness Protocol[3]. It provides selective dissemination of information to different parties, including the ability for two parties to see different values of the same property.

4. Selectivity: The SGAP matrix model

The central concept of SGAP is a 2-dimensional matrix implemented by the server, where one dimension (say the rows) represents *items* and the other dimension (say the columns) represents *viewers*. Items are data elements to be shared, while viewers represent clients that can be sharing the data. The intersection of an item and a viewer (called a *cell*) represents the information about the item to be exposed to that viewer. One special viewer column (the "default viewer") represents all otherwise-unrepresented viewers.

The matrix is the most important difference between SGAP and a protocol like NSTP. The matrix allows the selective dissemination of information that is impossible in NSTP. Indeed, SGAP restricted to the single default-viewer column has semantics that are roughly comparable to those of NSTP.

Conceptually, a client may act on behalf of any number of items and/or viewers. (We explain later how and why this is somewhat restricted in the actual protocol). A client acting on behalf of an item (an *item controller*) supplies the data that is then made available to other clients that request the item. The item controller also configures which viewers are considered to be default viewers (so that they get the data in the item's "default viewer" cell). A client acting on behalf of a viewer (a *viewing client*) may request the data of an item, and will receive the contents of the appropriate cell as determined by the server. In making this determination, the server uses the information previously supplied by item controller(s) of the item.

Viewing clients cannot name cells directly: they receive the contents of a cell only by requesting a particular item as seen by a particular viewer. Thus, a row of the server actually represents all of the different versions of the item at an instant in time. A crucial aspect of this design is that it is not possible for a viewing client to tell whether a change in the information it views is caused by a change in the underlying data or a change in access control. This feature means, for example, that a client receiving a notification of a person's status changing from "available" to "do not disturb" cannot determine whether that change applied to every client or only to it. For a single viewing client, rearranging access controls to provide more or less information is indistinguishable from changing the data in a single-column NSTP-like server.

5. Scaling: Enabling and disabling notifications

In NSTP, the mechanism for controlling delivery of notifications was effectively an extension of the access control mechanism. An NSTP Thing has attributes that determine whether the server provides notifications of all changes to the Thing (the norm), notifications only at creation and destruction of the Thing, or no notifications at all. This mechanism has useful virtues when trying to build a meeting-like application: it is easier to ensure that all clients are able to construct consistent versions of the shared state, because all clients in a single Place are getting the same notifications. However, in a presence distribution system there might be thousands or millions of changing Things, each corresponding to a different person, and each of interest to only a possibly-empty subset of all clients. Accordingly, SGAP includes the notion of enabling and disabling notifications of changes, on a per-viewer, per-item basis. When notifications are enabled, a notification is delivered only if the relevant cell changes, and contains only the information relevant to that cell. So, for example, if an item is changed for all the viewers except the one who has enabled notification , no notifications are sent out.

6. Dual-role declarations

As was noted earlier, the actual handling of names is slightly more complicated than simply saying that a client can behave as any number of item controllers and/or viewers. In SGAP, a name is *declared* by a client, and the normal meaning of this declaration is that the client intends to be *both* the controller of the item with that name *and* the viewer with that name. Earlier versions of the protocol required the separate manipulation of the item namespace and the viewer namespace, which was inefficient and error-prone (since a client could successfully create its state under an item name only to discover that another client had already taken that viewer name). The linking of item and viewer name makes sense in a colleague awareness tool when we consider the integration of interpersonal communication and instant messaging arise: a client noticing something interesting about item X might like to make a response or comment available to the client controlling item X. The simplest way to make that capability available is to know that the controller of item X views the state of the system as the viewer X. The server, too, is simplified by the knowledge that the “magic viewer” corresponding to the item controller is the viewer with the same name as the item. Overall, the linking of viewer names and item names in dual-role declarations means that one has to go to a certain amount of trouble to make the item “Mark Day” controlled by a different connection than the viewer “Mark Day”.

7. Related work

The ITU T.120 standards [8] constitute the international standard for dataconferencing, but include no mechanisms for how one finds people to include in a dataconference or how one might determine their availability. We spent some time exploring the idea of doing colleague-awareness functionality entirely within the T.120 family of standards and concluded that it was unworkable.

The problem is not with specific features of T.120 such as the size or structure of the roster; rather, it is with the quasi-meeting model of T.120 sessions. To a first approximation, in T.120 there is no way to share information without being in the same session, and there is no way to hide information from those in the same session as you. Accordingly, the T.120 way to build a corporate or global colleague-awareness service would be to have every colleague-awareness client join a single corporate or global T.120 conference. This structure is a good example of failing to scale appropriately: we would need to build a T.120 server structure that was capable of allowing every client to be aware of every other client, even if clients had no interest in the presence or absence of most of the others.

The protocol and servers might allow for this scale of implementation, but it would be wasteful and unnecessary. This huge T.120 conference would be useful only for buddy-list purposes: it would be of little use for actual conversations or sharing with other people once they were found, since everyone else in the larger community would also be able to see and join in what you did. It also seems likely that such a large single session would run into various architectural limits determined by T.120’s pervasive use of 16-bit integers for channel numbers, tokens, user IDs, and the like.

Finally, T.120 has no mechanisms that we could use to enable selective dissemination of information, so that different clients could see different versions of “the same” data.

8. Conclusion

Our experience with trying to build colleague awareness tools has been educational. While the notification server model is still applicable in that domain, our first example of a notification server (NSTP) is not. Accordingly, we have designed and implemented a new notification server using SGAP. As an agreeable side-effect, we have a clear distinction between the model (notification servers) and instances (NSTP and SGAP). Both NSTP and SGAP are designed to share small amounts of state among clients in a synchronous collaboration, and to provide notifications to clients about changes in that state; however, they differ in their access control models, and indeed in what might be called their “philosophy of sharing” -- NSTP is suitable for conferences in which the main goal is to share a single common collection of information, while SGAP is appropriate for the selective dissemination of information.

References

[1] R.M. Baecker (Ed) *Readings in Groupware and Computer-Supported Cooperative Work: Assisting*

Human-Human Collaboration. Morgan Kaufmann Publishers, San Mateo, CA 1993.

[2] M. Day. “What Synchronous Groupware Needs: Notification Services.” Sixth Workshop on Hot Topics in Operating Systems (HotOS-VI), 1997, 118-122.

[3] M. Day. “Simple General Awareness Protocol.” Lotus Technical Report 98-06, 1998.

[4] M. Day, J.F. Patterson, J. Kucan, W.M. Chee, and D. Mitchell. "Notification Service Transfer Protocol (NSTP) version 1.0." Lotus Workgroup Technologies Technical Report 96-08, 1996.

[5] M. Day, J.F. Patterson, and D. Mitchell. “The Notification Service Transfer Protocol (NSTP): Infrastructure for Synchronous Groupware.” *Computer Networks and ISDN Systems* 29 (1997) 905-915.

[6] C.A. DellaFera and M.W. Eichin. "The Zephyr Notification Service." *Proceedings of the USENIX Winter Conference*, Dallas TX: USENIX Association, 1988.

[7] I. Greif (Ed) *Computer-Supported Cooperative Work: A Book of Readings.* Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[8] International Telecommunication Union. *Data Protocols for Multimedia Conferencing*. ITU-T Recommendation T.120. International Telecommunication Union, Bern, Switzerland. 1995

[9] J.F. Patterson, M. Day, and J. Kucan. "Notification Servers for Synchronous Groupware." *ACM 1996 Conference on Computer Supported Cooperative Work*, pages 122-129. Also Lotus Technical Report 96-01, 1996.