Non-Interfering Software Distribution

Richard Schmidt
Lockheed Martin ATM
richard.b.schmidt@lmco.com

Terry Duffy
McDonnell Douglas
terry.duffy@lmco.com

Abstract

This paper describes software distribution in the Display System Replacement (DSR) project. Non-interfering software distribution refers to the capability of a distributed system to receive updates to its software while it is functional without compromising its mission. DSR is a major Air Traffic Control (ATC) system designed for the United States Federal Aviation Administration (FAA). Deployment has started and will continue to over 20 sites, becoming operational in 1998. A single DSR site encompases several hundred RISC workstations servers and clients interfacing to a local pair of redundent mainframes which supply radar and flight data. Updates to the workstation software will be incorporated without interfering with the real-time mission critical applications.

Although the implementation meets the specific requirements of DSR, the problem and solution are of a general nature that are applicable to other systems. The problem, simply stated, is how the components of a distributed system can receive a new release of software without interfering with the ongoing mission. This solution paces the distribution and stores the new software in a location independent from the currently operating software until the operator of the system is ready to use it. It may be necessary to restart the processor for the new software to be loaded, but the actual transmission of the data to the distributed processors can occur without degrading the function of the system.

System Requirements

The primary focus of the DSR system is to provide air traffic controllers with timely data, and the contract identifies numerous performance and availability metrics that must be met. Software distribution is performed by the Distribute Software function, and it must not interfere with these fundamental system objectives. Additionally, the Distribute Software subsystem has several system operator requirements levied upon it:

- It must certify the integrity of files after they have been transferred.
- It must verify that a release is complete and correctly installed or report that it has failed.
- It must inform the operator of progress as it is occurs.

In addition to these requirements there are derived requirements imposed by the system architecture:

- The distribution of a software release should not perturb the response time of other user commands and system functions.
- The Distribute Software server also processes unrelated user commands and must return responses to the clients within a specified time interval. This interval must be met even when a large volume of data is being distributed.

Connectionless transmission is used to distribute the release data in order to minimize LAN traffic. The buffer size chosen for this data is 3840 bytes, which conforms to the frame size for the OSI communications stack being used. When UDP/IP is used, three frames are transmitted for each buffer. Connectioned transmission is used for some of the control messages. A Message Services subsystem provides a 64 Kbyte message interface at the application layer. Messages larger than 64 Kbytes are broken down into 64 Kbyte messages by the Distribute Software subsystem.

System Software Releases

A system software release is a set of files necessary to support the ATC mission. There are several different types of releases, each with different characteristics and frequency of update:

- An Operating System (OS) upgrade consists of selected backup images, which are updates to the OS, rather than an entire copy. There may be prerequisite updates and the size could be as large as 76 Mbytes.
- A full software system release is a complete set of the developed software system and is approximately 85 Mbytes. This type of release occurs when either a large collection of changes has been bundled or a common component bound in the modules has changed.

Proceedings of the DASIA 97 *Conference on* 'Data Systems in Aerospace', *Sevilla, Spain, 26-29 May 1997* (SP-409, August 1997)

- A delta release is a subset of the full release. It occurs when a fully compatible subset of files have changed and the changes are independent and can be shipped as a delta to a release. An example is when the values of adaptable data unique to a site are changed. The implementation allows deltas to be built upon a full system release or other deltas.

Software Bill of Materials

Files to be shipped as part of a release are described in a special file within the release called a Software Bill of Materials (SBOM). An SBOM is a text file consisting of one or more stanzas for each file to be distributed. Two types of SBOMs are defined:

- Source SBOM - describes the file to be shipped as it exists in the development environment, consisting of source stanzas which have fields describing:
  - relative path name on the development system
  - source ownership
  - target file permissions
  - target file ownership
  - relative path name on the target system
- Distribution SBOM - describes the file as it will exist in the target environment including all of the fields in the Source SBOM and the target stanzas containing:
  - full path name on the target system
  - symbolic link so the file can be referenced by operational software
  - checksum.

The source SBOMs are configuration managed along with the source code through the entire development cycle, and are maintained by the software developer. These source SBOMs have an imbed structure that allows each development team flexibility in how it manages release information. During the software build process the source SBOM is transformed into a distribution SBOM by the build tools which walk the imbed structure until all leaf nodes have been traversed. This is done at build time because that is the earliest point the file checksums and the actual release name is known. The result is the distribution SBOM, which is a single file containing source and target stanzas for all files within the release. A text representation was selected to provide a universal (development language independent) format which is easy to parse by programs and easy to edit by support personnel.

These files can grow large, depending on the number of files in a software release, and parsing of the SBOM could be a performance issue on some systems. Two methods to improve performance could be used:

1. Parse off-line and provide an language dependent version of the SBOM.
2. Parse on-line and save the results in a checkpoint file for future use.

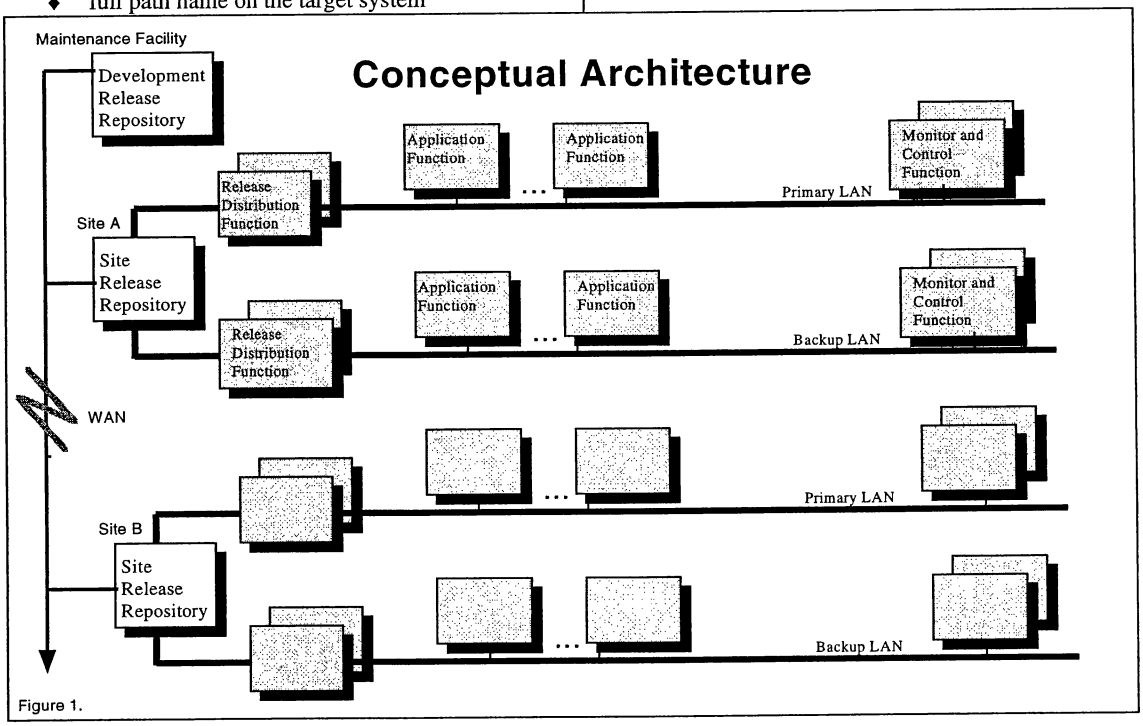

Figure 1.

System Environment

The environment of the system that relates to Software Distribution consists of the site, operational processors on two LANs for redundancy, a support processor connected to the operational distribution servers via a third LAN, and a technical center where Software Maintenance occurs. The releases are packaged and transmitted off-line via a WAN until they arrive at a site's support processor, as shown in the Conceptual Architecture (figure 1).

Commercially Available Software (CAS) is used to send the releases to sites because it is sufficient in a support environment. Currently available CAS is not appropriate to distribute the releases on the operational LANs because they would interfere with the real-time ATC mission. All releases that have been received by the support processors are said to be deployed. Support software is available to locally tailor the release for use at the site and make the release eligible for distribution over the operational LANs.

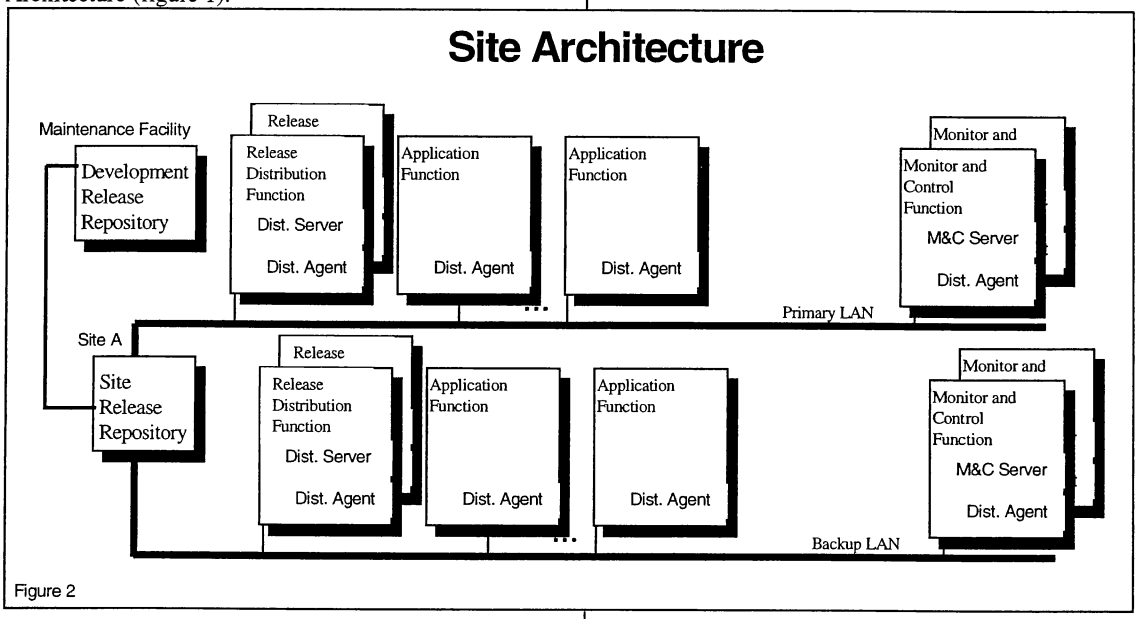

Figure 2

Network Architecture

Software Distribution is not dependent on a specific network architecture. For example, the distributed ATC system consists of hundreds of processors connected over two LANs (figure 2). Two independent subsystems are used because the system must be highly available and reliable. The primary subsystem uses IEEE 802.5 Token Ring with redundancy in the network. The backup subsystem uses IEEE 802.3 Ethernet 10Base2. Software distribution occurs independently on these networks and must occur on both since each processor is attached to only one LAN. Another architecture, such as FDDI, could be chosen without altering the Software Distribution subsystem.

Because the primary requirement of Distribute Software is to deliver the software updates without interfering with the ongoing ATC mission, a not to exceed data rate is selected for the software at build time. It currently is approximately 0.472 Mbps (59 Kbytes per second) for either LAN. With those values, Distribute Software's data rate will not exceed 3% of the Token Ring nor 5% of the Ethernet maximum data rate.

Software Components

The following operational software components are relevant to the distribution (figure 3):

- Monitor and Control Server -- processes user command to distribute software for a release to a set of processors and manages responses and displays.
- Distribute Software Server -- controls the long running distribution function and reports status.
- Distribute Software Agent -- prepares location to receive the new release, sets the OS characteristics of each file, and verifies its integrity.
- Broadcast Data Server -- Separates release files into blocks of data, broadcasts the blocks, and monitors their receipt on a per file basis.
- Broadcast Data Agent -- receives blocks and rebuilds them into files.
- Monitor and Control Agent -- registers the distributed release so that it is available for incorporation upon user request.

There is one active instance of each server per LAN, while each agent is active on every operational

processor. In general each server is replicated so that a backup is available should the primary server fail. The Monitor and Control Server is on a Monitor and Control (M&C) processor with a display and keyboard for the user to issue commands that control the system. The Distribute Software Server is on a Logical Interface Unit (LIUS) processor which has a dedicated connection to the support processor where the deployed releases are stored. The connection is referred to as a maintenance LAN and is used to remotely mount file systems from the support processor on the LIUS processor. Each deployed release is stored in a location on the support processor that can be remotely accessed by the Distribute Software Server.

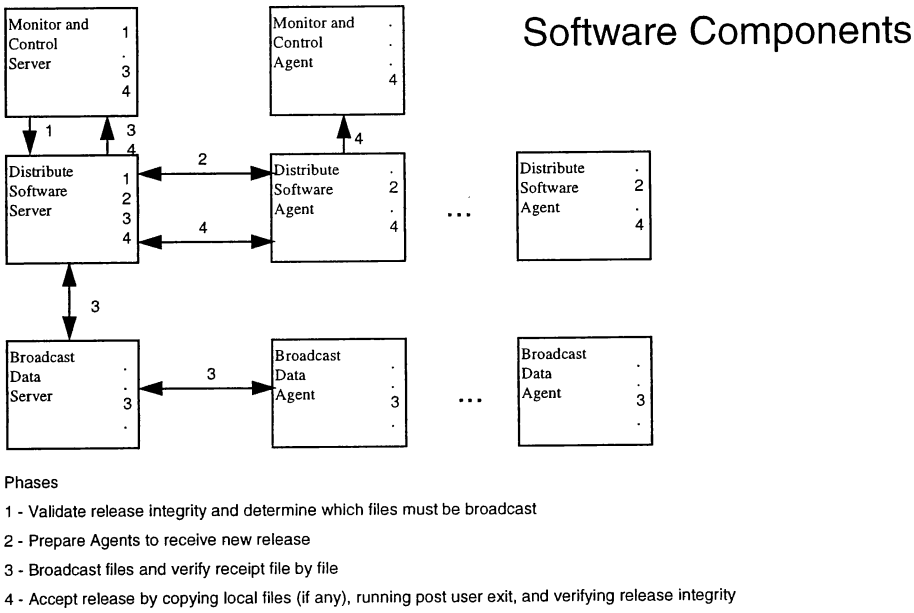

Figure 3

Phases of Distribution

There are four distinct phases of distribution:

- Phase 1 - Validate the release
- Phase 2 - Prepare each processor to receive the release
- Phase 3 - Broadcast the release files
- Phase 4 - Accept the release

Phase 1

Phase 1 begins when the Distribute Software Server receives a request from the Monitor and Control Server to start distributing a new release. The request specifies the new release, the set of target processors, and the current release for each target. The server validates that the releases exist in the list of deployed releases on the support processor. It parses the SBOMs for the releases and compares the file characteristics of the new release with those of the current release to establish commonality. This determines the minimal set of files that must be broadcast. Files that are identical between the two releases need not be broadcast because they are already on the target processors and may be copied locally during Phase 4. This is done to reduce network load. All relevant release information is stored in state data so that any SBOM is only parsed once per distribution. This is done to reduce the impact of the I/O on the operational system.

The backup Distribute Software Server is notified that a Distribute Software process has started. It is available, should the primary server fail, to send a response to the Monitor and Control Server indicating distribution has failed so the operator can assess the situation and start distribution again if desired.

With the number of files to be broadcast to the destination processors known, each file is verified by recalculating its 32 bit checksum and comparing it with the checksum specified in the SBOM which was retained in state data. Because this is CPU and I/O intensive, it must be paced. Each file verification is an event separated by a configurable value (for example, 2.5 seconds). When all files to be broadcast have been verified, Phase 1 is complete.

Phase 2

Phase 2 begins by broadcasting a message to the Distribute Software Agent on each processor to prepare a location to receive the new release. This message is a datagram that all agents on every operational processor receive. The message contains a bit map that signifies whether the agent should act on the request or ignore the message. This use of a bit map will occur in other

messages during distribution. It is a technique that allows any processor to be included.

If the message is intended for the agent, it creates and mounts a new file system to receive the release. Each processor may have three full releases of application software. Independent file systems are used so that one release may be easily deleted, without affecting another release. If successful, the agent is eligible to participate in Phase 3. If the release already exists, the distribution to that processor is considered to be complete (successful) and that processor receives no other messages that cause it to take further action. It is reported as having successfully received the release when all other processors are finished.

Each agent sends a response to the server, staggering the replies by separating them by a synchronized interval indexed by the processor's rank in the distribution list. This is done so that the replies do not arrive at the LIUS processor at the same relative time and overflow the server. This datagram is transmitted so that the adapter will filter it out at the other processors (ie., functional addressing is used).

The server sets a timer to monitor the arrival of a response from the agent on each target processor. A time-out or unsuccessful response indicates that the processor cannot receive the release. It is dropped from further consideration by modifying the state data that shows what is being distributed, and distribution continues. This "prune and continue" approach is fundamental to this solution. It was adopted because it is expected in a large distributed system that from time to time some processors will experience failures. There should be no single point of failure among the target processors that would prevent healthy processors from receiving software updates. A failure on the source processor, or one of its components (Distribute Software Server or Broadcast Data Server) will cause the entire command to fail. The command may be retried as soon as the backup servers assume the primary role, which happens in a few seconds. When all target distribution agents have responded (or a time-out was detected), the Distribute Software Server sends the list of files to be broadcast to the Export Broadcast server to begin Phase 3 and sets a timer to monitor responses. The Export Broadcast Server must transmit a health check to the Distribute Software Server every 60 seconds during the long running distribution. This marks the end of Phase 2.

Phase 3

Phase 3 begins with the Broadcast Data Server receiving a list of files to broadcast to a set of processors. Only one such request to broadcast a list of files is supported at any one time. The release information necessary to broadcast a complete release exceeds the 64 Kbyte buffer limit and requires multiple messages to be queued by the server. They will be completed in order and the Distribute Software Server will track all completions before proceeding to Phase 4.

There are three parts to Phase 3 and they occur on a file by file basis. Each file is subdivided into blocks.

1. The Broadcast Data Agents prepare to receive a file.
2. The Broadcast Data Server transmits the file and the agents receive it, block by block.
3. The Broadcast Data Server verifies that the agents receive the blocks in order.

Prepare for Broadcast

The Export Data Server sends a message to all Broadcast Data Agents with a bit map signifying which ones are target processors for the release. The agents will receive datagrams that contain blocks of the file if their bit is set; otherwise they will be unaffected by those messages. This technique is used to allow a retry should any processor fail to receive a file. The agents that successfully received the file do not receive it again, the datagrams are only delivered to those agents that need it.

Since this is a high availability system, it is entirely possible that should a soft failure occur during distribution, the agent software would be restarted before the retry limit was reached. It would then receive the file successfully and distribution would continue to all target processors for the next file. The server allows the initial attempt and up to three retries per file before it gives up on an agent and prunes it from any further action during the command. Although retries tend to slow down the distribution, we found it to be a valuable fault tolerance asset. A retry is necessary at the Application Layer if the lower layers of the OSI protocol stack did not detect and correct a missing frame. This could occur for example on Ethernet with a faulty repeater. The entire file is re-transmitted, rather than a missed frame. If frames are missed at the Application Layer frequently, the overhead of the file re-transmission can be reduced by switching to a frame-based retry. This involves more memory buffers to store frames as they arrive, and more frequent communication with the agents to detect a missing frame before the buffers are exhausted.

When all target processor agents have responded (or a time-out was detected and all retries exhausted), the first part of Phase 3 is complete.

Broadcast the File

The second part of Phase 3 begins when the Export Data Server opens the first file to be broadcast. Approximately 4 Kbytes of data are read into a block. Each block is assigned an incremented identifier and sent in a datagram to the agents. The transmission of each datagram by the server is paced to not exceed the configured LAN limit. If the server is also a destination

for the release, the block is copied locally into the new file system. The agents receive the blocks and monitor that they arrive in order. They are written to the new release file system. This continues until all blocks of the file have been read, transmitted, received, and written. The Distribute Software Server is notified periodically that the Broadcast Data Server is still healthy. The Monitor and Control Server is notified when another file has been transmitted so the number of files broadcast (n out of m) is displayed on the Monitor and Control processor. When all blocks in a file have been transmitted, the second part of Phase 3 is complete.

Verify the File

The third part of Phase 3 begins when the timer expires and a datagram is sent to the agents. If the bit map indicates the agent should take action, the number of blocks received for the file is verified against that in the message. If they are not equal, the response indicates the transmission error so the file can be retried. The third part of Phase 3 is complete when all target processor agents have responded (or a time-out occurred and all retries exhausted). Phase 3 is repeated for each file in the broadcast request, and for any other broadcast requests associated with this distribution. Phase 3 is complete when all responses from the Export Data Server have been received by the Distribute Software Server.

Phase 4

Phase 4 begins when the Distribute Software Server sends a datagram containing the relevant release information from the SBOM to the Distribute Software Agent on each target processor. This phase verifies the integrity and completeness of the release and makes it available to be used. The bit map signifies whether any action should be taken. There is a special file in the release, known by name to the agent. If it is present it is processed first and executed as a user exit script file, allowing the OS environment to be tailored and maintained without taking the processor off-line. If the file did not need to be broadcast because it was available in the current release, it is copied locally. Each file in the new release has its checksum recalculated and verified against that in the message. The owner and permissions of the file are set, and a symbolic link is made so other software can reference the file independently from its actual name. The processing of each file is separated by a configurable interval (for example, 2.5 seconds) so that it does not perturb the load on the processor. After the last file has been successfully verified, a message is sent to the Monitor and Control Agent to make the release available for local incorporation. A response is sent to the Distribute Software Server signifying that the release was complete to this processor. When all agents have responded (or a time-out was detected), the Distribute Software Server sends the Monitor and Control Server a final response signifying which processors successfully received the release, and phase 4 is complete.

Using the New Release

Once a release has been loaded onto the processors, the system operator selects the time when the change from the current release to the new release will occur. The system allows three (3) operational releases on each processor, the intention being that the three releases normally represent:

1. fallback release - a previous release known to be "good"
2. current release - the currently executing release
3. future release - the release that has just been distributed

After a new release has been distributed to the processors, the system operator directs a portion of the system to restart on the new release. Because there are two independent subsystems providing air traffic functions, using different processors and networks, there is no danger that changed message formats in the new release will cause conflicts. Procedurally, ATC operations are transferred to the backup or primary subsystem while the other is restarted into the new release. After the set of processors that was restarted with the new release has come back up and demonstrated the ability to reliably perform correct operations, ATC operations can resume on it, and the second subsystem can be restarted with the new release.

To protect against a problematic release the operator can at any time revert to a previous release via the same command to restart on a different release. In addition, if a processor fails IPL too many times successively (for example, 3), a release independent bootstrap program will automatically revert to the prior release.

Implementation Details

Each operational component consists of an Ada 83 task in a set of other tasks that comprise an address space under AIX. The basic unit of work in the system is an event. An event may be either a message or a timer. Messages are defined in common interface packages available to all components that use the interface. A message may be sent from one address space entity to another and is directed to a given task via a priority queue. Timers are associated with an interval and are used to signify that it is time to do something (not an error), or the time to do something has passed (error). They are important because they allow the distribution to be paced and avoid exceeding an established limit. An event may be processed by only one task. The amount of time that a task may work on an event is limited. It may be extended for certain long running operations, such as OS system calls or I/O.

Tasks that exceed the limit are considered to be faulty and may be terminated with their address spaces restarted.

Although the implementation is primarily in Ada 83, it uses some OS system calls and C functions invoked from the Ada run-time. This is done to manage the file systems for the releases and calculate the 32 bit checksum associated with a file. Due to the infrequent occurrence of distribution, the task allocated to it also performs other functions, namely file transfer for other commands using a connection-oriented protocol. Although only one distribution can occur on a network at a time, all processors on the network receive the release at the same time. The distribution method scales up well to larger numbers of processors.

Laboratory and test results have demonstrated that Distribute Software meets its requirements. It has been tested with up to 172 processors on the Token Ring and 99 on the Ethernet. Throughput for the distribution approached the 0.456 Mbps (57 Kbytes per second) limit and increased the LAN utilization by the expected 3 to 4.5%. The time to distribute a release varied by release size and number of processors, ranging between 40 to 60 minutes.

The software is not very large, nor overly complex. The approximate Source Lines Of Code (SLOC) associated with the main distribution components are :

- Distribute Software Server -- 1500 sloc
- Distribute Software Agent -- 800 sloc
- Export Broadcast Server -- 600 sloc
- Export Broadcast Agent -- 400 sloc

The cyclomatic complexity, adjusted for project considerations, does not exceed 10 in any given unit.

Alternative Considerations

The current SBOM-directed, broadcast distribution technique was not the first design. The system requirements dictated use of an OSI communications stack on the primary network (rather than TCP/IP), and in 1988 when the first version was designed there were no CAS products on the market to provide this function in such an environment. The initial implementation used hard-coded Ada types to define the interface between the support and operational environments. This interface underwent constant revision as other areas of the system were developed (new file types, permissions, etc. were needed). This meant that the previous version of the software already running could not understand the latest Ada types, and had to be reloaded on the processors manually whenever the common code had changed the interface from this SBOM. Another deficiency of early versions was the use of connectioned messages, which forced a file to be transmitted over the LAN for each processor. This is the most commonly used technique in the CAS products surveyed. The performance of a 90 Mbytes paced distribution repeated 200 times was clearly unacceptable and we switched to the broadcast technique described above.

Today there is a richer set of CAS products that perform distribution, although still primarily on TCP/IP. Some products, such as DM6000 by IBM and UNICENTER by Computer Associates, use a point-to-point protocol and do not pace the transmission. Other products such as Tivoli/Courier have migrated toward paced distribution implementations.

There are still some hurdles to overcome before general CAS products can replace the custom distribution solution in a demanding environment such as Air Traffic Management (ATM). For example, the rest of the ATM computer human interface is presented via custom format specially designed to help controllers in safety critical situations where quick recognition of data is imperative. They work in a nearly dark area with special display hardware. The Remote Graphics Language (RGL) product is used, which allows views to remain on-screen permanently. A CAS solution based on X Windows degrades performance and conflicts with the permanent view concept. It requires extensive color customization to display well in a dark area without making other windows difficult to see. Additionally, the operational system needs to know distribution was successful prior to switching execution over to the new release. This would require custom user exits in a CAS product. Another problem is that CAS distribution products do not have the scope required to regulate network traffic. The DSR system software enforces control when the LAN traffic is heavy to prevent interference with critical functions.

Future Enhancements

Although the current system meets or exceeds all current requirements, there are several aspects that if given the chance we would improve:

- Variable sized file systems

  The fielded version creates a fixed size file system for each release regardless of the necessary size. Because of the nature of the system, the size of releases doesn't vary much. It is virtually the same set of executable and data files each time. To generalize the distribution function to other applications we could get the size from the SBOM. By adding up the size of each file in the SBOM that will be distributed to a certain platform, the size of the release specific to that platform can be calculated and the file system created for that size.

- Distribution by function rather than processor name

  The SBOM now identifies all operational processors with the identifier "ATC" while the support processors each have unique identifiers for the different configurations. The result is that all operational processors receive the exact same release, which actually is useful given the

maintenance strategy of the current customer which may take a processor from one position and place it in another role after service. A planned future enhancement is to switch from using processor identifiers to using functional descriptions. For example, the off-line environment would use "compile" and "edit" rather than "SDE" which is the current identifier for the Software Development Environment processor. This enhancement would allow different size environments to combine or split functions without changing the SBOM.

- Additional SBOM stanzas

  The current source SBOM only has three types of stanzas; imbed, source, and target. A version stanza would be useful for systems that are expected to store and process many prior releases. Similarly more information about how the SBOM was created (automated or manual), would also have some use.

- Logical LAN Partitioning

  Today we do not have to partition the system for cutover because we have two independent systems on completely separate networks. Another future enhancement would be the partitioning of a single switched network via logical partitioning in the switch itself. The higher availability and capacity of new network hardware (i.e. dual attached FDDI) diminish the need for a separate network during steady state processing. Introducing switches to these networks would allow us the ability to use a single network during cutover and other non-steady state processing.

- Seamless completion of distribution upon server failure

  As discussed earlier, should the Distribute Software or Broadcast Data Server fail, the distribution would stop and the user would be notified of its failure, and would have to re-enter the command. Instead, the backup server could assume responsibility for completing the distribution from the point of failure and the user would not have to reenter the command.

- More status information upon agent failure

  The current system reports failures at or near the end of software distribution. Status on failures could be reported in near real-time so the operator could cancel the distribution. A cancel command can be added to stop a software distribution that is in progress.

Conclusions

Promulgating updates to a distributed system while it is operational is an interesting problem. We expect solutions to improve in ability to handle increasing system complexity, scope and capability. Users of distributed systems should develop an expectation that their systems be updated without undue interference or down time. Systems that have a large customer base, such as the Internet, should remain available without degradation in service, even as protocols and standards that affect its software components change. Solving this problem requires a delicate balance between the function and the mission. If the distribution is too slow, the system operator cannot install a new release in a timely fashion. If it is too fast, the network load and response times for critical applications suffer. Mission critical systems cannot tolerate very much interference without causing degradation in function. Self-describing releases are easier to maintain once deployed, not requiring bridge releases to understand changes in the distribution protocols when they occur. Non-interfering software distribution is one method to solve this problem.

Copyrights