공동작업을 위한 응용 프로그램에서의 동시성 제어 문제 및 인터페이스 설계에 관한 연구

윤 석 환[†] · 이 재 영[†] · 박 치 항[†] · 신 용 백[††]

요 약

본 논문은 공동작업을 위한 응용 프로그램의 설계 시 고려되어져야 할 사항들을 분류하고 이 들이 사용자 인터페이스에 미치는 영향에 관한 연구 결과를 제시한다. 사용자와 시스템간, 사용자와 사용자간의 실시간 상호작용에 바탕을 두고 있는 공동 작업을 위한 응용 프로그램은 실시간 분산 시스템적 성격 외에도 사용자 인터페이스라는 제약 조건을 가지고 있다. 따라서 프로그래머들은 단일하고 논리적 일치성을 보장하는 사용자 인터페이스를 제공해야 하며 이를 위해 고려 되어져야 할 사항들로는 크게 동시성 제어 전략, 시스템 구성(topology), 객체들의 복제 여부(replication scheme)등을 들 수 있다. 이 중 가장 중요한 항목은 동시성 제어 전략이며 본 논문은 시스템 구성과 객체의 복제 여부와 함께 동시성 제어 전략이 사용자 인터페이스에 미치는 영향을 연구하였다. 공동 작업을 위한 편집기인 Coshed/SAS를 통해 위 요소들의 적용 예를 보였다.

On a Concurrency Control and an Interface Design of Collaboration-aware Applications

Seok-Hwan Yoon[†] · Jaeyoung Lee[†] · Chee-Hang Park[†] · Yong-Back Shin[††]

ABSTRACT

In this paper we discuss the elements which must be considered for collaboration-aware application design and their effects on user interfaces. Collaboration-aware applications have inter-user and user/system interaction features besides generic real-time distribution system features and this restricts the design freedom with the requirement of consistent user interfaces. Programmers, therefore, must provide uniform and logically consistent user interfaces to users and, for this end, they should consider main design features such as concurrency control algorithms, system topology and object replication scheme. Among the design factors concurrency control algorithms have relatively significant impact on user interfaces and we consider the impact of concurrency control algorithms along with that of the system topology and the object replication scheme. Coshed/SAS, a group editor, is given as an example of application of the factors.

1. Introduction

Recent surge of cooperative applications such as shared editor and teleconferencing has created new work environment known as CSCW(computer supported cooperative work). Users work in a *virtual shared work space* provided by CSCW applications

† 정 회 원:한국전자통신연구소 컴퓨터연구단
†† 정 회 원:아주대학교 산업공학과
논문접수:1996년 2월 16일, 심사완료:1996년 6월 7일

and can communicate with each other while working on shared objects:documents, graphics and even multimedia data. Cooperative applications have been considered as either collaboration transparent or collaboration aware[1].

Users of collaboration transparent applications are unaware that more than one interface is provided. Through the use of collaboration aware applications users should be able to work in an environment, virtual shared work space, that simulates an ordinary collaboration environment in which users are highly informed of other users' activities. Imperfections of network, however, present serious obstacles in maintaining a unique copy of the workspace and users can be presented with different realizations of the shared workspace. The user interface inconsistency is further distorted by simultaneous user inputs and can seriously degrade effectiveness of collaboration aware applications.

In this paper, we analyze the concurrency control problem and the effects on user interfaces of collaboration aware applications. The importance and impacts of concurrency control on groupware applications have been researched by many researchers[2, 3, 4, 5, 6, 7]. We present a few issues of collaboration aware applications and Quadruple notation scheme. We, then, study the relationship between such factors as concurrency control and system configuration and user interfaces. We consider Coshed/SAS a graphical group editor as an example.

2. Collaboration Aware Applications

Collaboration aware applications feature most aspects of distributed systems and require another consideration over traditional distributed systems: Multiple Human-to-Machine interfaces are provided and they are communicate in real time.

2.1 Issues

Visualization If each user sees a slightly different or out-of-date version then the session's cohesiveness is soon lost. WYSIWIS(What You See Is What I See) interfaces require fast response time and an appropriate concurrency control mechanism.

Distributed Usage of groupware applications in wide-are networks where users are separated by great physical distance is one of the main advantages of groupware applications. Developers should take suitable counter measure for slow response time caused by the physical separation.

Replication Because the response time demands of collaboration aware systems are so high, the object table is usually replicated for each participant. This allows more time critical operations such as window redrawing caused by window repositioning or scrolling to be performed locally.

Volatility Participants are free to come and go during a session.

2.2 Quadruple Notation

For further discussion we assume the object table is replicated over n nodes. An object table is a one-dimensional vector whose elements are also vectors. In the simplest case of a text editor, objects are characters and the object table is character strings. Objects of graphical editors contain more attributes such as color, bounding rectangle and line width. Operations are defined as replacement of a *contiguous subarray* with an array of objects and can be presented as quadruple vectors. Below we summarize notations necessary for further discussion.

Object χ

Object table $V_j(t)=\{\chi_1, ..., \chi_i, \chi_m\}$
where $i=1, 2, ..., m$ and $j=1, 2, ..., n$

Operation $O=(m, k, V_{old}, V_{new})$
$V_{old,new}=\{\chi_1', ..., \chi_i', ..., \chi_l'\}$

where m is the cardinality of the object table and n is the number of nodes of the system. The object table $V_j(t)=\{\chi_1, ..., \chi_i, ..., \chi_m\}$ corresponds to the object

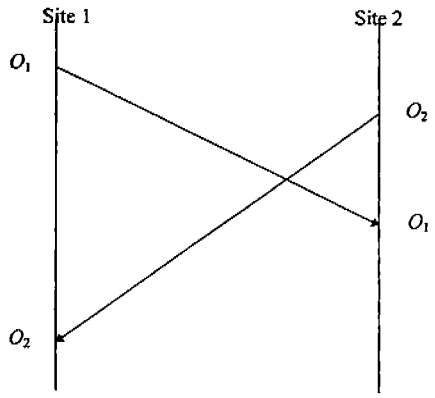
table of j-th node at time t. If object tables match with each other, index j can be removed and the table is called the *reference object table* at time t. Below we assume the initial object tables are the reference table. V_{old} and V_{new} correspond to an old subvector and the replacing vector. Definition of contiguousness of a subvector is intuitive. A subvector $V' = \{\chi_1', ..., \chi_j', ..., \chi_l'\}$ is contiguous with respect to the original vector $V = \{\chi_1, ..., \chi_i, \chi_m\}$ if

$$\chi_j' = \chi_{j+k-1} \quad \text{and} \quad \chi_j' \neq \text{NULL}$$

where $j = 1, ..., l$ and k is the original index of the beginning object of the subvector. When V_{old} is null the operation is insertion and if V_{new} is NULL the operation is deletion.

Given $O = (m, k, V_{old}, V_{new})$
If (V_{old} != NULL)
    replace the subvector Vold with the new vector V_{new}
else if ($k == m$)
    append the new vector to the original vector.
else
    insert the new vector after k-th object

The requirement of the contiguousness of a subvector is a pure design issue. Since users usually create a compound object with logically connected basic objects the notation offers logically maximal and application independent units. Another design issue is that there are two choices of defining insertion point when insertion parameter k of the quadruple vector O is given: Before or after the k-th object. Below we present an exemplary algorithm in which the new array is inserted after the k-th object.



(Fig. 1) Example of event re-ordering

2.3 Concurrency control problem

The core of concurrency control problem in collaboration-aware applications is non-commutability between two generic operations $O_1(t_1)$ and $O_2(t_2)$. Using the commutator notation this can be put as

$$[O_i, O_j] \equiv O_i * O_j - O_i * O_j \neq 0.$$

This corresponds to the situation where operations are overlapped and executed in the opposite order at each site and the situation is graphically shown in Figure 1. The aim of this paper is not to address this problem which is well-known in distributed systems. CSCW environment has one major issue which is not discussed in distributed systems: User interaction. In the next section we discuss the effects of and the relation between the concurrency control and user interfaces.

3. Concurrent User Interaction and Effects on User Interfaces

3.1 User Interaction

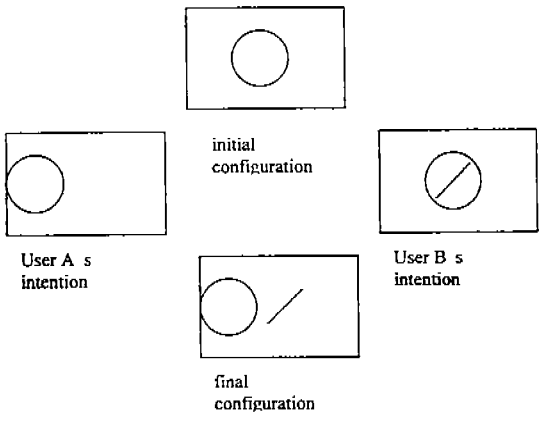
Interaction is an action that is influenced by the presence of, knowledge of, or the activities of another person. For routine collaborative activities, users can make decision with high knowledge of other participants. In CSCW environments, however, user knowledge is limited to the space(virtual shared workspace) provided by the system and user interaction based on limited information can be errant.

Interaction between distributed sites can be thought of as the exchange of messages or execution events. An event goes through many stages before it is completely carried out: creation, local execution, transmission, reception and remote execution. Without concurrency control events created in different sites may not be executed in the same order. As an example, consider a graphical editor shared by a group of users. Users *A* and *B* simultaneously operate on a centered circle by drawing a line passing through the center of the circle and moving the circle up, respectively. The resulting diagram is shown in Figure 2. The final result mismatches both users' intentions and additional corrective actions must be taken by the users.

From the simple example it is obvious that the concept of unique shared workspace and unique shared documents is an idealization of real world situation. Network delay distorts the ideal implementation of single copy shared workspace into possible multiple copy shared workspace. User action is displayed to other users with finite amount of time delay which can mislead users to make incorrect judgment of what others are doing.

3.2 Concurrency control and user interfaces

Conflicts of user interactions can happen frequently and unexpectedly. Without proper mechanism of correcting conflicts, loss of logical context is unavoidable. Many algorithms have been devised to prevent or restore the system from conflicts, and applied to real applications with success of varying degree. We reviewed basic concurrency control algorithms *locking, token, timestamp ordering, and optimistic concurrency control.*



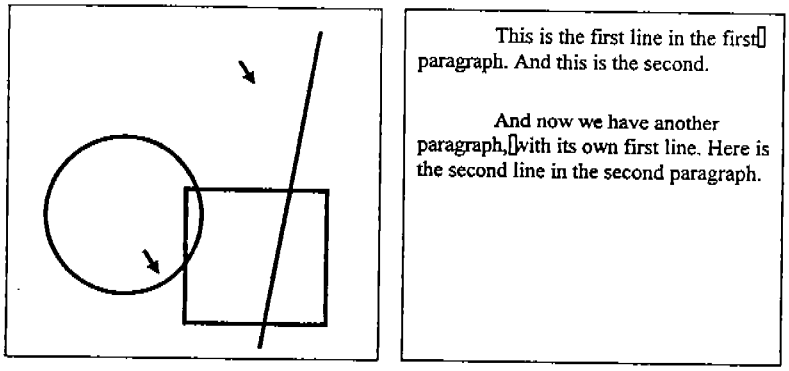
(Fig. 2) An example of a graphical shared editor

Locking Locking is the most common in groupware applications. Before a user operates on an object he should request a lock on it. Depending upon the lock status of the shared object, the replication server responsible for the lock management verifies the request and decides whether to grant or reject the request. After the user acquires the lock, he has the exclusive right to manipulate it. Other users who want to modify the shared object have to wait until the lock is released by the lock owner. This behavior is radically different from collaboration transparent applications which allows users to work on any object at any time.

Consider two collaboration aware applications: a group drawing and a group editor. In group drawing tool, drawing object are circles, lines and other geometric objects. Text objects in a group editor can be paragraph, line or the whole text. As long as user A and B work on different objects, there is no conflict and both continue without noticing presence of the other. If, however, a user attempts to modify already locked objects he may see very unnatural and possibly annoying behavior: the objects won't move. In actual implementations locked objects can be grayed out so that users can be aware of the fact.

Locking in collaboration aware applications is simplest to implement but the behavior of locked objects can be annoying to users.

Token Some derived algorithms worth mentioning are token-based algorithms. The token-based algorithm is a variation of locking. The token holder has the exclusive lock on the whole object table and concurrent operations are totally prohibited. Even though the concurrency control in token-based algorithms is very easy users may perceive turn-taking as unnatu-



(Fig. 3) a) A group drawing tool, and b) a group editor

ral.

Timestamp ordering Operations from clients in the timestamp ordering algorithms are processed in the order of the timestamp issued according to a pre-defined rule. Timestamp ordering achieves global serialization by issuing globally unique timestamps. But the global serialization may or may not coincide with logical serialization(See Figure 2).

Optimistic In groupware applications based on the optimistic concurrency control mechanism, a user can proceed with his operations on shared objects until the operation is verified and committed. Operation verification and commitment is handled by the replication servers. If all servers decide to commit and consensus among them is established, the originating client is notified and local object table is updated. The user proceeds with his work unaware of the verification process. Therefore, as long as conflicting operations rarely occur, the optimistic concurrency control algorithms give cooperation aware applications the closest feel of cooperation transparent applications. If consensus is not established, however, and the operation is aborted, the client must process the abort message and roll back to the reference state. This process of rolling back can be quite hard to implement and confusing to users.

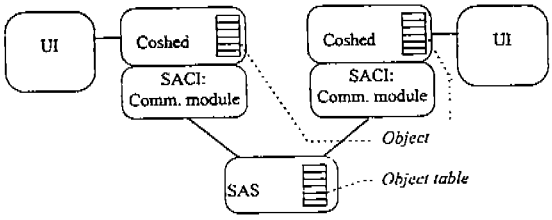
Given $O_1(t_1)$ and $O_2(t_2)$
if ($[O_1(t_1), O_2(t_2)] == 0$)
  $V_{result} = O_2(t_2) * O_1(t_1)\ V_{reference}$
else {
  $V_{result} = V_{reference}$
  NotifyUser
}

Replicated vs. centralized architectures Groupware researchers have long argued the merits of centralized vs. replicated architectures. On the surface, the simplest way of implementing concurrency control is through a *centralized* architecture. Synchronization is easy, as state information is consistent since it is all located in one place. Events will never be received out of order(they are usually handled first-come, first-served). Locking is also easy, as only one copy of the object exists. *Replicated* architectures, on the other hand, execute a copy of the program at every site. Thus each replication must use specific concurrency control algorithms to coordinate their actions and must worry about handling roll-back if optimistic schemes are used.

Because of its simplicity at handling concurrency, centralized architectures for groupware has had many advocates, and one may wonder why a replicated ap-

proach would ever be considered. The answer concerns the issue of latency. A centralized scheme implies sequential processing, and is inherently non-optimistic. A request is received and handled by the central application before the next one can be dealt with. If the system latency is low, this is not a problem. But if it is high, the entire system will become sluggish. A replicated scheme, on the other hand, implies parallel processing which maximizes the use of optimistic schemes. Events can occur in parallel at each replication, with the optimistic method mediating any problems. While overkill for low latency, it can address the interface issues in systems that have noticeable delays.

There is no real answer to whether a centralized or replicated scheme works best for groupware. Rather, it is a set of trade-offs that revolve around the way they handle latency, ease of program installation and connection, programming complexity, synchronization requirements, processor speed, the number of participants expected, communication capacity and cost, and so on.



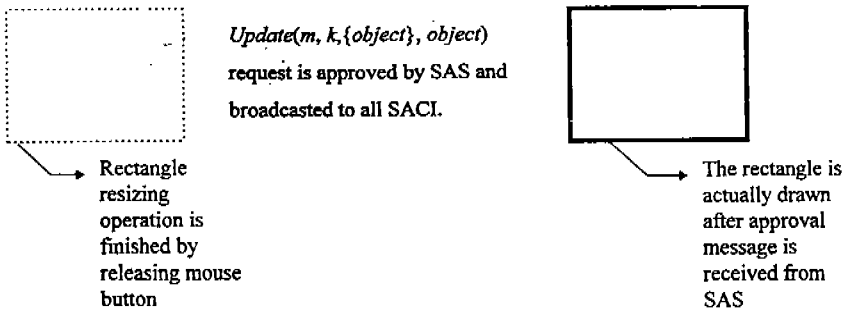
(Fig. 4) Coshed/SAS-An example of collaboration aware applications

3.3 Coshed/SAS

In Figure 4, we presented the architecture of Coshed/SAS, a graphical group editor, which has a hybrid structure of the replicated/distributed and the server architectures. Coshed/SAS consists of three parts. Coshed is the graphical editor part of the system. SAS(Shared Area Server) is the object server and also functions as the central communication point. SACI(Shared Area Client Interface) is the communication client and Coshed uses APIs offered by SACI to communicate to SAS. All objects reside in a virtual common space-Shared Area-maintained by SAS. For robustness and logical consistency, SAS/SACI architecture uses locks as the exclusive concurrency control algorithm. Update operations to shared objects are handled by SAS. Unless otherwise noted any changes in Shared Area are, then, broadcasted to participants including original senders. All user interfaces even that of original sender's are updated only after the broadcast message is received. So simple operations such as line drawing(releasing mouse button) are not immediately reflected on local user interface. This approach, however, contrary to the initial perception of sluggishness, does work well in many situations. Users can be assured that other users' interfaces have been updated correspondingly by observing operations being realized.

Another point worth mentioning is the genericity of objects supported by SAS. SAS does not have pre-defined object format proprietary to SAS. Only attribute handled by SAS is lock status of the object. Other attributes are defined by applications. Five generic operations on objects are defined in SAS/SACI architecture: *CreateObject*, *UpdateObject*, *LockObject*, *UnLockObject* and *DeleteObject*. Therefore, by using SAS/SACI architecture as the communication layer and using locking facilities offered by the architecture as the exclusive concurrency control method, applications can be *completely* shielded from the concurrency control operations. To exchange other messages applications can use two other communication features provided by SAS: *Broadcast* and *TalkTo* which correspond to broadcast and peer-to-peer communication, respectively.

This scheme works best in local are network where network latency is tolerable in most situations. To provide different concurrency control algorithms applicable in situations where network latency is measurably large, however, current implementation of SAS/



(Fig. 5) How user input is updated through SAS to local GUI.

SACI architecture should be extended. For this end usage of Quadruple notation can be helpful. Firstly, it reduces the five object manipulation functions into one, *Update*(*m, k,* {*object*(*attributes*)}, {*object*(*attributes*)}). Below we present how the five separate operations can be represented as one operation.

SAS/SACI	Quadruple notation
CreateObject(*object*)	*Update*(*m, m, NULL,* {*object*})
UpdateObject(*object ID*)	*Update*(*m, k,* {*object*(*attributes*)}, {*object*(*attributes*)})
Change attributes to attributes	
LockObject(*object ID*)	*Update*(*m, k,* {*object*(...,*lock flag*,...)}, {*object*(...,*lock flag*,...)})
Change lock flag of the object locked	
UnLockObject(*object ID*)	*Update*(*m, k,* {*object*(...,*lock flag*,...)}, {*object*(...,*lock flag*,...)})
Change lock flag of the object to unlocked	
This operation is granted only to the lock holder of the object	
DeleteObject(*object ID*)	*Update*(*m, k,* {*object*}, NULL)

It also allows seamless incorporation of additional concurrency control algorithms since concurrency control operations are transferred from SAS to Coshed, local applications. In this way application developers can benefit from such features offered by SAS as member access control, automatic update broadcasting, and peer-to-peer communication while enjoying developmental freedom of choosing appropriate concurrency control and replication schema.

4. Conclusion

In this paper we reviewed the inherent problem of concurrency control in collaboration aware applications and its effects on user interfaces.

Quadruple notation is presented as the descriptive basis of operations in collaborative aware applications. In this notation the object table is a state vector and operations are transformations defined on the state vectors and can be represented as quadruple of two integers and two sub-vectors.

Collaboration aware applications have one notable feature that regular distributed systems lack: Inter-user interactions. The user interaction feature can be realized into four issues that must be addressed by application developers: *Visualization*-Applications must provide WYSIWIS(What You See Is What I See) interface with fast response time and an appropriate concurrency control mechanism. *Distributed*-Users of collaboration aware applications need not be located geographically closely. Developers should incorporate necessary counter measures. *Replication*-Even with many useful features that centralized system offers such as ease of implementing concurrency control algorithm, replication of the object table necessary for collaboration aware applications. *Volatility*-User can join and leave as needs arise. To address these issues

developers should solve the design issues such as choice of one or combination of concurrency control algorithms-*locking, token, timestamp ordering, optimistic concurrency control*-and the replication/centralized debate in conjunction with user interfaces.

A group editor, Coshed/SAS, is presented as an example of collaboration aware applications. Coshed is a graphical editor which uses SAS/SACI architecture as communication and concurrency control layer. SAS was developed to simulate a virtual shared work space and through use of locking as the exclusive concurrency control mechanism, SAS/SACI offers collaboration-transparency to the application developers. Developers can use five generic operations including lock/unlock operations and do not have to worry about the implementation as long as they only use locking. This approach, however, found to be inappropriate in situations where users are geographically separated and network latency is not ignorable and other concurrency control algorithms that allow more concurrent user actions such as optimistic concurrency control algorithms.

A possible extension of Coshed/SAS by using Quadruple notation is discussed. In the extension five generic Coshed-to-SAS operations are represented as one operation *Update*($\{m, k, V_{old}, V_{new}\}$) and there is no concurrency control specific operations. Application developers can use the new SAS/SACI architecture and benefit features such as the network transparency while customize collaboration specific features including concurrency control mechanisms.

REFERENCES

[1] J. C. and K. A. Lantz. Collaboration awareness in support of collaboration transparency: requirements for the next generation of shared window systems, *CHI 1990 Proceedings*, (CHI, 1990) pp. 303-310.

[2] C. A. Ellis and S. J. Gibbs, Concurrency control in groupware systems. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, (1989) pp. 399-407.

[3] I. Grief, R. Seliger and W. Weihl, Atomic data abstractions in a distributed collaborative editing system. In *Proceedings of the 13th Annual Symposium on Principles of Programming Languages*, (1986) pp. 160-172.

[4] A. Karsenty and M. Beaudouin-Lafon, An algorithm for distributed groupware applications. In *Proceedings of the 13th International Conference on Distributed Computing Systems ICDCS'93*, (1993).

[5] M. Knister and A. Prakash, Issues in the design of a toolkit for supporting multiple group editors. *Computing Systems (The Journal of the Usenix Association)*, 6(2), (1993) pp. 135-166.

[6] R. E. Newman-Wolfe and H. K. Pelimuhandiram, MACE:A Fine Grained Concurrent Editor, In *Proceedings of the ACM COCS Conference on Organizational Computing Systems*, (1991) pp. 240-254.

[7] G. Coulouris, J. Dollimore and T. Kindberg, Distributed Systems-Concepts and Design, Addison-Wesley Publishing Company, Wokingham, 1994.

[8] K. P. Eswaren, J. N. Gray, "The Notion of Consistency and Recovery in a Database System," *Comm. of ACM*, vol. 19, no. 11. (1976) pp. 624-633.

[9] P. A. Bernstein, N. Goodman, Timestamp Based Algorithms for Concurrency Control in Distributed Database Systems, In *6th International Conference on Very Large Databases*, (1980) pp. 285-300.

[10] H. T. Kung and J. T. Robinson, "On Optimistic Methods for Concurrency Control," *ACM TODS*, vol. 6, no. 2, (1981) pp. 213-226.

[11] J. Huang, J. A. Stankovic, K. Ramamritham and D. Towsley, "Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes," In *19th International Conference on Very Large Data Bases*, (1991) pp. 35-46.

[12] S. R. Ahuja, J. R. Ensor, and S. E. Lucco, "A comparison of applications sharing mechanisms in realtime desktop conferencing systems." In *Proceedings of the ACM COIS Conference on Office Information Systems*, Boston, April 25-27 (1990) pp. 238-248.

[13] S. Greenberg, "Sharing views and interactions with single-user applications." In *Proceedings of the ACM COIS Conference on Office Information Systems*, Boston, April 25-27 (1990) pp. 227-237.

[14] J. F. Patterson, R. D. Hill, S. L. Rohall, and W. S. Meeks, "Rendezvous: An architecture for synchronous multi-user applications." In *Proceedings of the ACM CSCW Conference on Computer-Supported Cooperative Work*, Toronto, November 7-10 (1990) pp. 273-280.

윤 석 환

1982년 2월 아주대학교 산업공학과(공학사)
1984년 2월 건국대학교 산업공학과(공학석사)
1992년 3월~현재 아주대학교 산업공학과 박사과정
1992년 8월 품질관리 기술사 자격 취득
1986년 1월~현재 한국전자통신연구소 책임연구원
관심분야: 그룹웨어, S/W 공학, 생산정보시스템

이 재 영

1988년 2월 서울대학교 물리학과(이학사)
1990년 5월 The Johns Hopkins Univ.(이학석사)
1994년 8월 The Johns Hopkins Univ.(이학박사)
1994년 12월~현재 한국전자통신연구소 시각언어연구실
관심분야: 그룹웨어, 분산처리, S/W 공학

박 치 항

1974년 2월 서울대학교 응용물리학과(이학사)
1980년 2월 한국과학원 전산학과(이학석사)
1987년 12월 파리 6대학 전산학과(공학박사)
1974년 2월~1978년 2월 한국과학기술연구소 연구원
1978년 2월~1985년 7월 한국전자기술연구소 선임연구원
1985년 7월~현재 한국전자통신연구소 책임연구원 멀티미디어연구부장
관심분야: 멀티미디어, 분산시스템, 그룹웨어, 네트워크 컴퓨팅, 에이전트 아키텍처, 가상현실

신 용 백

1964년 2월 연세대학교 화학공학과(공학사)
1968년 8월 한양대학교 산업공학과(공학석사)
1987년 2월 한양대학교 산업공학과(공학박사)
1973년 10월 생산관리 기술자 자격 취득
1976년 3월~현재 아주대학교 산업공학과 교수/산업대학원장
관심분야: 생산 및 품질공학, 생산정보시스템