Journal of International Information Management

Volume 9 | Issue 2 Article 2

2000

Use of replication technology in a global software development environment

Bhushan L. Kapoor
California State University, Fullerton

Ram Singhania
California State University, Fullerton

Use of replication technology in a global software development environment

Bhushan L. Kapoor
Ram Singhania
California State University, Fullerton

ABSTRACT

In this article, we have proposed a new global software development support system. Until now, the technological challenges in global software development support have been addressed on a semi-automatic ad hoc basis by Groupware technologies such as electronic mail, teleconferencing, electronic meetings, calendaring and scheduling, and workflow. These methods are useful but do not address the issues of site autonomy, and transactional consistency. Our proposed software development support system is based on replication technology. In our system, each software development center has the ability to make additions and modifications. Further, the system also maintains transactional consistency so all sites have the identical copy of documents in near real-time.

INTRODUCTION

Large global distributive software development is a complex and cooperative process. The development work could be done at several locations, and is highly inter-dependent. The locations have a logical partitioning of program components and their ownership. Each location has a primary responsibility for a set of programs, system components, and the related material and is labeled as the owner of these documents. However, the programs may be developed and maintained across the enterprise. So, it is important that each location also has (not limited to read-only) access to the programs owned by other locations. Further, at a single point in time (when the additions and modifications are taking place) the documents at all sites may not be consistent; however, all sites should be guaranteed to have the same documents in near real-time. Existing systems are ad-hoc and deficient in achieving this objective.

Replication technology provides a quick, systematic, and reliable way to disseminate information to multiple locations in a distributed environment. Replication has been frequently used for distributing data across an enterprise. We are proposing a new global software development support system based on replication technology. The system provides desired level of logical partitioning, site autonomy, and transactional consistency in a distributed environment, wherein all sites can work independently and yet are guaranteed to have the same documents eventually.

GLOBAL SOFTWARE DEVELOPMENT

Traditionally, software development was done by in-house and co-located teams. Until 1989, world's most software (at least 60 percent) was produced by the software professionals who worked and lived in USA. However, in the last decade, there has been a proliferation of microcomputers, servers, network, and telecommunications equipment. The demand for software products and software services has increased at a very fast pace. This has been further compounded by the fact that today's computer hardware and software are more powerful and contain many more features and funcitons at substantially lower prices. Thus, the need for software professionals in the U. S. has increased far beyond the supply of qualified personnel. Available supply has simply not met the demand for software professionals despite the escalating labor costs.

To keep up with the demand for software professionals at a reasonable cost, the companies have started looking for the qualified resources outside the company and outside the USA. A recent Information Technology Association of America study (ITAA, 1998) found shortage of 346,000 programmers, system analysts and computer scientists. The report indicates that 40% of the US software companies are hiring immigrants and 16% are outsourcing abroad. Today, more and more software companies are finding it productive to invest in software development in countries, such as India and Israel where qualified well-trained professionals are available at much lower costs. In fact, six of the world's top 12 software development centers are now located in India, according to a ranking by the US Carnegie Mellon Software Engineering Institute (1999). Exports of software services from Israel have grown from $500 million in 1997 to $1 billion in 1998, and to 1.5 billion in 1999 (Israel Software Export, 1999). The growth of exports of software services from India is even more spectacular; from $1 billion in 1997 to $2.7 billion in 1998, and to $3.9 billion in 1999 (Karp, 1999). These numbers are expected to grow exponentially (at least 50% per year) in the next decade. According to industry projections, software exports from India and Israel will reach 112.5 and 45.2 billions of dollars respectively, in the year 2010 (Table 1).

Table 1. Software Exports from India and Isreal (in billions of dollars)

Year	India	Israel
1997*	1.2	0.5
1998*	2.7	1.0
1999	3.9	1.5
2007*	33.5	15.4
2010**	112.5	45.2

* Sources: Israel Software Exports (1999) and Karp (1999)
** According to industry projections

The fact is that software development has gone global. There are many other factors that are enhancing this trend; one of these is that in today's global economy, the market for the software has also turned global. In recent years, Microsoft and other major software vendors have derived bulk (more than 50%) of their revenue from outside the USA (Miller, 1994; Malhotra, 1994). Further, because personal computers, servers, and other network components are increasingly cheaper and have more power, they are becoming more readily available even in the third world countries. In fact, the market for software products is increasing faster outside the USA than within the USA. To gain market access, software companies will rely more heavily on strategic partnerships to develop and sell their software products and services. The trend is that large companies are doing business on a global scale. Table 2 describes the characteristics of software development teams of yesterday (traditional) and today (global (Grenier, 1995). Today's software development team could be from many divisions (or even many companies) in geographically dispersed (including overseas) locations.

Table 2. Characteristics of Traditional and Global Teams*

Traditional	Global
Co-located Members	Distributed Members
Verbal Communication	Electronic (digital) Communication
Same Organization	Different Organizations
Information Distribution (Push)	Information Access (Pull & Push)
Paper Media	Electronic Media
Hierarchical Management	Networked Management
Uni-direction Flow	Bi-direction Flow of Information
Access through Value Added Networks	Access through the Internet

*Source: Adapted from Grenier, R. and Metes, G., *Moving Your Organization into 21st Century*, Prentice Hall, 1995.

While software companies are reaping the benefits of global software development (at significantly lower costs) it has also created some challenges (Thomas, 1997). These challenges include cultural, infrastructure, technological and management issues. Technological issues include primarily the need for increased communications and interoperability, transactional consistency and site autonomy. Until now, some of the technological challenges have been addressed by the existing Groupware technology. Groupware technology builds on five fundamental technologies: electronic mail, teleconferencing, electronic meetings, calendaring and scheduling, and workflow (Goldman, 1999). Groupware technologies do allow the employees to obtain timely information and increase their communication and collaboration efforts with others to some extent. However, they only provide a semi-automatic solution and do not solve the problem of each

dispersed location to have a near real-time and automatic (without human intervention) access to the latest versions of all programs, procedures, re-engineering specifications, test data, test results, on-line help, and other supporting material. Nor do they address the issues of site autonomy and transactional consistency.

Site autonomy is the ability of each site to do its work independently of others and the effect it has on the operations of other sites. Transactional consistency is that after the data has been replicated, the data at all sites in the application will appear as if the transactions had actually occurred at that particular location.

In summary, the desired characteristics of a distributed global software support system which have not been addressed by the existing system include:

1. Each software development center must have the ability to make additions and modifications. This is an essential requirement.
2. System must maintain transactional consistency; that is, all development centers eventually will have the identical copy of documents. This is also an essential requirement.
3. Transactional consistency should be real time.
4. Each development center should be autonomous; that is, each development center is able to work independently of others.

We are proposing a new global software development support system based on replication technology. Replication can be most simply defined as providing copies of documents (related to a specific software project) to different locations and then automatically synchronizing so that all sites eventually have the same information. Replication technology has matured in recent years to provide a quick and reliable way to disseminate data, procedures and text to multiple locations in a distributed business environment. Major DBMS software vendors, such as Oracle, Sybase and Microsoft are packaging this powerful technology in their DBMS products. The following section explains how replication technology can alleviate the problems in the distributed environment of global software development to provide superior communications and operability than supported by the existing systems.

REPLICATION TECHNOLOGY

Our replication scheme is based on Microsoft SQL Server 7.0 replication model. The main components in a replication scheme consist of a publisher, several subscribers, distributors, publications and push and pull subscriptions. The publisher is a server that makes publications available for replication to other servers. The publisher specifies which documents are to be replicated and flags those changed since the last synchronization process occurred. Subscribers are servers that store replicas and receive updates. Previously, updates could only be performed at the publisher providing read-only access to the subscribers. Now, in a variation with the 'Immediate Updating Subscribers' Option,' the subscribers are also allowed to update and send documents to the publisher. This information (new or changes) sent from subscribers to the publisher

is eventually distributed to other sites. This option will be further explained in a subsequent section. The publication is simply a collection of documents to be replicated. The distributor is a server that contains the distribution information. The exact role of distributor can change in each type of replication.

There are two basic types of subscriptions: Push and Pull subscriptions. In Push subscription the publisher propagates (or pushes) the changes to a subscriber without a specific request from the subscriber. Push subscriptions are good for applications where you need to send changes to subscribers as soon as they occur, or for applications where publications are scheduled for publications on a periodic basis. Push subscriptions require that there are reliable links between subscribers and the publisher. In Pull subscriptions, the subscriber requests changes from the publisher. Pull subscriptions are good for loose and unreliable links. A single system can support a mixture of both push and pull subscriptions.

MS SQL Server provides three major types of replication to use: Snapshot, Transactional and Merge replications. Further, the 'Immediate Updating Subscribers' option is available with Transactional replication, and the 'logical partitioning' is available with Merge replication. Each of these has different benefits and capabilities to satisfy the application requirements, such as transactional consistency and site autonomy. However, these replication types are not mutually exclusive and it is not uncommon for the same application to use multiple replication types (as is recommended in the present case study of global software development).

SNAPSHOT REPLICATION

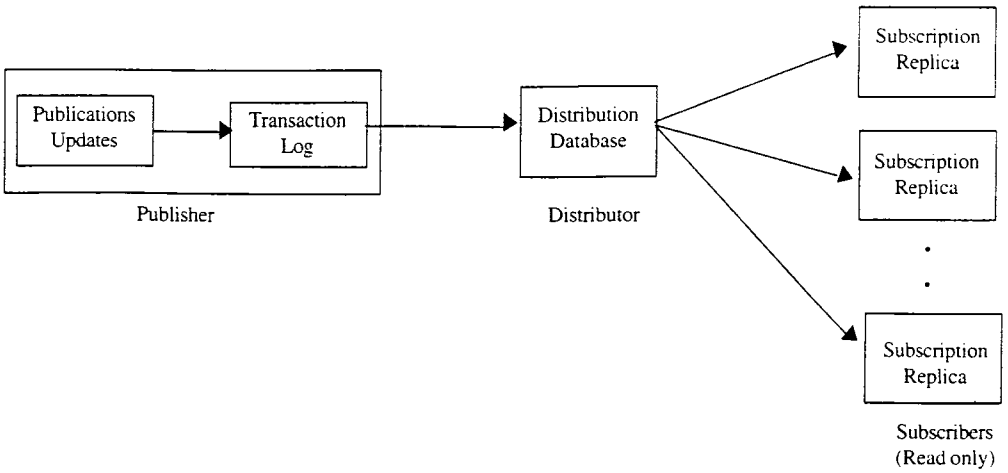
The snapshot replication is the simplest type of replication. It takes a snapshot or a total refresh of the published documents in the database at one point in time. The snapshot replication then sends a complete set of documents to all participating sites instead of sending the changes only. If the article is large, it can require substantial network resources to transmit. Further, if the subscribers are also permitted to make modifications, snapshot replication provides low site autonomy limiting its usefulness in a disconnected environment.

Snapshot replication is appropriate in application scenarios such as look-up tables, or in decision support systems in which data latency requirements are not strict, data volumes are not excessive, and subscribers do not need update access from the publisher. The snapshot replication is not suitable in cases such as distributed software development where program components need to be created and modified at each development site.

TRANSACTIONAL REPLICATION

The second mode of replication is called transactional replication. Transactional replication monitors changes to the documents in the transaction log of the publisher server. Any modifications are then distributed to the subscribing servers, continuously or at scheduled times. The changes are first sent to the distribution database, which holds them until they can be distributed to the subscribers. This process is depicted in Figure 1.

Figure 1. Transactional Replication Process (with Read-only Subscribers)



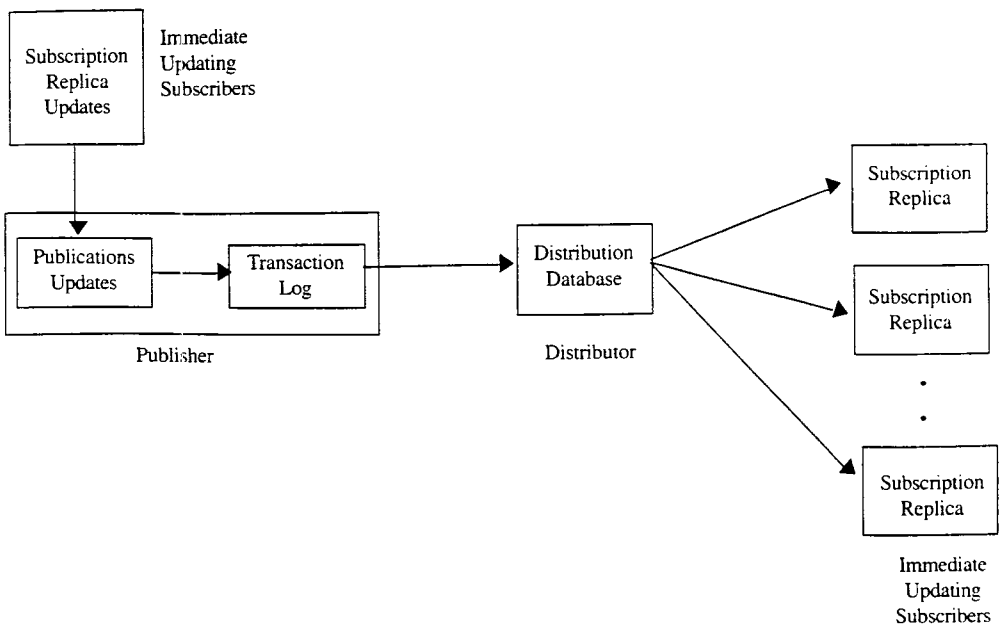
Ultimately, all the subscribing sites will achieve the same values as those at the publisher. If subscribers need near-real-time propagation of changes, they will need reliable links to the publisher. Transactional replication in a well-networked environment can provide low latency (near real-time) to subscribers. Push subscribers would often receive changes soon after they occur at the publisher, provided the network link is fast and reliable. Pull subscribers will receive changes as and when requested (Microsoft, 1998).

Immediate Updating Subscribers Option

In their simplest form, transactional replication is based on a model of one-way replication, in which data is modified only at the publisher and flows downstream to a subscriber. However, some applications, such as distributed software development, require the ability to update documents at subscribing servers and have those changes flow to other sites. The 'Immediate Updating Subscribers' option, available with transactional replication, allows data to be updated at subscribing sites. This option allows a subscriber to update the copy of its local data, as long as that update can be immediately reflected to the publisher with the partial two-phase commit protocol. A partial two-phase commit is a procedure that ensures transactions when applied to an immediate updating subscriber, also applies to the publisher immediately. The partial two-phase commit is automatic and, if it is successful, the subscriber can work with the changed values immediately. However, if the publisher is not available, then documents cannot be updated on the subscriber. The subscriber also does not need a distributor, since the publisher, through its distribution server, will handle the task of propagating the modifications to other subscribers. The partial two-phase

commit ensures transactional consistency because the documents are updated as though they were on the publisher. This process is depicted in Figure 2.

Figure 2. Transactional Replication Process (with Immediate Updating Subscribers)



This approach does not have the availability constraint of doing full two-phase commit to all participating sites because partial two-phase commit only requires that the publisher is available. In full two-phase commit, changes have to be made on all servers simultaneously (or on none of the servers). This requires that all the servers and links to them must be available before any changes can be made on any server.

Full Two-Phase Commit

A full two-phase commit is a process of commitment among replication servers in which the servers first vote on whether or not they can commit a transaction. If all the replication servers vote yes, the transaction is committed at each server. If any server votes no, the transaction is aborted. The two-phase commit protocol is used to synchronize updates on replication servers, so that they either all succeed or all fail.

With full two-phase commit, there is no need of a distribution server, because each transaction is applied at every other server directly and immediately. However, the site autonomy is extremely low, because every site must be available for any site to make any modifications. In a case such as distributed software development (with slow and unreliable links), full two-phase commit protocol would not be a feasible solution.

MERGE REPLICATION

Merge replication also allows the subscribers to update documents. In this replication, publishers and subscribers can work independently and connect periodically to synchronize their changes. Two or more sites may add or update the same document, resulting in a conflict during Merge replication. When conflicts occur, merge replication provides automatic conflict resolution. The conflict can be resolved automatically based on factors such as ownership, assigned priorities, who first submitted the change, and/or a combination of multiple factors. Documents are replicated and changes are applied to other sites only when the reconciliation process occurs.

Although conflicts can be resolved automatically, but they could result in loss of the transactional consistency. In order to resolve conflicts, modifications may have to be rolled back or discarded.

Merge Replication with Logical Partitioning

In merge replication with logical partitioning, each site is responsible for program components that are different from other sites. This will avoid conflicts and therefore is a useful strategy for maintaining transactional consistency. Merge replication with logical partitioning provides transactional consistency and the highest level of autonomy for any replication solution. These characteristics make merge replication an ideal solution for distributed software development, in which users need full read/write access to local replicas of documents in a disconnected environment.

COMPARISON OF REPLICATION TYPES

We have considered the following replication types: snapshot replication, transactional replication with immediate updating subscribers, merge replication with logical partitioning, and full two-phase commit. We have also discussed the desired characteristics of the distributed global software development support system. Table 3 summarizes the essential and desired characteristics for each replication type.

From Table 3 we conclude that snapshot replication does not satisfy an essential requirement; that the subscribers must be able to update. Hence, snapshot replication is not a suitable candidate for our system. Further, the full two-phase commit has a vary low level of site autonomy. When the network contains slow and unreliable links (a global network will typically

Table 3. A Comparison of Replication Types in Distributed Glocal Software Development Support Systems

Replication Type	Subscribers' Update Capability	Site Autonomy	Latent Transaction Consistency	Real-Time Transactional Consistency
Snapshot	No	Low	Yes	No
Transactional (with immediate updating subscribers)	Yes	High	Yes	Near-Real Time
Merge (with logical partitioning)	Yes	High	Yes	Delayed
Full Tow-Phase Commit	Yes	Low	Yes	Yes

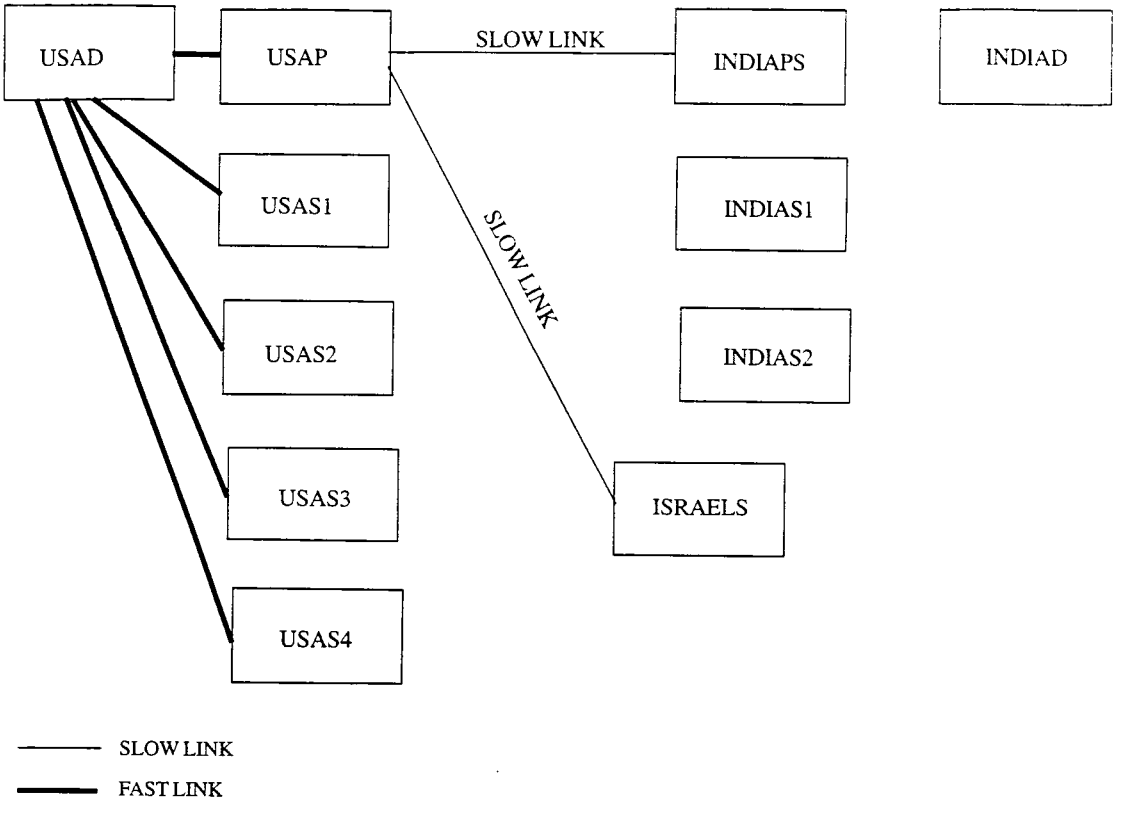
have slow and unreliable links) full two-phase commit is not a feasible option either. Transaction replication with immediate updating subscribers maintains a high level of site autonomy as long as there is a fast and reliable link between updating subscribers and the publisher. System also maintains latent transactional consistency. Subscriber updates are replicated throughout the system in near real-time. Finally, merge replication with logical partitioning maintains a high level of site autonomy even in the slow networks. This replication type also maintains transactional consistency. The transactional consistency may be delayed for a few minutes (or even hours depending upon requirements) but may not be a problem for some environments.

CASE STUDY: A Hypothetical But Realistic Example

A typical global software development communication infrastructure has a combination of fast and reliable links as well as relatively slow and unreliable links. In our example, we have sites in three countries located in two different continents. Sites USAP, USAS1, USAS2, USAS3, USAS4, and USAD are in the USA and are connected to one another by fast and reliable links. Similarly, sites INDIAPS, INDIAS1, INDIAS2, and INDIAD are located in India and have fast and reliable links. On the other hand, sites USAP, INDIAP, and ISRAELS are connected by slow and unreliable links (Figure 3). Because of this diversity in the quality of communication infrastructure in our case study, it is important that we select the appropriate replication types mix suitable for the link qualities.

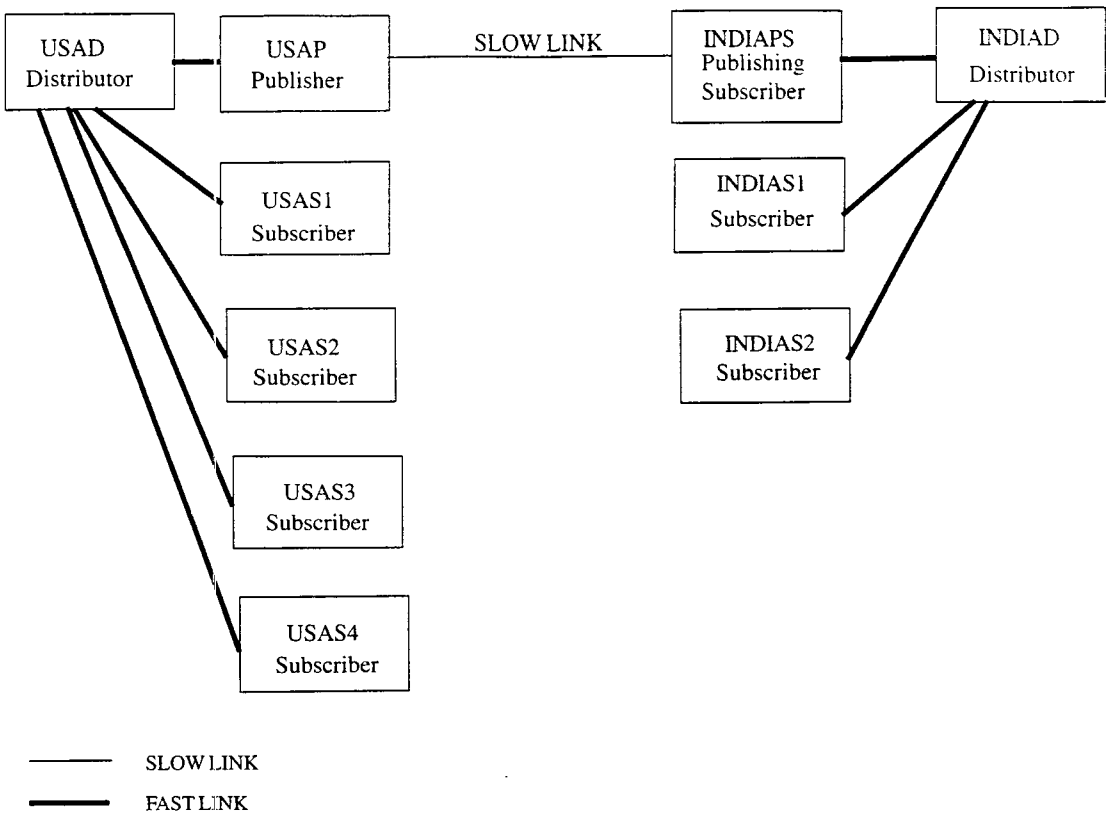
When we need tight consistency between sites, the transactional replication is most appropriate. However, the transactional replication also requires fast and reliable communication networks. The transactional replication is based on monitored changes in the transaction log of the source server. Any modifications are then distributed to destination servers.

Figure 3. Building the Communication Infra-Structure Design



In our case study, we have implemented transacitonal replicaiton on USAP, USAS1, USAS2, USAS3, and USAS4 servers. USAP is the publisher and all other servers are subscribers. In our transactional replication design (Figure 4), these subscribers are configured with 'Immediate Updating Subscribers' option and therefore are allowed to update information as long as the update can be immediaely applied at the publisher using the two-phase commit protocol. Each subscriber is allowed to make updates and these updates are replicated to the publisher immediately first and subsequently to other subscribers in the near real-time. USAD is the distribution server that stores all changes that need to be distributed to subscribers. Some of these changes, such as project standards and policies, and re-engineering policies originate at the publisher, USAP and others may start from the publisher or any subscriber. The distributor USAD and the publisher, USAP may be located on the same machine or connected by a fast and reliable link.

Figure 4. Building the Transactional Replication Design (with Push Subscriptions



An identical transactional replication scheme has also been implemented on INDIAPS, INDIAS1, INDIAS2, and INDIAD servers. INDIAPS is a Publication server, INDIAS1 and INDIAS2 are subscribers and INDIAD is a Distribution server. INDIAS1 and INDIAS2 are also 'Immediate Updating Subscribers', and therefore each subscriber is allowed to make updates and these updates are replicated to the publisher immediately and eventually to the other subscriber as well (Figure 4).

In transactional replications, a log reader agent monitors the transaction lob of each database participating in replication. When the log reader agent finds replication transactions, it hands them off to the distribution database, which holds them until they can be distributed to the subscribing servers. Users are permitted to access the destination documents while they are being updated.

The push, rather than the pull, subscriptions are appropriate for transactional replications of this case study. Push subscriptions are good for applications where you need to distribute changes to subscribers whenever and as soon as they occur. Push subscriptions are also suitable for reliable, well-connected fast links between the publisher and subscribers.

The publication database's transaction log holds the transaction marked for replication until the log reader moves them into the distribution database. Then the log reader agent truncates the transaction log of the distribution database. Once this occurs, the transaction log of the publication database can also be safely truncated, which purges only transactions not marked for replication.

Merge replication allows data to be updated independently at multiple sites. The sites then synchronize with each other later at prescheduled times or manually any time on demand. Merge replication works will in a situation that has: (1) loose and slow connections, (2) logical partitioning of data, and (3) updates performed at different sites do not conflict with each other.

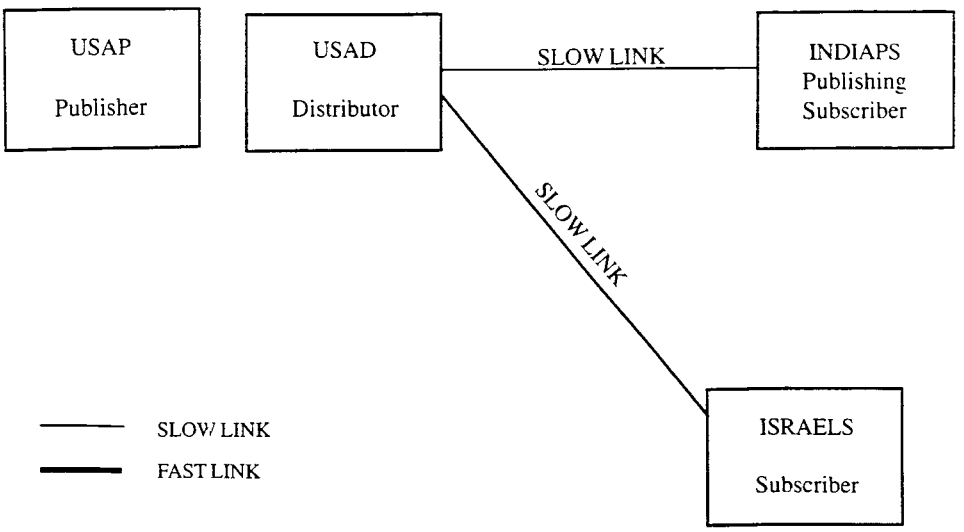
The pull subscriptions are a good choice for the merge replication of this case study. With a pull subscription, the subscriber asks for periodic updates of all changes to the publisher. Pull subscriptions are appropriate for slow and unconnected links between the publisher and subscribers. Pull subscriptions to merge publications have a Merge Agent that runs on each subscriber, rather than the publisher. This offloads some processing work from the publisher to subscribers.

The servers located at USAP, INDIAPS and ISRAELS sites are connected with merge replication (Figure 5). They are scheduled to merge their changes just once a day sometimes after or before their normal working hours. This not only will save transmission time and cost, but also each site, USAP, INDIPS and ISRAELS would receive new information and changes, right before their normal work starts in the morning. Because of some 12-hour difference in time zones between sites in USA and INDIA and some 10-hour difference in time zones between sites in USA and ISRAEL, there is no overlap in their normal working hours.

SUMMARY AND CONCLUSION

Large global distributive software development is a complex and cooperative process. The global software development is also substantial and growing at a very rapid pace. Some newly emerging and developing nations, particularly India and Israel, are very attractive centers for software development. Until now, the technological challenges in global software development support have been addressed on a semi-automatic ad hoc basis by Groupware technologies, such as electronic mail, teleconferencing, electronic meetings, calendaring and scheduling, and workflow. These technologies are useful but do not address the issues of site autonomy, and transactional consistency. In this article, we have proposed a new software development support system that is based on replication technology. The replication technology is available from major software vendors such as Oracle, Sybase, and Microsoft.

Figure 5. Building the Merge Replicaiton Design (with Pull Subscriptions)



In our proposed system, each software development center has the ability to make additions and modifications. Further, the system maintains transactional consistency so all sites have the identical copy of documents in near real-time. Each development center is also able to maintain autonomy - that is, each development center is able to work independently of others.

We considered four major replication types: snapshot, transactional replication with immediate updating subscribers, merge replication with logical partitioning, and full two-phase commit. We discussed the essential and desired characteristics of each replication type and concluded that transactional replication with immediate updating subscribers and merge replication with logical partitioning are the best replication types for our system. At the end, we have applied our conclusion to a hypothetical but realistic case study.

REFERENCES

Carmel, E. (1999). *Global software teams.* Prentice-Hall.

Carmel, E. & Sawyer, S. (1998). Packaged software development teams: What makes them different? *Information Technology & People, 11*(1).

Carmel, E. & Bird, B. (1997). Small is beautiful: A study of packaged software development teams. *Journal of High Technology Management Research, 8*(1), 129-148.

Grenier, R. & Metes, G. (1995). *Moving your organization into the 21st century.* Prentice-Hall.

Grinter, R. E. (1997, September). Doing software development: Occasions for Automation and formalization. *European Conference on Computer Supported Cooperative Work,* pp. 7-11. Lancaster, UK.

Goldman, J. E., Rawles, P. T., & Mariga, J. R. (1999).

Israel Software Exports. (1999). Available from *library@export.gov.is*; Israel Holland Trade; Israel Association of Software Houses at software@industry.org.il.

Information Technology Association of America. (1998). *Report on software labor shortage.* http://www.itaa.org/

Jarvenappa, S. & Tractinsky, N. (1995). Information systems design decisions in global versus domestic context. *Management Information Systems Quarterly, 19*(4), 507-534.

Karp, J. (1999, September). New corporate gurus tap India's brainpower to galvanize economy. *Wall Street Journal,* A24.

Leonard, D. A., Brands, P., Edmonson, A., & Fenwick, J. (1997). Virtual teams: Using communication technology to manage geographically dispersed development groups. In *Sense and Respond: Capturing Value in Network Era.* Cambridge, MA: Harvard Business School Press.

Malhotra, Y. (1994). Controlling copyright infringements of intellectual property: The case of computer software. *J. Systems Management, 45*(6), 32.

Meadows, D. J. (1996). Globalizing software development. *Journal of Global Information Management, 4*(1), 5-15.

Microsoft (1998). Microsoft SQL server - replication for Microsoft SQL server 7.0 *Microsoft Part Number: 098-80829.*

Miller, C. L. (1994). Transborber tips and traps. *Byte, 19*(6), 93.

Rothman, J. (1998, August). Managing global teams. *Software Development,* 36-40.

Thomas, S. L. (1997). Collaboration may bring headaches to IS managers. *LAN Times, 14*(25).

Vitalari, N. & Wetherbe, J. C. (1996). Emerging best practices in global systems development. In *Global Information Technology and systems Management: Key Issues and Trends.* Nashua, NH: Ivy League Publishing.