Replication Management of Application Sharing for Multimedia Conferencing and Collaboration

H. Abdel-Wahab — P. Kabore — O. Kim — J.P. Favreau

Information Technology Laboratory
National Institute of Standards and Technology
100 Bureau Dr, Stop 8920
Gaithersburg, MD 20899-8920 USA
Email: {wahab,kabore,okim,favreau}@antd.nist.gov

ABSTRACT Multimedia desktop conferencing systems are gaining momentum and increasingly becoming a reality. In these systems, besides using audio, video and multi-user applications, users can share a vast number of single-user applications for collaborative work. In these systems, there are two basic architectures for sharing single-user applications, centralized and replicated.

In this paper, we present the important issues involved in designing and implementing application sharing systems based on the replicated architecture. We show what type of objects need to be replicated, the specification of these objects and the different approaches for implementing the replication processes. The concepts presented were used in implementing JCE, a multimedia desktop conference system for sharing Java applications.

RÉSUMÉ. Les environnements collaboratifs multimédia sont en train de gagner de plus en plus d’intérêt et deviennent une réalité. Dans ces systèmes, en plus de l’utilisation de la voix, la vidéo et les applications multi-utilisateurs, les utilisateurs peuvent collaborer en utilisant des applications prévues pour une seule personne. Il existe deux types d’architectures pour partager de telles applications: l’architecture centralisée et l’architecture repliquée.

Ce papier décrit la conception et l’implémentation d’un système de partage d’applications basé sur l’architecture repliquée. Nous montrons quels types d’objets doivent être repliqués, la spécification de ces objets et les différentes approches pour implémenter les processus de replication. Ces concepts ont été utilisés pour implémenter JCE, un système de conférence multimédia et de partage d’applications Java.

KEY WORDS : Application Sharing, Multimedia Desktop Conferencing, Reliable Multicasting, Java Media Framework and Collaboration.

MOTS-CLÉS : Partage d’applications, Conférence multimédia, Multicast fiable, JMF (Java Media Framework) et Collaboration.
1. Introduction

As the Internet has gained popularity over the past decade, the need for collaborative multimedia conferencing and application sharing systems has risen significantly. Application sharing allows participants to view and interact with the same application (e.g., spreadsheet) during their conference. These systems are beginning to play large roles in research, education (e.g., distance learning) as well as businesses. For example, JCE (Java Collaborative Environment) [ABD 96], developed by the National Institute of Standards and Technology (NIST) in collaboration with Old Dominion University, is a framework for shared interactive multimedia applications in multi-platform. We have developed Java-based collaboration mechanisms that provide solutions to overcome the platform-dependency problems for collaborative computing in heterogeneous systems. In JCE, mechanisms were developed to intercept, distribute and recreate the user events that allow Java applications to be shared transparently. All the Graphical User Interface (GUI) components defined in the standard java.awt package are extended in JCE to implement the mechanisms. It also contains functions such as network management, conference management, floor control management and object replication management.

In JCE we are using the newly released Java Media Framework (JMF) [JMF] for playing audio and video streams. JMF provide a unified, non-proprietary, platform-neutral APIs to support the integration of audio and video streams based on the RTP and RTCP protocols. However, we are using native C code to capture and send both the audio and video to the JMF players. This is just an interim solution to experiment with JMF players until such time when Sun Microsystems finalizes the capturing part of JMF.

This paper examines and addresses the important issues involved in designing and implementing the replicated architecture for sharing single user applications in multimedia collaborative systems. We have implemented and experimented with a prototype replication system as part of JCE [ABD 96].

Section 2 outlines the two basic approaches that are used in most application sharing systems. In Section 3, we discuss "what" objects needs to be replicated while in Section 4 we show "how" to efficiently replicate these objects. Section 5, introduces the concept of application profile which is the basis for actual replication management algorithm presented in Section 6. Section 7 is the conclusion.

2. Basic Approaches for Application Sharing

There are two basic architectures [STE 95] for sharing single-user applications in computer supported cooperative work environments [GRU 94] such as multimedia desktop conferencing [KIM 97] and interactive distance learning [MAL 97]:

Centralized Architecture (see Figure 1): In this architecture, only one copy of the application runs on any one of the participants’ machines. The output of the application is first intercepted and then multicast to all participants for dis-
play on their workstations. The input from the participants (usually one user at a time), using any of the many possible floor control policies [DOM 97] are forwarded to the application as if it were provided by one single user. The application is not aware of the identities of the participants who are interacting with it. The application naively thinks that it is being used by only one single user. Figure 1 shows two participants sharing one copy of an application. This centralized architecture is adopted in many application sharing systems such as XTV [ABD 91] for sharing X windows applications.

**Replicated Architecture** (see Figure 2): In this architecture, there are $N$ copies of the shared application, one copy is running at each of the $N$ participants’ machines. The input from any participant is first captured and then multicasted to all $N$ copies of the application. Similar to the centralized architecture, one participant at a time may be allowed to provide input to the application according to the desired floor control policy. Figure 2 illustrates this replicated architecture for two participants where the output of each copy of the application is displayed to only one participant. JCE [ABD 96] for sharing Java applications is based on this replicated architecture.

Note that in the centralized architecture, only the application output is multicasted to all participants, while in the replicated architecture, only the input is multicasted to all participants. In some cases, either one of the two architectures can be used, but in some other cases only one architecture can be used. For example, in sharing X appli-
lications, both the centralized and replicated architectures can be used because it is easy to intercept both the input to and the output from the X applications. For example, the original version of XTV [ABD 91] is implemented using the centralized architecture while a recent implementation of XTV uses the replicated architecture [TAL 96]. However, in sharing Java applications [ABD 96], we were only able to intercept the user input events, and therefore, the replicated architecture approach was our only possible choice.

In this paper, we are not concerned with the legal and economic issues of replication. For example, the application may be licensed for use in group-collaboration mode, but not for making permanent copies and use in a single-user mode. As a matter of fact, some may argue that this issue may be a determining factor in deciding between the central and the replicated architecture. It is quite normal to license an application to run only on one specific host and in such situations, the central architecture is the only legal way to share the application collaboratively. Also, in some other cases, the application runs only on a specific manufacturer’s machine (e.g., specialized high performance parallel computer), and in this case, the central architecture may be the only way to share the application.
3. Replicated Objects

In this section, we will discuss what type of objects need to be replicated and what to do with replicated and newly created objects at the end of a collaboration. The following is the list of the objects that typically needs replication at each participant’s site before the start of collaboration:

1. Shared application executable code.
2. Initial application input objects, such as bit maps, images, resource files, input files etc.
3. Operating system environment, e.g., shell and environment variables.

It is important for the correct operation of the shared application to ensure that each participant has the same version of the program code. In addition, to ensure smooth operation, it is advisable that each participant have the same initial objects and environment variable settings. It is possible, say, that each participant may customize his environment to suite his needs as long as it does not interfere with or hinder the correct sharing of the same application. For example, some participants may choose larger fonts or different colors. While different customization of the same application is possible under the replicated architecture, it is recommended that all participants have the same view, feel and look of the shared application. We should emphasize that under the centralized architecture, there is no possibility for any private customization, because there is only one copy of the application. However, under the replicated architecture, it is theoretically possible for each participant to have his own private customization of the shared application. This is allowed as long as a private customization of one of the participants does not conflict with other participants’ customization of the shared application.

The problem of replication is not only concerned with what to replicate before the start of the shared application, but what to do after the termination of the shared application. For example, should all copies of objects made due to the replication process be removed? Should the values of the environment variables be restored back to their original values? Moreover, what should be done with the newly created objects during the collaborative session? Should all copies of a newly created object, except one, be removed, or should we let each participant keep his own copy of the object? Obviously, there are many alternative solutions to these questions. However, there is best or optimal solution, the participants themselves should ultimately decide what is the best under each given condition.

4. The Replication Process

In the previous section, we discussed what are the objects that need to be replicated, and in this section, we will discuss how to achieve this replication process in an efficient and orderly manner. Toward this end, we assume that for each shared application, there should be a profile. The application profile contains all the necessary
information about the objects to be replicated before the start of the shared application (initial objects). In addition to the initial objects, some objects may be newly created during the use of the shared application, and in such cases, the profile may specify what to do with these objects.

This information given for each object (initial or new) includes the source and destination locations of each object. An object location is specified by the three attributes: the host where the object is located, the directory where the object is installed, and the object name. The host in the destination location is implicitly defined as the local host address of the conference participant, while the host in the source location gives the host address where the desired program version is stored. In addition to the source and destination information, the application profile should specify the desired retention status of each object. The retention of an object determines whether an object should be retained or purged at the end of collaboration. Some may argue that it should be kept in order to speed up the start of the application the next time the same application is used again for collaboration among the same group, assuming no change in the application profile. Others may argue that it should be purged, since the object was copied for a specific temporary purpose and the status before using the application should be preserved.

Finally, many applications need to access the values of certain shell and environment variables. In such cases, the application profile should contain the needed environment variables and their desired values. The replication system should ensure that all copies of the shared application have the same or consistent values of all needed variables before the start of the application.

In order to start a new shared application $A$, usually one of the participants is responsible for initiating $A$. In the following, let us denote this participant as $P_0$, while all others are denoted as $P_1, P_2, \ldots, P_n$. Also, let $A$ needs the set of objects $O_1, O_2, \ldots, O_n$. The following are two possible ways to ensure that all participants have the same copy of each object $O_i$ of the shared application $A$:

**One-by-One:** Each participant $P_i$ independently examines $A$’s profile and is individually responsible for obtaining and installing the latest version of each object $O_i$ of the shared application $A$ and setting the values of each required variable.

**One-for-All:** For this purpose, we assume that all participants join a special multicast group $G_A$ reserved for application $A$. One of the participants, e.g., the initiator $P_0$, is responsible for obtaining the objects and multicasting them to all participants who might need them. This may be achieved as follows:

1. $P_0$ may get the object and then multicast it over $G_A$. This would require that the object be transmitted twice, once to obtain it and once to multicast it.

2. Let us assume that for each object $O_i$ there is an object server $S(O_i)$. In such a case, $P_0$ may inform $S(O_i)$ to multicast the object $O_i$ directly over $G_A$. Thus the object is transmitted only once. Here we might introduce the concept of self-multicastable objects. An self-multicastable object is an object that implements a method which multicasts the content of the object to a specified
multicast group. If we adopt and implement such concept, then \( P_0 \) can remotely invokes this method for each object to be replicated.

5. Application Profile

In this section, we show the details involved in designing and using the application profile. First, Figure 3 show a sample of a possible user interface to conveniently specify the application objects and variables. The data entered through the interface are used to produce several lists of objects and variables. These lists are used as input during the actual replication process. The interface design should be based on the important principle of using default values to fill-in the information that are not provided explicitly by the user. In the following we describe the major components of a profile interface:

**Defaults:** This gives the default host, source (src) and destination (dst) directories. The default host is where the initial objects needed by the program reside and where the newly created objects should be copied. The default src directory is where one finds the objects and the default dst directory is where these objects should be copied. Example:

- **Host:** sand.ncsl.nist.gov
- **Src Directory:** /home/jason/JCE/whiteboard
- **Dst Directory:** $HOME/JCE/whiteboard

![Figure 3. Profile Specification Interface](image-url)
Shell variables like $HOME can be used in specifying directories.

**Initial Objects:** This is for the specification of the objects to be replicated to the conference participants. The item **Name:** is used to specify either a single object or a group of objects using the pattern matching character "*". For example: * means all files under the src directory and *.class means all the class files under the src directory. The item **Remove at the End:** is used to specify whether the objects will be retained or deleted after the session is terminated.

**New Objects:** This is to specify what to do with the newly created objects during a session. By default the new objects will stay where they are at each participant’s machine. However, the user may specify that an object be copied back to a specified Dst directory and/or be deleted from each participant’s directory.

The information collected from the user interface is used as a basis for managing the replicated objects as described next. Each shared application may have two profiles:

1. Default Profile: This profile is specified by the application programmer. Normally, it contains the program object code and few input files such as bitmap images.
2. User Profile: This is provided at run time by the application initiator to augment or redefine some of the default profile attributes.

The result of parsing both the default and user-provided profiles is three lists:

1. Environment List (Env-List): Contains a list of $N \cdot V$ where $N$ is an environment Name and $V$ is its value.
2. Initial Object List (Init-List): Contains a list of $N \cdot H \cdot S \cdot D \cdot R$ where:
   - $N$: the object name
   - $H$: the host where the object is stored
   - $S$: the source directory where the object is stored
   - $D$: the destination directory where the object should be copied
   - $R$: specify the retention of the object. The values of $R$ are: $n, e$ where $n$ means do nothing, $e$ means to delete the object. The assumed default value is $n$.
3. Final Object List (Fin-List): Contains a list of $N \cdot H \cdot S \cdot D \cdot R$ and the meaning of each symbol is the same as described above in the Initial object list.

**6. Replication Management Algorithm**

In this algorithm, we assume each participant is running a process called Replication Manager (RM). The first step performed by RM is to join a special replication managers multicast group $G_{RM}$ for exchanging messages among all RMs of the current session. To start a new shared application, one of the participants specifies the application profile and provides his own user-provided profile to his local RM. This
local RM is referred to as the application initiator (AI) for the new application. In this section, RM refers to both an RM and an AI unless explicitly excluded. The AI fetches the application default profile and parses it along with the user-provided profile to produce three lists: Env-list, Init-list and Fin-list. The following are the steps performed by the RMs before starting the application.

1. AI multicasts over $G_{RM}$ the Env-List to all RMs.
   Consequently, each RM sets the environment variables as specified in the Env-List.

2. For each object $I = <N, H, S, D, R>$ in the Init-List, AI do the following:
   - Get the official date-of-last-update $d$ of the object $H.S.N^1$.
   - Get the local date-of-last-update $u$ of the object $D.N^2$:
     - If the object does not exist or $u \neq d$, get the object from host $H^3$
   - Join $G_I$ a special multicast group for object $I$.
   - Multicast over $G_{RM}$ a message $M = <I, d>$:
     - Each RM (except AI) obtains the local date-of-last-update $u$ of object $I$
     - Each RM which finds out that either the object $I$ does not exist or $u \neq d$ will do the following:
       - Joins the object multicast group $G_I$ to receive object $I$.
       - Waits a random amount of time then multicasts over $G_I$ a request to get object $I$.
       - Any RM hearing such request, should cancels its own request to get $I$.
   - If AI hears any such requests, it multicasts over $G_I$ the object $I$.
   - Each RM who joined $G_I$ receives and installs the object $I$.

3. After handling all objects in the Init-List, the AI multicasts over $G_{RM}$ a message to start the shared application.

4. Upon the termination of the shared application AI performs the following tasks:
   - For each final object $F = <N, H, S, D, R>$ in the Fin-List copy $S.N$ to $H.D.N$:
     - Multicast over $G_{RM}$ a message to delete all objects whose $R$ value is $\epsilon$.
     - Caution: Do not delete $H.D.N$
   - Multicast over $G_{RM}$ a message to delete each initial object $I = <N, H, S, D, R>$ whose $R$ value is $\epsilon$.
     - Caution: Do not delete $H.S.N$.

Figure 4 shows an example of five RMs: $AI, RM_i, i = 1, 2, 3, 4$ and three multicast groups: $G_{RM}, G_{I_1}$ and $G_{I_2}$. Note that all RMs are members of the $G_{RM}$ group. Note also that $AI$ is a member of all object groups, while the other RMs are not necessarily members of all such object groups. For example, $RM_4$ is not a member of any such group, while $RM_1$ is member of only $G_{I_2}$.

---

1. This notation means object name $N$ located at source directory $S$ of host $H$.
2. Means object name $N$ located at destination directory $D$ of the local host.
3. This is our criteria for determining whether an object is updated. If an object is updated, a new fresh copy should be retrieved.
7. Conclusions

Single-user application sharing in multimedia collaborative systems is a very powerful and useful facility. Users can use the same familiar set of applications in group work, just as they have been accustomed to using it individually. Many recent application-sharing systems are based on a replicated architecture which requires each participant to run locally a copy of the shared application on his machine. In this paper, we have discussed the different issues and problems associated with this replicated architecture among of which are:

1. How to identify and specify for each shared application the required initial objects.
2. How to deal with the transferred and newly created objects after the termination of the shared tool.
3. How to efficiently ensure that all participants will have the same version of the specified initial objects.

As we have shown in the paper, our concept of application profile and interface is a key component in addressing both the first two questions; and the use of reliable multicasting is an important ingredient in addressing the third question.

The issues that were not addressed in this paper are related to the reliability and recoverability aspects of replication. For example, we should have a feedback mechanism to ensure that indeed all required objects are in-place at every participant before the application gets the signal to start. In addition, adequate protocols should be developed to deal with transient errors such as network and local processing overload.
Références


[TAL 96] TALLA R. Replicated XTV. Master’s Project, Department of Computer Science, Old Dominion University, June 1996.