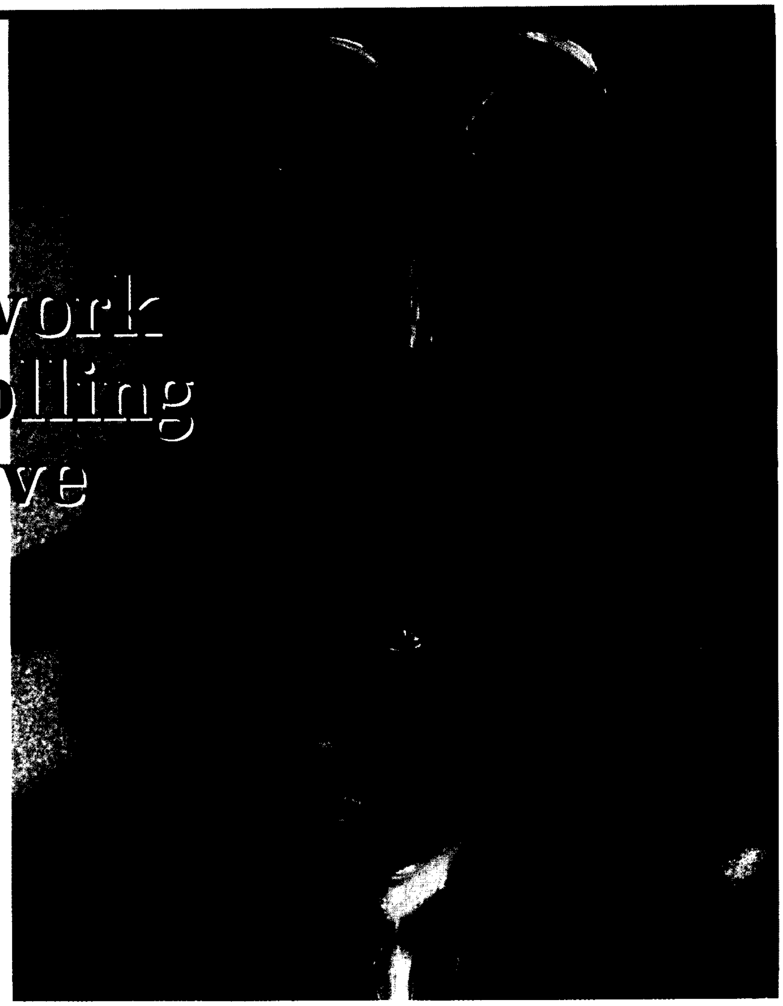
A Framework for Controlling Cooperative Agents



Kuo-Chu Lee,
William H. Mansfield Jr.,
and Amit P. Sheth
Bellcore

For advanced applications involving multiple users, the ITX system supports adaptive cooperation among agents and reliable operations on shared objects stored in heterogeneous and autonomous component systems.

Complex applications in multimedia telecommunications[1] and information services, interactive television, and computer-supported cooperative work[2] are likely to become widely available in the near future. Such applications might involve multiple users participating in a computing environment consisting of heterogeneous and autonomous information resources. The systems that manage the resources (for example, database-management systems) might also be heterogeneous and autonomous. These applications might be supported by distributed and heterogeneous computing and networking platforms (both hardware and software components), and have multiple administrative and access control authorities.

A software paradigm that can support such applications flexibly and reliably is a distributed cooperative task. In this paradigm, an agent supports a user, represents the user to the system, and handles complex interactions with other cooperating agents and system resources. A critical issue in such a paradigm is controlling interactions among the cooperating agents to meet the application objective, despite unpredictable user interventions and system failures.

Conferencing environment. The multimedia conference is a complex application that demonstrates the distributed cooperative task model and the issues in controlling the interactions among cooperating agents. In a typical conferencing-

0018-9162/93/0700-0008$03.00 © 1993 IEEE

system environment, each workstation supports video, audio, and graphic-display capabilities. The users are geographically distributed and connected by high-speed packet networks. A conference call starts with a user issuing a conference call attempt to the system. This call attempt includes the requested *conference configuration* that specifies the user group (the users asked to participate in the conference call), media types for each user in the user group, and the access control information of the conference call (for example, who can modify the conference configuration or request changes in media).

After receiving the conference call attempt, the conference control system must signal members of the user group for incoming calls and allow users to determine whether to accept or reject the call or to change to other media based on their preferences and available resources. While the call is in progress, authorized users can change the conference configuration. In addition, the environment can change because of media failures and other resource constraints.

As the flexibility provided by multimedia conference services increases, interactions among users and systems will become too complex and time consuming for humans to handle directly. To handle the complexity, we can model the multimedia conference as a *cooperative task* in which agents representing users cooperate to support the task. The agent must know the objective it should try to achieve, which in this case is the desirable conference configuration relevant to the user's participation. The conference configuration specifies the user group and the media used to communicate with each user.

Setting up the configuration. To establish an agreeable conference configuration, agents interact with each other on the users' behalf. Each agent tries to allocate the proper resources (video or audio) to meet the conference objective. However, if the agent cannot find the proper resources, it must either inform the user and request further instruction or automatically change the objective to an alternative media type. Other participants then have a chance to alter their media type to accommodate the agent. Obviously, this process might take more than one iteration to complete. The agents, therefore, must be able to observe and react to the changes.

After the agents working together achieve an agreeable configuration, each agent must continue to observe the state of the conference and make adjustments during the time it participates in the conference call. It must react to interventions (for example, change of configuration by a qualified user) and unpredictable resource failures. For example, a failure on one user's video connection might force that user to interact with other users via an audio connection. Or the user who initiates the call might change the conference configuration, asking an additional participant to join the conference call. Agents of the other users then need to establish connections with the new participant. We model adaptations to changes as iterative executions that observe the changes and react to them.

Conferencing application issues have been addressed in various multimedia conferencing and computer-supported cooperative work systems.[3-5] Ahuja, Ensor, and Horn discuss presentation and operation issues in scheduling meetings and running shared applications in a multimedia conference.[3] A report on an integrated multimedia conferencing system[5] addresses the issues in providing flexible capability for user participation in conversations. The negotiation-based call-establishment model[6] addresses problems in resolving conflicting user interests. A research project[1] addresses integrated transport, connection management, and programmability issues related to providing multimedia communication services. A graph-model-based signaling protocol[7] addresses the representation and control issues for conference calls over broadband networks. A computer-supported cooperative work system[4] addresses issues related to group editing and meeting scheduling. The Touring Machine project[8] addresses a wide class of infrastructure issues related to multimedia communications in public telecommunications networks.

In this article, we focus on the system issues related to modeling and control of interactions among cooperating agents in a heterogeneous environment. We propose a system framework that integrates feedback control[9] and transaction-processing techniques[10] to support reliable interactions of the agents with the shared resources. To cope with the heterogeneity and autonomy of hardware and software systems, we represent shared data structures and resources as shared objects, storing them in heterogeneous object databases and providing reliable access using transactions. This approach reduces the need for pairwise interfaces among heterogeneous agents and objects. It also supports concurrent access from agents to distributed shared objects using interactive transactions (ITXs). We use the feedback control technique so that agents can adapt to environmental changes to carry out a cooperative objective assigned by users. We call the interactive transaction-control system the *ITX system*.

System features. The proposed ITX system has the following features:

- It models the coordination and resource-management aspects of a cooperative user agent as an interactive transaction, or *ITX*, that executes transactions on shared objects to achieve the cooperative objective. Distributed agents achieve coordination by manipulating shared objects without direct pairwise communications. Cooperative objectives, shared data structures, and system resources are all modeled as shared objects.
- It uses a novel application-independent criterion called fixed point that defines a stable state of the system with respect to an agent.
- Using *ITX*s and the fixed-point control criterion, it implements a feedback controller strategy that involves observing the changes in the system state (that is, states of the shared objects) caused by user interventions and failures. Thus, after a change, the system can transition to a state that satisfies the cooperative objective.

The interactive transaction model facilitates implementation of a distributed cooperative task in an environment with heterogeneous and autonomous systems and resources. The fixed-point control criterion supports the correctness criterion and, with the ITX model, allows the application to adapt to changing cooperative objectives and changes or failures in the environment.

In this article, we present an overview

of the ITX system, define its components, and describe the ITX system's unique fixed-point criterion for feedback control of iterative interactions. We use the example of a multimedia teleconference to explain additional details and advantages of the ITX system.

Implementing an ITX system and supporting multimedia teleconferencing applications involve issues from many areas of computer science, such as heterogeneous databases, object-oriented systems, transaction management, and multimedia management (including storage, access, communication, and presentation), so we cannot discuss here all the details of the ITX system and the multimedia teleconferencing application. We have implemented a software prototype demonstrating the features of the ITX system discussed in this article.

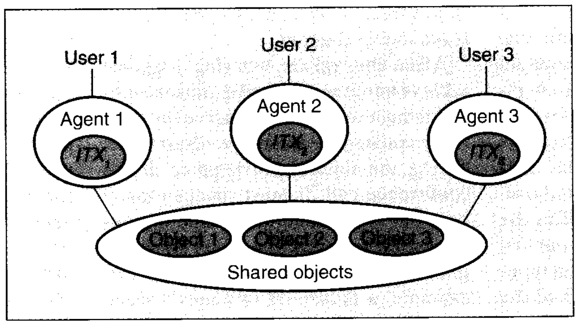


Figure 1. Application perspective of the ITX system.

ITX system overview

An ITX system (shown in Figure 1) consists of a set of user agents associated with the users participating in a cooperative task and a set of shared objects representing data and resource information to be shared by the user agents. The ITX system stores shared objects in one or more databases to use database system facilities such as data definition, manipulation languages, and transaction management. Each *ITX* in the ITX system represents a user participating in a cooperative task. *ITX*s coordinate with each other indirectly by changing and observing changes of shared objects. Since there is no direct communication between *ITX*s, an individual *ITX* can be created and deleted dynamically in the ITX system independent of other *ITX*s.

An *ITX* is specified between begin_*ITX* and end_*ITX* statements in the code that implements a user agent and consists of the following components:

- $\{TX_n\}$: A set of n atomic transactions in the *ITX*. A *TX* can perform operations on multiple shared objects (a *TX* commits if and only if all its operations complete successfully).
- *TX* selection logic: An application-dependent criterion to select *TX*s for execution during an iteration of the *ITX* (described later).
- $\bar{O}$: A set of observation sets O_i (i = 1 to n) maintained by the *ITX*. Each O_i is an observation set of a transaction TX_i in the *ITX*.
- *OBJ*: The local cooperative objective of the *ITX* defined over $\bar{O}$. It might be defined by the user agent or might be assigned to the *ITX* by the ITX system based on the global cooperative objective OBJ_g. Therefore, the local *OBJ* can vary from *ITX* to *ITX*.

An iteration of an *ITX* involves execution of the code from begin_*ITX* to end_*ITX*. During an iteration of an *ITX*, some (not necessarily all) of its *TX*s are executed. The *TX* selection logic determines the *TX*s and their partial order of execution in any iteration, so at the completion of the iteration *OBJ* is satisfied (we discuss *OBJ* later). In performing this task, the selection logic typically uses the local objective and the current observation set described below. We do not specify a language for the *TX* selection logic. Candidates include the host condition statements in the programming language used for coding the user agent, a predicate logic, and a set of intratransaction (intersubtransaction) dependencies used in various advanced transaction models.[10,11] Elsewhere, we show how to define in an ITX system some of the dependencies discussed in the literature on extended transaction models.[12]

Unlike a traditional transaction, an *ITX* does not need to satisfy all of the ACID (atomicity, consistency, isolation, and durability) properties. Usually, it does not satisfy the isolation and atomicity properties. It supports the durability property in the sense that the *TX*s atomically update the shared objects, which are (typically) persistent. The ITX system does not use serializability as its consistency criterion. Instead, it uses the application-independent fixed-point criterion and the application-dependent cooperative objective.

An observation set O_i associated with a TX_i can consist of one or more of three types of observations: *inputs* (states of the shared objects in the read set of the *TX*), *outputs* (states of the shared objects in the write set of the *TX*), and *execution states* (for example, commit or abort status) of the *TX*. A user agent can manipulate the shared objects only through a *TX* of its *ITX*. The system can modify shared objects to record an independent event such as a resource failure. Each *ITX* maintains an observation set. The size of an observation set is application dependent. Later, we discuss how the system uses the observation set to determine whether the ITX system has reached a stable state (fixed point).

When an iteration completes (the end_*ITX* statement is executed), the *ITX* does not terminate. Instead, it monitors the state changes in its observation set so that it can start another iteration immediately when state changes occur that violate the fixed-point criterion.

The OBJ_g of a set of cooperative *ITX*s (and hence the local *OBJ* of each *ITX*) is an application-dependent cooperative objective that an authorized user (or a user agent) specifies explicitly and manipulates dynamically. The *OBJ* of each cooperating *ITX* is usually a subset of OBJ_g relevant to its participation in the conference call (that is, the local objective is derived by projecting the global objective on the users and the resources relevant to the *ITX*). The system accomplishes the OBJ_g of a cooperative task cooperatively by distributing *OBJ* to participating *ITX*s. The *OBJ* of an ith *ITX*, ITX_i, is denoted as OBJ_i. An ITX_i, then, executes its *TX*s to achieve the assigned local objective OBJ_i.

An *OBJ* can consist of multiple-ordered or unordered alternative objectives. (We do not discuss a specific language for defining the objectives.) The *ITX* attempts to reach a system state (the states of the shared objects it observes) that satisfies one of the alternatives specified in the local *OBJ*. This provides flexibility in achieving different alternatives when the environment changes. In the conference call example, OBJ_g specifies the desirable conference configuration consisting of a set of users and their corresponding communication media. When a desired connection medium is not available, the agent can request an alternative connection medium to satisfy the *OBJ*.

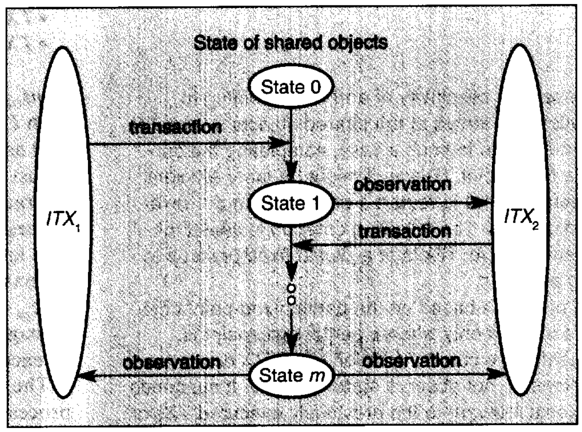


Figure 2. Execution of two ITXs.

Iterative execution control

We base the control of the iterative executions of *ITX*s on a new application-independent criterion called the *fixed-point criterion*. An *ITX* has reached a fixed point if observations resulting from two consecutive iterations are the same. The observation set at the end of each *ITX* iteration represents the states of all relevant objects (resource allocations, cooperative objective) and the execution status of various *TX*s. Thus, no change in the observation set implies a stable environment with respect to the *ITX*.

Figure 2 shows the activities of two *ITX*s. The state changes caused by committed *TX*s or independent system failures are observed by the relevant *ITX*s. Alternatively, an *ITX*'s observations can be affected by other *ITX*s through their changes to the shared objects in its read set.

Fixed-point criterion. Defining a fixed point more precisely, each iteration of an *ITX* consists of

(1) submission of some of its *n* *TX*s to manipulate shared objects based on the observations of the previous iteration and

(2) new observations.

The *m*th iteration of an *ITX*, $\{TX_k^m\}$, involves execution of *k* of its *TX*s ($k \le n$) that are relevant to that iteration. Let $\overline{O}_j^m = \{O_i(m)\}$ for $i = 1$ to j ($j \le k$) denote the state of the observation sets after the execution of TX_j^m in the *m*th iteration of the *ITX*. Let $\overline{O}$ at the end of the *m*th iteration of an *ITX* be denoted by $\overline{O}^m$. Note that $\overline{O}_n^m = \overline{O}^m$. ITX uses *OBJ* and $\overline{O}^m$ to control the execution of *TX*s in the $(m + 1)$th iteration.

Hence we have the definition

An *ITX* is at a fixed point if $\overline{O}^{m-1} = \overline{O}^m$.

An *ITX* in an iteration updates only the observations corresponding to the *TX*s executed in that iteration; other observations remain the same as those in the $\overline{O}$ of the previous iteration.

When an *ITX* reaches a fixed point, it checks to see if its local objective is satisfied by comparing the states of the relevant shared objects with its *OBJ*. The *ITX*'s observation set has the information needed to determine this. An *ITX* can reach a fixed point that does not satisfy its local objective. Such a state, while stable, does not meet the application requirements. In this case, the *ITX* attempts to reach another fixed point that satisfies the cooperative objective by trying alternative *TX*s. Disturbances caused by transient failures or user interventions trigger new iterations of the transactions to reach a new fixed point. This feedback-controlled iterative process provides robust control to meet cooperative objectives.

Each *ITX* reaches a fixed point based on the local *OBJ* and the observation sets that are different from those of other *ITX*s. Consider a conference call among users U_1, U_2, and U_3, with audio connections among all three users and video connections between U_1 and U_2. Let ITX_i be the *ITX* of the agent serving U_i ($i = 1, 2, 3$) and assume that each video connection includes an audio connection. It is possible that the local objective for U_3 is making and maintaining an audio connection to the conference. Therefore, ITX_3 defines an observation set only on the audio connections among all three users. If the video connection of U_1 fails, ITX_1 can switch to an audio connection and change OBJ_g to audio connection among all three users. In this case, the *OBJ* of ITX_2 will change to audio connection also, and it will iterate until it reaches a fixed point, at which the video connection is deallocated and the audio connection is established. However, ITX_3 remains at the fixed point throughout this change since it does not detect any change in the shared objects that affects its *OBJ* and its observations.

We can extend the definition of fixed point to accommodate observation errors. Furthermore, in some cases two consecutive observations do not have to be exactly the same to satisfy the fixed-point criterion, as long as the difference is within an acceptable limit. The fixed-point-based strategy of the ITX system can be extended to better react to the changes (see the sidebar).

Example of a teleconferencing application

A simplified multimedia teleconferencing application demonstrates the ITX system's basic concepts and advantages. We define the shared objects used in the example, define the *ITX*s, and discuss the iterative execution of the *ITX*s to support the application.

Definition of the shared objects. A more complex teleconferencing application would require shared-object structures more complex than we present here. However, we can model an *n* user conference configuration using a shared object type $Conf_{SO}$ defined as follows:

Adaptive control

An adaptive strategy increases the sensitivity of an *ITX* in reacting to changes. During an *ITX* iteration, the states of the shared objects relevant to the already executed *TX*s can change. In such a case, completing the current iteration will not lead to a fixed point. Hence, it might be more efficient not to complete the current iteration, but to restart a new iteration or reexecute previously executed transactions. The following criterion is useful for defining an adaptive control strategy: An *ITX* is at a *partial fixed point* up to TX_j^m if $\bar{O}_i^{m-1} = \bar{O}_i^m$ for $i = 1$ to j and $j < n$.

Adaptive execution control of an *ITX* based on the partial fixed-point criterion requires that a TX_{j+1}^m be executed only when a partial fixed point is reached up to TX_j^m. Consequently, the current iteration of an *ITX* continues only if the preceding transactions do not observe state changes. It suspends if the states of the shared objects relevant to the previously executed *TX*s of the current iteration have changed. Optionally, we can also specify that the *ITX* should reach a fixed point within time T_{out} to provide a time-out (for example, to guarantee the termination of an *ITX*).

Using the adaptive execution control strategy requires a partial fixed-point detection mechanism as well as a reexecution strategy. One detection method is to treat each *TX* as an *active* process that continues to monitor the states of the relevant shared objects even after it is executed in an iteration and restarts its execution after stopping the progress of the current partially completed iteration. This might require controlling execution of multiple affected *TX*s. Our prototype implementation uses a simpler alternative: The *ITX* continues to observe the states of the objects relevant to already executed *TX*s in its observation set. If it detects a change in such objects before the iteration completes, it aborts the execution of the currently executing *TX*(s) and reexecutes some or all of the previously executed *TX*s using the *TX* selection logic.

$Conf_{SO} = (User_i, Connection_i)$,
where $i = 1, n$
$User_i = U_i$ if U_i is connected; null otherwise.
$Connection$ = video (audio) if video (audio) link and display are allocated; null, otherwise.

$Conf_{SO}$ is the basic object class used to create the (global) conference objective $Conf_{OBJ_g}$ (OBJ_g of the conference application) and the current state of the conference configuration $Conf_{state}$. Consider the following $Conf_{OBJ_g}$ for a conference among three users U_1, U_2, and U_3:

$Conf_{OBJ}$:: U_1 video
U_2 video
U_3 audio

This cooperative objective specifies that users U_1 and U_2 are connected to the conference with video connections and user U_3 is connected to the conference only via audio. Video connections also include the audio connections so that U_3 can talk to the other conference group members. We use a circle to represent a conference bridge system. A conference bridge system consists of multiple video input signals and displays them to users connected to the bridge. It also selects one of the many audio signals and outputs the signal to multiple speakers. The system allocates a bridge for each conference. In the following example, we assume that an *ITX* participates in a conference by connecting a link of the appropriate media type between a user and the bridge allocated for the conference.

Definition of the *ITX*s. The user agent serving each user U_i consists of an ITX_i. Each *ITX* consists of a set of *TX*s, and each *TX* consists of a set of operations atomically performed to manipulate shared objects. We define the following *TX*s for our simplified application:

- TX_0: Create $Conf_{OBJ_g}$ and $Conf_{state}$.
- TX_1: Read $Conf_{OBJ_g}$ and $Conf_{state}$.
- TX_2: Allocate specified resources such as a video display and a video communication link.
- TX_3: Deallocate specified resources.
- TX_4: Update $Conf_{state}$.
- TX_5: Update $Conf_{OBJ_g}$.

$Conf_{OBJ_g}$ and $Conf_{state}$, read by TX_1 of each *ITX*, update its local objective as well as its observation set.

Just as a user agent uses an *ITX* to interact with shared objects, a system process not directly serving a user can also use an *ITX* to manipulate shared objects. Our example has two *ITX*s — ITX_{rm} and ITX_{sys} — that are used to manage resources and initialize a conference, respectively.

The system resource management process uses ITX_{rm} to ensure that the shared object $Conf_{state}$ is consistent with the actual physical states of the resources used in the conference. For example, ITX_{rm} might observe the status of a video link either by trapping the asynchronous messages from the physical resources or by polling their flags. When a hardware failure or a preemption (an allocated resource is reallocated for other uses) occurs on the allocated video link, ITX_{rm} will change the video attribute of $Conf_{state}$ to failure.

The system initialization process uses ITX_{sys} to activate *ITX*s of the user agents corresponding to the *User* attribute of the newly created cooperative objective. ITX_{sys} also identifies the relevant shared objects ($Conf_{OBJ_g}$ and $Conf_{state}$) to the activated ITXs.

Figure 3 shows the interactions between ITX_2 and ITX_3 in setting up the conference. ITX_1 initiates a conference by creating and initializing shared objects $Conf_{OBJ_g}$ and $Conf_{state}$ using its TX_0. Then ITX_{sys} activates ITX_2 and ITX_3 and identifies to them the shared objects (used for this conference call) and initiates their first iteration.

Iterative execution of *ITX*s. An iteration of a cooperative ITX_i that supports our application for user U_i uses the *TX*s (assuming the implied *TX* selection logic) as follows. Each ITX_i first performs TX_1. Thus, each *ITX* reads $Conf_{OBJ_g}$ and $Conf_{state}$. Each ITX_i projects a part of $Conf_{OBJ_g}$ into a local objective $Conf_{OBJ_i}$. For example, as Figure 3 shows, $Conf_{OBJ_3}$ is equal to (U_3, audio). Next, each *ITX* determines which resources it needs to allocate or deallocate to meet the $Conf_{OBJ_i}$ or to minimize the difference between the $Conf_{OBJ_i}$ and $Conf_{state}$, and issues TX_2s and TX_3s accordingly. (The difference between $Conf_{OBJ}$ and $Conf_{state}$, called a distance function, is application dependent.) For example, in Figure 3, ITX_2 uses its TX_2 and TX_3 to allocate and deallocate all resources for a video connection (video and audio link, bridge port, display, camera, microphone, and speaker), and ITX_3 uses its TX_2 and TX_3 to allocate and deallocate all resources for an audio connection (audio link, bridge port, microphone, and speaker). If any TX_2 (used to allocate a resource) fails, the ITX_i might determine alternative resources and issue additional TX_2s, or suggest a different conference configuration by executing a TX_5. Finally, before ending the iteration, based on the successful TX_2s and TX_3s, the ITX issues a TX_4 to update the current conference configuration $Conf_{state}$.

As the conference progresses, the execution of ITXs changes $Conf_{state}$. $Conf_{OBJ_g}$ itself might also change because of user interventions and failures. Figure 4 shows the state changes during the conference.

After initialization, each participating ITX_i obtains a local conference objective $Conf_{OBJ_i}$ (State 1 of Figure 4) from a global cooperative objective $Conf_{OBJ_g}$. In normal cases, if all the resources are available and allocated successfully, the conference is established in one iteration. State 2 of Figure 4 shows $Conf_{state}$ after ITX_1, ITX_2, and ITX_3 successfully allocate the resources and complete their first iteration. In the next iteration, if no failure or user intervention occurs, the ITXs will obtain the same observation set from TX_1s and reach a fixed point. At the fixed point, the ITX waits to start a new iteration in response to either an external event from the shared objects or the next polling to the shared objects that indicates a change.

If the video link is not available for user U_2, TX_2 of ITX_2 will try to allocate an audio connection as an alternative media type. Next, ITX_2 uses its TX_4 and TX_5 to update $Conf_{state}$ and $Conf_{OBJ_g}$, respectively. In the meantime, ITX_1 and ITX_3 might have established the required video and audio media types and updated $Conf_{state}$ in one iteration (State 3 of Figure 4). These ITXs remain active to detect changes in their observation sets, $Conf_{OBJ_g}$ and $Conf_{state}$. The update operation on $Conf_{state}$ from ITX_2 violates the fixed-point criterion on ITX_1 and ITX_3 because it changes the states of their observation sets. At this point, both ITX_2 and ITX_3 have only audio capability. Thus, ITX_1 changes its media type to audio and updates $Conf_{state}$ and $Conf_{OBJ_g}$ (State 4). A new fixed point different from the original objective is reached. Later on, if U_2 can allocate a video resource, $Conf_{OBJ_g}$ can change so that U_1 and U_2 can install the video connection in the next iteration (State 2).

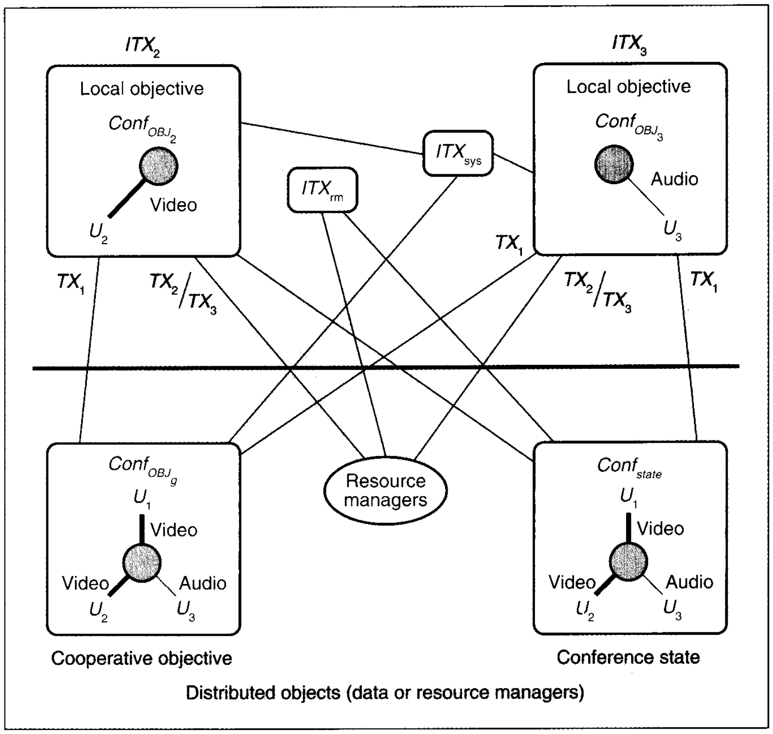


Figure 3. The ITX system for the example application.

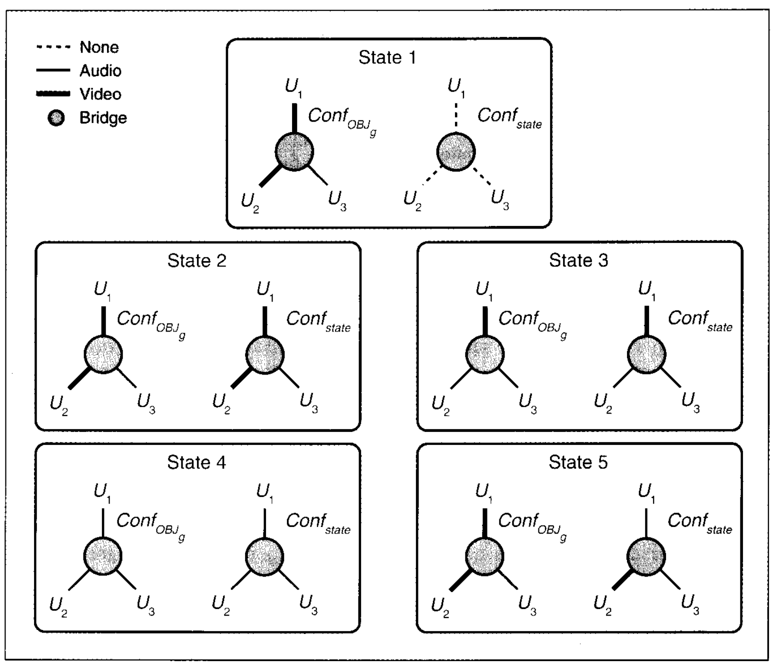


Figure 4. Snapshots of conference states.

Iterations also determine low-level parameters for a connection after a media type is determined. For example, video and audio connections might have various bandwidths. Higher bandwidth provides better video quality and faster data service at a higher cost. Suppose two users want to establish a video connection. One user prefers higher quality to lower cost, but the other prefers lower cost to higher quality. Establishing an agreeable connection becomes a negotiation.[5,6] The arbitration between conflicting objectives can be based on the *ITX* logic or can involve interactions with the users. In our example, multiple iterations of proposals and counterproposals might be required to determine a final bandwidth.

ITX system features

Several important features of the ITX system provide robust and flexible control to support complex applications.

Specifying complex objectives. Consider a slightly more complex conference service involving four participants requested by participant U_1. The conference service is first created when U_1 creates two shared objects $Conf_{OBJ}$ and $Conf_{state}$ of the type $Conf_{SO}$. Suppose that U_1 would like to create two sessions of conferences with two groups of users by specifying two objectives in sequence as follows:

$$Conf_{OBJ_g} = \{Conf_{OBJ_1}; Conf_{OBJ_2}\} \text{ where}$$
$$Conf_{OBJ_1} = \{(U_1, \text{video}), (U_2, \text{video}), (U_3, \text{audio})\}$$
$$Conf_{OBJ_2} = \{(U_1, \text{video}), (U_2, \text{video}), (U_4, \text{audio})\} \text{ or } \{(U_1, \text{audio}), (U_3, \text{audio}), (U_4, \text{audio})\}$$

The *ITX*s first try to reach the cooperative objective $\{(U_1, \text{video}), (U_2, \text{video}), (U_3, \text{audio})\}$. Then they try to reach one of the disjunctive terms of the cooperative objective: $\{(U_1, \text{video}), (U_2, \text{video}), (U_4, \text{audio})\}$ or $\{(U_1, \text{audio}), (U_3, \text{audio}), (U_4, \text{audio})\}$.

A disjunctive objective such as the one in this example allows the conference to switch between two acceptable cooperative objectives and adapt to changes in environment or user participation. If U_3 drops out, U_2 can join in the conference and vice versa. In addition, with the proper privilege, a user can change the cooperative objective dynamically so that participating *ITX*s can work on alternative conference configurations. In this example, we assume that all the *ITX*s agree to use the same $Conf_{OBJ}$ as the initial local objective. In other cases, it might be necessary to negotiate the cooperative objective before each user agent starts to work on it. The negotiation process, however, is similar to the process for establishing the desired conference configuration.[6]

Dynamic additions of new agents. The ITX system framework can support dynamic addition (or deletion) of agents or *ITX*s without recompiling, linking, or restarting *ITX*s that are currently running. This is possible because *ITX*s communicate over shared objects and are notified if needed. To add an *ITX* to a cooperative task, we only need to define the observation set of an *ITX* over a set of shared objects. After that, an *ITX* can react to all changes in the shared objects.

This feature is useful for real-time conferencing systems. The addition or deletion of user agents does not disturb the ongoing conference. In addition, tests with our prototype show we can dynamically add administrative agents to observe the cooperative work for billing or debugging purposes without affecting the ongoing application.

Dealing with user interventions. The ITX system can also accommodate unpredictable user interventions. At any moment during the conference call, any authorized user can decide to change the conference configuration $Conf_{OBJ_g}$. Suppose user U_2 must leave to attend a meeting and would like to record the remainder of the conference. To do that, ITX_2 of U_2 changes the tuple (U_2, video) of $Conf_{OBJ_g}$ to (recorder, video). All the other *ITX*s detect the changes in $Conf_{OBJ_g}$ and start a new iteration. If U_1 and U_3 accept the substitution of U_2 by a recorder, they will not further change $Conf_{OBJ_g}$. If U_1 and U_3 do not wish to be recorded, they change $Conf_{OBJ_g}$ back to the original configuration; that is, they change (recorder, video) back to (U_2, video). In this case, U_2 will detect changes in $Conf_{OBJ_g}$, notice the disagreement of other agents, and consider other alternatives. Each *ITX* needs to consider appropriate logic for performing such a negotiation in defining the *TX* selection logic.

Dealing with transient failures. An ITX system conference call is robust to transient failures. Assume that resource failures during the conference are detected by either a separate process that polls the status of the resources repetitively or a transaction in the *ITX* with an observation set defined over the resources' status. After a failure is detected, $Conf_{state}$ changes.

For example, suppose that after a conference has been established successfully according to the first conference objective (State 2 of Figure 4), user U_1's video display fails. A system process detects this failure and changes (U_1, video) of $Conf_{state}$ to (U_1, audio) (State 5 of Figure 4). Since $Conf_{state}$ is one of the observation sets of each *ITX*, any change in $Conf_{state}$ will lead to violation of the fixed-point criterion and, thus, new iterations. In this case, ITX_1's local objective is changed to have an audio connection. ITX_1 then allocates the audio medium to achieve a graceful degradation. Subsequently, ITX_2 observes that all the other participants have only audio connections. To eliminate the cost of an unused video connection, ITX_2 switches to audio (State 4 of Figure 4). The ITX system handles failures and user interventions (changing the cooperative objective) via the same iterative feedback control paradigm.

Enforcing constraints on resources. The ITX system also provides a convenient way to enforce constraints on system resources. For example, a video connection requires a video link, a video display, and a camera. When one video-connection resource becomes unavailable (because of a failure or a preemption), other video-related resources (for example, displays) should also be deallocated automatically to save system resources. To achieve that, a separate *ITX* for resource management is created, ITX_{rm}, which enforces the resource relationships. The ITX_{rm} observes the status of the video link and video display either by polling or by receiving signals from the physical resources (devices). After observing failures, the ITX_{rm} executes user-defined transactions to enforce the constraints among the resources. This ITX_{rm} is a customized constraint enforcer written by application or system programmers.

Crash recovery. The ITX system provides robust crash recovery for cooperative agents. When an *ITX* crash occurs, the cooperative work can continue after crash recovery because all the information required for a cooperative task is reliably stored in shared objects that are updated using atomic transactions. Since the *ITX*s do not communicate directly, a crashed *ITX* will not affect other *ITX*s. A node crash, on the other hand, might cause multiple *ITX*s to crash and make shared objects unavailable. Since the operations issued by *ITX*s are transactional, a node crash can result in losing only partially executed operations. The states of the shared objects before the crash can be recovered to the states after the commitment of the previous transaction.

Reliable communication. A shared object of type mailbox provides message queues for reliable communication among *ITX*s. The transaction facility permits atomic write operations to multiple mailboxes. It can also guarantee that messages delivered via transactions are serialized among each other. As a result, the states of the mailboxes are guaranteed to be consistent, messages will not be delivered out of order, and no message will be partially delivered. Using the ITX system, the protocol designer can concentrate on solving the problem itself rather than on the mechanisms for reliable communications.

The objective of our research with the ITX system is to develop a framework for supporting applications that involve multiple cooperating users, resources managed by heterogeneous and autonomous systems, and changing cooperative objectives and resource failures. Distributed cooperating tasks with cooperating agents and shared objects provide a natural model to support such applications. The ITX system proposed in this article addresses the issue of controlling the interactions among the cooperating agents.

Shared objects represent resources and their status as well as control information. Control information includes the global cooperative objective of the application. Providing a uniform interface to shared objects reduces the need for pairwise communication. Shared objects can be made persistent to preserve the context of cooperation, while agents are dynamically created and deleted. We model operations of each agent on shared objects as an *ITX*.

An *ITX* consists of a set of atomic transactions and the logic that decides which transaction to execute, based on an explicitly defined cooperative objective and the current state of the system known to the *ITX*. *ITX*s execute transactions iteratively until a fixed point is reached. The fixed-point criterion defines the stable state of the system as the application-independent correctness criterion for controlling interactions among agents. Transactions executed during an *ITX* iteration change the shared object's state to a state closer to the cooperative objective the *ITX* needs to achieve.

Changes in the cooperative objective or failures of resources also result in changes to the shared objects' states. *ITX*s observe such changes as violations of the fixed-point criterion and then execute their transactions to reach another fixed point that satisfies their respective objectives.

We presented the example of a simplified multimedia teleconferencing application to demonstrate the features of the ITX system.

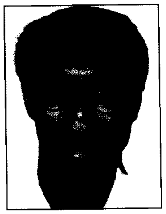
We have implemented a software prototype of the ITX system in a distributed workstation environment. An object-oriented database-management system manages the shared objects. The agents are programmed in C++ and invoke database transactions. The prototype has helped us understand issues in *ITX* design such as feedback control principles, transaction management, failure modeling, and real-time performance. We focused on control and coordination, so we have not fully addressed other issues related to supporting specific applications. These issues include heterogeneous resources (media management), multimedia presentation, and various heterogeneity and autonomy implications of a complex telecommunications environment. ■

Acknowledgments

We thank Nancy Griffeth, Gomer Thomas, and other colleagues for their valuable comments that helped us to improve the article. We also thank Mike Yu for the design and implementation of the prototype system.

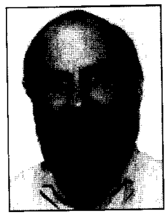
References

1. W.F. Leung et al., "A Software Architecture for Workstations Supporting Multimedia Conferencing in Packet Switching Networks," *IEEE J. Selected Areas Comm.*, Vol. 8, No. 3, Apr. 1990, pp. 380-390.
2. J. Grudin, "CSCW Introduction," *Comm. ACM*, Vol. 34, No. 12, Dec. 1991, pp. 30-34.
3. S.R. Ahuja, J.R. Ensor, and D.N. Horn, "The Rapport Multimedia Conferencing System," *Proc. ACM Conf. Office Information Systems*, ACM Press, New York, 1988, pp. 1-8.
4. I. Greif and S. Sarin, "Data Sharing in Group Work," in *Computer-Supported Cooperative Work*, I. Greif, ed., Morgan Kaufmann, San Mateo, Calif., 1988, pp. 479-508.
5. H.M. Vin et al., "Multimedia Conferencing in the Etherphone Environment," *Computer*, Vol. 24, No. 10, Oct. 1991, pp. 69-79.
6. N. Griffeth and D. Velthuijsen, "The Negotiating Agent Model for Rapid Feature Development," *Proc. Eighth Int'l Conf. Software Eng. for Telecommunications Systems and Services*, IEE, London, 1992, pp. 67-71.
7. S. Minzer, "Signaling and Control for Multimedia Services," *Proc. IEEE Multimedia 90*, CNET, Lannion, UK, 1990, p. 6.
8. G. Gopal, G. Herman, and M.P. Vecchi, "The Touring Machine Project: Toward a Public Network Platform for Multimedia Applications," *Proc. Eighth Int'l Conf. Software Eng. for Telecommunications Systems and Services*, IEE, London, 1992, pp. 27-31.
9. B.C. Kuo, *Digital Control Systems*, Holt, Rinehart, and Winston, New York, 1980.
10. A. Elmagarmid, ed., *Database Transaction Models for Advanced Applications*, Morgan Kaufmann, San Mateo, Calif., 1992.
11. D. McCarthy and U. Dayal, "The Architecture of an Active Data Base Management System," *Proc. ACM SIGMod*, ACM Press, New York, 1989, pp. 215-224.
12. K.C. Lee, W. Mansfield, and A. Sheth, "An Interactive Transaction Control System for Distributed Cooperative Work," *IEEE Data Eng. Bull.*, Vol. 14, No. 1, Mar. 1991.



Kuo-Chu Lee has been a member of the technical staff in the applied research area of Bell Communication Research (Bellcore) since 1985. His research interests are systems architecture, prototyping, and software engineering with applications in the areas of multimedia communication systems, enterprise information networking systems, parallel and distributed database systems, and telecommunications systems.

Lee received his MS in computer engineering from Ohio State University in 1983 and his PhD in computer engineering from Rutgers University in 1990. He holds three US patents. Lee is a member of the IEEE Computer Society.



William H. Mansfield Jr. joined Bellcore at its inception in 1984 following 11 years with Bell Laboratories. He directs research focused on network control and scaling issues associated with database access for future network services. He has worked in systems performance analysis and application modeling, and has directed the assessment of various emerging computing technologies for a technology transfer organization.

Mansfield holds a BS and an MS in computer science and is a member of the IEEE and the IEEE Computer Society.

Amit P. Sheth is working on multidatabase systems issues of managing interdependent data and work-flow management at Bellcore, as well as developing a corporate heterogeneous information management environment. He has led projects to develop a heterogeneous distributed database system, integrate artificial intelligence systems with database systems, and develop tools for schema integration and view update.

Sheth was editor of the December 1991 *SIGMod Records* special issue on semantic issues in multidatabase systems and general chair of the First International Conference on Parallel and Distributed Systems. He is an ACM lecturer and a program co-chair of the Third RIDE (Research Issues in Data Engineering) Workshop on Interoperability in Multidatabase Systems. He is a member of the IEEE Computer Society.

Readers can contact Sheth at Bellcore, RRC-1J210, 444 Hoes Lane, Piscataway, NJ 08854, e-mail amit@ctt.bellcore.com; or Lee and Mansfield at Bellcore, 445 South Street, Morristown, NJ 07960; e-mail: (kasey, whm)@bellcore.com.

CALL FOR PAPERS

PERFORMANCE EVALUATION OF PARALLEL SYSTEMS

University of Warwick - November 29 - 30, 1993

THE WORKSHOP

The first annual workshop on the Performance Evaluation and Assessment of parallel computers, sponsored by the European Community Esprit Programme, the British Computer Society and the TTCP XTP3 Technical Panel on Computer Architectures.

This workshop is intended to be a forum to exchange information about innovative efforts and experience concerning the use and the choice of a parallel computer.

TECHNICAL PROGRAM

Submission of papers is invited in the following areas of High Performance Computing:

- Benchmarking (initiatives, experiences, limits, substitution techniques)
- (Computer and/or application) Modelling
- Monitoring
- Characterisation
- Market trends and needs
- Future concepts (in technology, architectures, software)
- Current applications of Massively Parallel Systems Global Performance Evaluation

PARTICIPATION

All interested parties are invited to attend and are encouraged to present issues and experience on any of the above areas. Participation from government, academia, industry personnel involved in research, program management, project engineering are welcome.

RESPONSE DATES

Paper due: 1 August 1993
Notification of acceptance: 31 August, 1993
NOTE: Full paper is requested for submission, and should include the following information:
(1) Application area(s) as categorised above. (2) Name, address, telephone number, affiliation of all authors. (3) A 200 word abstract with the full paper. Full paper is limited to 8 single-sided pages
Submissions should be mailed to the **Program Chair** given below. Selected papers will be included in a published text following the workshop.

PROGRAM CHAIR

Graham NUDD, Department of Computer Science, University of Warwick, Coventry, CV4 7AL, UK
e-mail: conf@dcs.warwick.ac.uk
Phone: +44 203 523668, Fax: +44 203 525714