A Formal Basis for Structured Multimedia Collaborations *

Sreeranga Rajan†, P. Venkat Rangan‡, and Harrick M. Vin◇

† Computer Science Laboratory, EK-237,
SRI International, Menlo Park, CA 94025, E-mail: sree@csl.sri.com
‡ Multimedia Laboratory, Department of Computer Science and Engineering,
University of California at San Diego, La Jolla, CA 92093-0114, E-mail: venkat@cs.ucsd.edu
◇ Computer Sciences, TAY 2.124,
University of Texas, Austin, TX 78712-1188, E-mail: vin@cs.utexas.edu

Abstract

In this paper, a formal basis for multimedia collaborations is proposed. The formal basis establishes a precise mathematical framework for supporting a wide spectrum of structured collaborative tasks, ranging from simple meetings and conferences to classrooms and examinations, and from corporate negotiations, work flow tasks, and team design endeavors to courtroom hearings. We mechanize three levels of abstraction: **streams** at the lowest level, for media communication, *sessions* at the next level, which represent collections of semantically related media streams, and *conferences*, which represent temporally related sequences of sessions. The mathematical framework has been mechanized in Prototype Verification System (PVS) from SRI International. The mechanization allows us to experiment with different formal models, and reason about properties of multimedia collaborations efficiently. The mechanized framework supports unification of both interactive and non-interactive collaborations, sophisticated access control and intra-group and inter-group collaborations, yielding a powerful set of building blocks for constructing multimedia applications. A formal specification of collaborations can form the basis of a transformational approach to automatically synthesize low-level, network environment specific constraints, necessary for managing media communication in multimedia collaborations. A prototype collaboration management system based on the framework has been implemented at the UCSD Multimedia Laboratory.

*To appear in the Proceedings of the 2nd IEEE International Conference on Multimedia Computing and Systems, Washington, D.C., May 1995

1 Introduction

In this paper, we provide a formal basis for reasoning about a wide range of structured collaborations in a mechanical verification system. Structured collaborations among individuals range from simple meetings to conventions, from lectures to classrooms and examinations, and from corporate negotiations, work flow tasks, and team design endeavors to courtroom hearings, each with its own unique set of requirements for media communication [24, 27]. For instance, a convention usually involves a sequence of presentations by its speakers and questions posed by its attendees, each of which must be permitted by its chairperson. A classroom is similar, except that it has only one speaker who also serves as its chairperson. An examination, on the other hand, may involve students and a professor, with communication restricted to that between the professor and each of the students, but absolutely no communication among the students themselves. Negotiations among corporate groups may involve both intra-group and inter-group communication, which may need to be separated for purposes of confidentiality [28].

Technological advances such as virtual reality, which promise to provide life-like imagery using stereovision and spatially distributed voice using stereophonic audio, coupled with the availability of high speed networks, are expected to transform computer systems from being a mere data processing unit into a highly effective means for carrying out multimedia collaborations [4, 11]. Media communication among the participants of a structured collaboration can be either interactive (as in a telephone conversation), or non-interactive (as in electronic mail) [29]. A formal model of collaborations could form the basis of the process to automatically synthesize low-level, network environment specific constraints, necessary for managing media communication in multimedia collaborations.

Whereas interactive collaborations involve contemporaneous exchange of media information, non-interactive collaborations are characterized by recipients' discretionary delays after transmission by senders. During such intervening delays, the media information in transit may have to be managed by servers (such as storage and mail servers). Supporting a wide range of structured collaborations, both interactive and non-interactive, necessitates a rich framework for characterizing the semantics of collaborations and mechanisms for their instantiation. In an earlier paper [29], we proposed a characterization of the semantics of structured collaborations. In this paper, we extend the characterization in order to provide a mechanical reasoning framework for mechanisms involved in the establishment of structured collaborations and subsequent management of their progress

1.1 Related Work

Mechanisms for person-to-person conferences and shared text-oriented workspaces have been the focus of research in many multimedia projects [5, 12]. Lantz [16] proposes a text/graphics teleconferencing architecture whose goal is to fit into existing systems with minimal impact, and Sarin and Greif [26] have studied teleconferencing architectures for text and graphics, but have not explored time dependent media such as audio and video. Dewan [10] has addressed the issue of sharing and coupling among windows displaying shared workspaces. Casner et al. at ISI [6] have developed media transport protocols for teleconferencing among geographically distributed sites in wide area packet network, but do not address issues of flexible capabilities for user participation. Ahuja et al. at Bell Laboratories [2] and Aguilar et al. at SRI [1] have proposed architectures for multimedia conferencing, and the Etherphone system at Xerox PARC [31] has explored workstation-controlled telephony.

A model for coordinating concurrent development has been worked out by Harrison et al. [14]. However, their model is restricted to concurrent update of text files. Holt [15] has presented a general framework for expressing coordination requirements in an organized work arena, but does not address the requirements of digital multimedia on high speed networks. Danielsen et al. [9], and Prinz and Speth [21] have described group communication models for computer-based communications environments. However, features such as independent access control of different media streams, and nested collaborations have not been fully explored. In fact, the emphasis in much of the above work is on person-to-person teleconferencing and not on general models for supporting a wide range of structured multimedia collaborations. Greenberg [13, 25] has presented techniques for building personalizable groupware applications in which groups are permitted to tailor the collaboration semantics to their own needs. Their approach is closest to the formal model for structured collaboration presented in this paper.

1.2 Contributions of this Paper

The main contribution of this paper is that it provides a formal basis for structured multimedia collaborations in a mechanical reasoning system. We formalize a taxonomy of collaborations, and develop a formal model for capturing the requirements of diverse types of structured collaborations. The formal model is based on our earlier work [29], where we presented hierarchy of three abstractions. At the lowest level, are *streams* for media communication modulated by access rights of participants within collaborations. The higher two levels of abstractions are *sessions*, which represent collections of semantically related media streams, and *conferences*, which represent temporally related sequences of sessions. These abstractions support unification of both interactive and non-interactive collaborations, sophisticated access control, and intra-group and inter-group collaborations, thereby yielding a rich environment for carrying out multimedia collaborations. A prototype collaboration management system based on the model has been implemented at the UCSD Multimedia Laboratory.

The rest of the paper is organized as follows. The taxonomy is presented in Section 2. A brief overview of Prototype Verification System (PVS) [20, 19], a mechanical reasoning system from SRI International is presented in Section 3. A discussion of the hierarchy of abstractions appears in Section 4, and the operations that they support appears in Section 5. Finally, Section 7 summarizes our results.

2 Taxonomy of Multimedia Collaborations

Generally, interactions involving communication among multiple participants are referred to as collaborations. There are several different types of collaborations, which can be classified as follows:

Simple Collaborations contain only *simple* participants, i.e., individual users. **Nested Collaborations** contain at least one *super* participant, i.e., another collaboration.

Interactive Collaborations involve contemporaneous exchange of media information among their participants. **Non-Interactive Collaborations** can be characterized by recipients' discretionary delays after transmission by senders. During such intervening delays, the media information in transit may have to be managed by servers (such as, storage and mail servers).

Static Collaborations do not permit changes in the access rights of their participants after initiation. **Dynamic Collaborations** permit access rights of their participants to be changed at any time.

Symmetric Collaborations require all their participants to possess the same set of access rights. **Asymmetric Collaborations** permit each participant to possess different access rights.

Homogeneous Collaborations do not allow conversion between different media during communication between participants. **Heterogeneous Collaborations** allow conversions between media types during communication between participants. For instance, to enable a user without audio capability to participate in an on-going audio conference, heterogeneous collaborations may employ voice recognizers and synthesizers to translate speech to text and vice versa.

Transient Collaborations are usually short-lived, and their existence depends on the duration of participation of each of their participants. In the simplest case, a collaboration terminates when there are less than two participants (e.g., a telephone conversation). A more involved semantics may require that a collaboration should terminate or change when a key participant leaves the collaboration (e.g., a classroom lecture may transform into an informal meeting once the professor leaves). **Persistent Collaborations** have a much longer duration of existence, and can continue to exist irrespective of participants. Such collaborations usually consist of a sequence of activities separated by periods of inactivity among participants, and hence, can naturally model periodic events. For instance, a weekly meeting can be modeled as a persistent collaboration which is activated every week at a scheduled time.

3 Prototype Verification System

The Prototype Verification System (PVS) [20, 19] is an environment for specifying entities such as hardware/software models and algorithms, and verifying properties associated with the entities. An entity is usually specified by asserting a small number of general properties that are known to be true. These known properties are then used to derive other desired properties. The process of verification involves checking relationships that are supposed to hold among entities. The checking is done by comparing the specified properties of the entities.

PVS has been used for reasoning in many domains, such as in hardware verification [7, 8, 22], protocol verification, and algorithm verification [18, 19]. We briefly give the features of the PVS specification language in Section 3.1, the PVS verification features in Section 3.2 and some notes on the syntax of the PVS specification language in Section 3.3.

3.1 PVS Specification Language

The specification language features common programming language constructs such as arrays, functions, and records. It has built-in types for reals, integers, naturals, and lists. A type is interpreted as a set of values. One can introduce new types by explicitly defining the set of values, or indicating the set of values, by providing properties that have to be satisfied by the values. The language also allows hierarchical structuring of specifications. Besides other features, it permits overloading of operators, as in some programming languages. The PVS specification language allows embedding of special logics. In particular, property specifications can be expressed in temporal logics specialized for specific hardware/software/real-time applications.

3.2 PVS Verification Features

The PVS verifier is used to determine if the desired properties hold in the specification of the model. The user interacts with the verifier by a small set of commands. The verifier contains procedures for boolean reasoning, arithmetic and (conditional) rewriting. In particular, binary decision diagram (BDD) based state-space exploration may be invoked for model-checking and Boolean reasoning. It also features a variety of general induction schemes to tackle large-scale verification. Moreover, different verification schemes can be combined into general-purpose strategies for similar classes of problems, such as verification of microprocessors [7, 8].

A PVS specification is first parsed and type-checked. At this stage, the type of every term in the specification is unambiguously known. The verification is done in the following style: we start with the property to be checked and repeatedly apply rules on the property. Every such rule application is meant to obtain another property that is simpler to check. The property holds if such a series of applications of rules eventually leads to a property that is already known to hold.

3.3 Notes on Specification Notation

In PVS specifications an entity followed by a colon and a type indicates that the entity is a constant belonging to that type. If the colon is followed by the key word *VAR* and a type, then the object is a variable belonging to that type.
For example,

```
x: integer
y: VAR integer
```

describes x as a constant of type integer, and y as a variable of type integer

Sets are denoted by {...}: they can be introduced by explicitly defining the elements of the set, or implicitly by a characteristic function.
For example,

```
0,1,2
x: integer | even(x) AND x /= 2
```

The symbol | has to be read as *such that*, and the symbol `/=` stands for *not equal to* in general. Thus, the latter example above should be read as "set of all integers x, such that x is an even number and x is not equal to 2".

New types are introduced by a key word *TYPE* followed by its description as a set of values. If the key word *TYPE* is not followed by any description, then it is taken as an unspecified type.

Some illustrations are:

```
even_time: TYPE = {x: natural| even(x)}
unspecified_type: TYPE
```

One kind of type that is used widely in this work is the *record type*. A record type is like the *struct* type in the C programming language. It is used to package objects of different types in one type. We can then treat an object of such a type as one single object externally, but with an internal structure corresponding to the various fields in the record.

The following operators have their corresponding meanings:

```
FORALL x: p(x)
```

means *for every* x, predicate p(x) is *true*

```
EXISTS x: p(x)
```

means *for at least a single* x, predicate p(x) is *true*

We can impose constraints on the set of values for variables inside `FORALL` and `EXISTS` as in the following example:

```
FORALL x, (y| y = 3*x): p(x,y)
```

which should be read as
for every x and y such that y is 3 times x, p(x,y) is *true*.

A property that is already known to hold without checking is labeled by a name followed by a colon and the keyword `AXIOM`. A property that is checked using the rules available in the verifier is labeled by a name followed by a colon and the keyword `THEOREM`. The text followed by a `%` in any line is a comment in PVS.
We illustrate the syntax as follows:

```
ax1: AXIOM  % This is a simple axiom
FORALL (x:nat): even(x) = x divisible_by 2

th1: THEOREM  % This is a simple theorem
FORALL (x:nat):  prime(x) AND x /= 2 IMPLIES
         NOT even(x)
```

We also use the terms *axiom* and *theorem* in our own explanation with the same meanings. A *proof* is a sequence of steps that leads to a *theorem*.

4 A Formal Model for Structured Multimedia Collaborations

In order to derive some intuition for developing a formal model for capturing the semantics of such a wide range of structured collaborations, let us examine some common collaborative scenarios: In a group discussion, all the participants are considered equal, and hence, possess the same set of access rights. Lectures and examinations, on the other hand, involve two types of participants (namely, students and a professor). Whereas during lectures, both the students and the professor can freely exchange media information, during the examination, students are disallowed from exchanging media information directly. In both of these scenarios, only the professor is generally permitted to write onto the electronic blackboard.

Consequently, at the most basic level, a model for capturing the requirements of such a collaborative environment must completely characterize the transport of media information between participants. Media transport can be built upon either 1-to-1, 1-to-n, m-to-1, or m-to-n communication paradigms. However, the chosen paradigm will have to be implemented on the underlying network, and must, hence, be designed so as to make use of network services efficiently. Since most networks provide multicast services, the abstraction at this lowest level is a communication channel from a set of senders to a set of receivers (i.e., n-to-n communication), which we will call as a *stream*. The definition of each stream encapsulates the permitted access rights of both the sender and the receiver with respect to the media information transmitted on the stream. For instance, during an examination, since direct communication among the students is prohibited, the receiver list for each of the streams originating from each of the students contains only the professor. On the other hand, the receiver list for the media information originating from each participant in a group discussion contains all of its participants.

A stream, however, is a very rudimentary form of communication. A collaborative activity, such as a lecture or an examination, typically consists of a collection of streams carrying media information from various senders to various receivers. Furthermore, such activities are also associated with an *activity period* (e.g., scheduled time and the duration of a lecture, an examination, an inter-organizational meeting, etc.). We refer to such a collection of media streams with its associated value of activity period as a *session*, and is the abstraction at the next higher level above streams. The activity period for each session can be defined by an Activity Start Time (AST) and an Activity End Time (AET).

Traditionally, collaborations have been separated into interactive and non-interactive categories, and separate programming as well as user interfaces have been used for their management. Associating an activity period with each session enables us to unify the representations of both interactive and non-interactive collaborations. Such a unification of both interactive and non-interactive collaborations in our model permits co-existence of both types of collaborations within a single multimedia application. Notice that associating an activity period with a session also provides an estimate of the network resources required for carrying out a collaborative activity, which can then be utilized by a network resource manager to reserve the resources for the entire activity period of a session.

Whereas a session provides a convenient abstraction for a collective activity in a common time span, a complete collaborative endeavor may, in general, involve several periods of collective activity, possibly with dependencies between the outcomes of some periods and causations of others. For instance, a course at a university involves a sequence of lectures followed by an examination, each of which can be characterized as a session. Similarly, a panel discussion involves a sequence of presentations by the panelists, which is followed by a question-answer period. All such sessions, whose activity periods are temporally related, constitute a *conference*, which is the term denoting a complete collaborative endeavor.

A collaboration $\mathcal{C}$ is a tuple:

$$\mathcal{C} = < H, M >$$

where,

- H is the set of hosts, which send/receive messages.
- M = {text, audio, video} is the set of media.

Based on the intuitive ideas presented above, we provide precise definitions of the stream, session, and conference abstractions in the following subsections.

4.1 Stream

A stream S is defined as a relation among sets of hosts (senders and receivers) and media:

$$S\text{: Powerset(H) x Powerset(H) x M}$$

where, `Powerset(H)` is the set of all subsets of set of hosts `H`.

In PVS, we first define `host` as an *uninterpreted* type, media_type as a scalar type, and a stream_type as a 2-tuple whose elements are sets of hosts:

```
host: TYPE % uninterpreted
media_type: TYPE = {text, audio, video}
stream_type: TYPE = [set[host], set[host]]
```

A *stream* is defined as a record type containing a stream_type and a media_type:

```
stream_rec: TYPE = [# stream:stream_type,
                      media:media_type #]
```

The definitions of senders and receivers follows from the definition of a stream by respecively taking the first and the second element of a stream_type, using projection functions: streams:

```
senders(s) = proj_1(stream(s))
receivers(s) = proj_2(stream(s))
```

where senders denotes the permitted senders of media information; and receivers represents the set of permitted receivers. A user is a member of the receivers if and only if he or she possesses access rights to receive streams transmitted by sender. We formalize the mechanism of providing access rights as part of a *session* in the next Section.

4.2 Session

A session ψ is formally defined as a 4-tuple:

$$\psi = <\widehat{\mathcal{S}}, \mathcal{A}, \mathsf{AST}, \mathsf{AET}>$$

where $\widehat{\mathcal{S}} = \{S_1, S_2, ..., S_n\}$ represents a set of media streams; $\mathcal{A}$ represents access rights, and AST and AET denote the activity start and end times, respectively. Media streams can have existence only insofar as they are associated with a session, but never independently. Since a stream encapsulates the access rights of both the sender and the receiver with respect to the media exchanged on the stream, a session denotes collective access rights of all the participants of a session with respect to all of its constituent media streams. Thus, a session represents a semantic context, and aggregates all the media streams associated with the context. The definition of a session in PVS is specified as follows:

```
time: TYPE = real
session_type: TYPE = set[stream_rec]
session_rec: TYPE = [# AST: time,
           session: session_type,
  has_access?:pred[[host,stream]],
                          ASE: time
                       #]
```

where, `session_type` is defined as a set of stream records, and a `session_rec` is the record containing activity start time `AST`, `session_type`, a predicate `has_access?` denoting if a host has a access right to the session, and the activity end time `AET`.

We can now define the senders and receivers of a session as follows:

```
senders(ses) =
LET stream_recs =
  {s:stream_rec| member(s,session(ses))}
IN
{h:host | EXISTS (s1:stream_rec):
          member(h,senders(s1)) AND
          member(s1,stream_recs)}

receivers(ses) =
LET stream_recs =
  {s:stream_rec| member(s,session(ses))},
    receiversset = {rs:set[host] |
            EXISTS (s1:stream_rec):
    (rs = receivers(s1)) AND
            member(s1,stream_recs)}
IN  Union(receiversset)

participants(ses):set[host] =
{h: host | member(h,senders(ses)) OR
           member(h,receivers(ses))}
```

where, `participants` denotes all the senders and receivers of a session. However, we further make sure that the senders and receivers of a session have the proper access right to their sessions:

```
access_ax: AXIOM % note that the other way
                 % round need not be true
member(s,session(ses)) AND
member(h,senders(s)) OR
member(h,receiver(s)))
     IMPLIES
has_access?(ses)(h,s)
```

We express the requirement that the sender be also one of the receivers by the following axiom:

```
sender_receiver_ax: AXIOM
FORALL h,s: member(h,senders(s)) IMPLIES
            member(h,receivers(s))
```

A special kind of a session, in which all the participants receive exactly the same set of media streams is referred to as a *confluence*. In a confluence, all the participants can be regarded as transmitting media information to the confluence as a whole, and all the receivers can be regarded as receiving mixed media stream from the confluence as a whole. Most commonly held sessions fall under this category.

Confluences provide an excellent opportunity for optimizing network bandwidth usage. This is because, in a confluence, rather than transmitting individual media streams from senders to receivers, media streams transmitted by all the senders can be mixed together and broadcast to all the receivers on the network [23, 30]. The definition of a confluence is formalized in PVS as follows:

```
confluence?(ses:session) =
FORALL (s1|member(s1,ses)),
       (s2|member(s2,ses)):
  receivers(s1) = receivers(s2)
```

We can then derive a theorem that a confluence can be regarded as a single host for optimization:

```
  server_transform: THEOREM
  confluence?(ses) AND
  every_receiver_is_a_sender?(ses) IMPLIES
  singleton(remove_media_type(ses))
```

where, `every_receiver_is_a_sender?` and `remove_media_type` are defined in PVS as:

```
every_receiver_is_a_sender?(ses:session) =
senders(ses) = receivers(ses)

remove_media_type(ses:session) =
{s:stream_type| EXISTS s1: s = proj_1(s1)
             AND member(s1,session(ses))}
```

The `server_transform` theorem stated above can be used to automatically detect confluences within a session, and optimize network bandwidth usage. This is illustrated in Figure 1, which represents an inter-organizational meeting session through mediators (namely, P_{M12} and P_{M23}). Members of each organization (namely, $\{P_1, P_2, P_3\}$ and $\{P_4, P_5, P_6\}$) have closed-door discussions amongst themselves, and then communicate the consolidated views of their organization to the other through the mediators. Such a session is amenable to decomposition into three confluences c_1, c_2, and c_3, consisting of the set of participants $\{P_1, P_2, P_3, P_{M12}\}$, $\{P_4, P_5, P_6, P_{M23}\}$, and $\{P_{M12}, P_{M23}\}$, respectively.

4.3 Conference

A conference $\mathcal{C}$ can be formally defined as a 2-tuple:

$$\mathcal{C} = <\Psi, \mathcal{T}>$$

where $\mathcal{P}$ denotes the list of participants of the conference, Ψ denotes the set of sessions constituting the conference, and $\mathcal{T} = \{T_1, T_2, ..., T_k\}$ denotes the set of temporal ordering relations between the sessions. In general, ordering of the activity periods of sessions ψ_i and ψ_j can be represented by any of the 13 temporal relations (namely, *before*, *meets*, *during*, *overlaps*, *starts*, *ends*, and *equal* plus their inverse relations - except *equal*) [3]. Since a session encapsulates the collective access rights of all its participants with respect to all its constituent streams, temporal sequencing of sessions causes dynamic changes in the access rights of its participants.

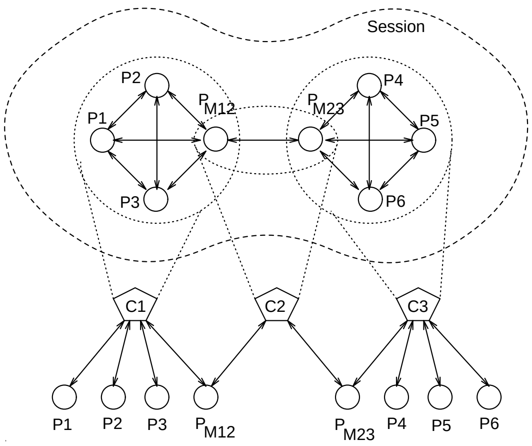

Figure 1: Decomposition of a session into confluences

4.4 Discussion

Nested collaborations can be constructed by permitting the sender or any member of receiverList in a stream of a session to itself be another session. If the sender is a session, then the stream carries a mix of the media information transmitted by the session's participants. On the contrary, if a receiver is a session, then each participant of that session has the rights to receive the media information transmitted by the stream's sender.

Asymmetric collaborations can be obtained by associating different receiverLists with different media streams constituting a session. Heterogenous collaborations, which entail conversion of one medium into another, can be constructed by chaining together multiple streams, with software services acting as *filters* in between the streams. For instance, if the audio transmission from participant P_1 is to be received in textual form at participant P_2, and if A_TO_T denotes the speech-to-text filter, then the heterogenous media exchange between P_1 and P_2 can be accomplished by chaining together two streams S_1 and S_2, where S_1 is an audio stream from P_1 to A_TO_T and S_2 is a text stream from A_TO_T to P_2. Non-Interactive collaborations can be captured by the model by setting AST = AET.

Persistent collaborations (such as meeting rooms proposed by Ahuja et al. [2]) can be obtained by permitting temporal relations among sessions to be mutually recursive. For example, a temporal relation $\psi_i \overset{\tau}{\leadsto} \psi_i$ indicates that session ψ_i is to be re-activated τ units of time after the termination of its previous activation.

4.5 Illustration

In order to illustrate the use of the model presented in the previous section, let us consider the example of a *course* at a university, which involves two types of participants, namely, *students* and a *professor*. Generally both the professor and the students are simple participants, but it is not uncommon for some of the students to be super participants representing groups of individuals belonging to various departments or organizations. The course consists of a sequence of *lectures* followed by an *examination*. Whereas during lectures, both the students and the professor can freely exchange media information, during the examination, students are disallowed from exchanging media information directly. Students may attend a lecture live while it is in progress, or view its recorded proceedings at a later time. Even the professor may choose to broadcast a pre-recorded lecture instead of presenting it live. Additionally, lectures permit students to participate using the media of their choice which can even be changed on the fly. However, the examination does not permit any of these flexibilities. Supporting handicapped students may further require media conversion during both the lectures and the examination. Thus, a course typifies a collaboration that can take on any of the taxonomic dimensions[1].

5 Interfaces for Managing Conferences

The model for structured multimedia collaborations presented in Section 4 enables application designers to precisely specify the requirements of a wide variety of collaborative environments, each of which can be maintained in a database. While initiating a conference, a user can utilize a pre-defined collaboration semantics from the database, or define a completely new type of collaboration. In either case, the transport requirements of the conference are derived from the chosen collaboration semantics, and are utilized as 'hints' during the resource reservation process. Notice that, whereas the chosen collaboration semantics characterize the most common, and hence the expected, form of interaction among the participants of a collaboration, participants may be permitted to alter the semantics at any time during the progress of the conference.

5.1 Editing Operations

Streams can also be manipulated in a manner similar to strings using INSERT, DELETE, and REPLACE operations. Precise semantics of these operations depend on whether the streams being manipulated are interactive or non-interactive in nature. Interesting effects can be obtained by inserting non-interactive streams into interactive streams, and vice versa. For instance, while receiving media from a stream S_1, if a participant inserts an interval I of another stream S_2, the participant's media reception is switched from S_1 to S_2 for the duration of I, at the end of which, reception from S_1 is resumed. On resumption, media exchanged on S_1 during the interval I is skipped for reception at the participant if S_1 is an interactive stream, and not otherwise.

[1]Namely, simple or nested, interactive or non-interactive, static or dynamic, symmetric or asymmetric, homogenous or heterogenous, and transient or persistent.

6 Implementation

A prototype collaboration management system has been implemented at the UCSD Multimedia Laboratory in an environment of multimedia stations, each consisting of a Sun SPARCstation, a PC-AT, a video camera, and a TV monitor (see Figure 2). The SPARCstations and PC-ATs are connected via Ethernets and FDDI. The PC-ATs are equipped with digital video and audio processing hardware produced by UVC Corporation [17]. The audio hardware digitizes audio signals at 8 KBytes/sec, and the video hardware can digitize and compress motion video at real-time rate up to NTSC broadcast with a resolution of 480x200 pixels and 12 bits of color information per pixel. A file server equipped with high capacity disks acts as the multimedia storage server.

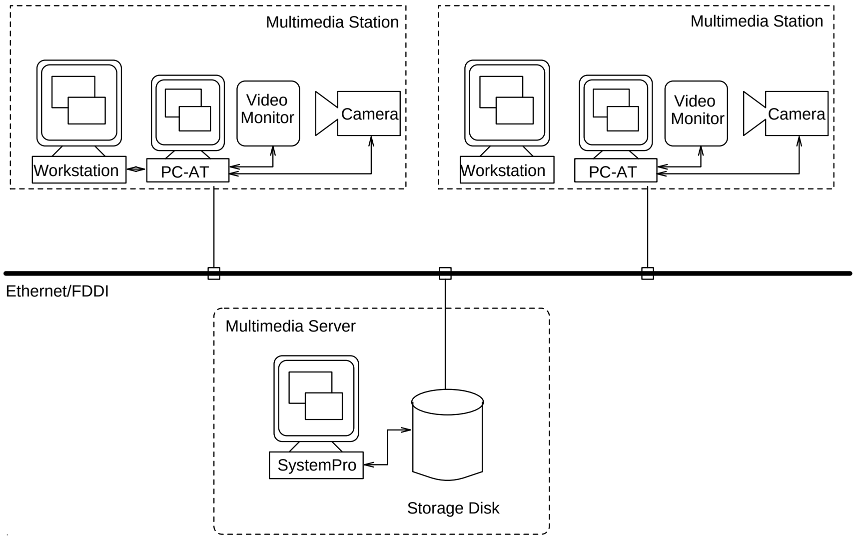

Figure 2: Hardware configuration

The software architecture of our collaboration management system consists of four types of components: *user interface manager*, *conference managers*, *session managers*, and *stream managers*. The user interface manager provides the linkage between the user and the underlying collaboration management system, and supports mechanisms for initiating different types of collaborations. Once the type of the collaboration is selected, the user interface module creates a conference manager, and initializes it with the chosen semantics. Each conference manager performs the functions of a 'schedule keeper' and initiates sessions in accordance with the temporal relationships specified in the semantics. For each session, the conference manager creates a session manager, which is responsible for managing the establishment and progress of a session. The key functions performed by a session manager include: sending preliminary invitation to a list of invitees and accumulating acceptances, as well as carrying out negotiations to decide the mode of participation for each of the participants in the session. On termination of the negotiation process, each session manager creates a stream manager for each of the streams constituting the session. Each stream manager is responsible for setting up the network connectivity for media communication, as well as modifying it in response to changes in the mode of participation of users.

Using the prototype collaboration management system, we have developed several multimedia collaborative applications that are in daily use in the Department of Computer Science and Engineering at the University of California, San Diego. One such collaborative application is a *tele-presenter*. The tele-presenter enables users at their workstations to remotely attend lectures in progress. Lectures are broadcast as sessions to which users can join or leave at any time. Each participant in a tele-presentation receives snapshots of video (the snapshots can be arranged to be captured whenever the speaker puts up a new slide for presentation), as well as continuous audio originating from the lecture.

7 Conclusions and Future Work

In this paper, we have presented a formal basis for reasoning about structured collaborations, and have developed a model that can capture the requirements of diverse types of interactive and non-interactive collaborations. Both efficiency and power are combined via a hierarchy of three abstractions, at the lowest level of which are *streams* for media communication modulated by access rights of participants within collaborations. The higher two levels of abstractions are *sessions*, which represent collections of semantically related media streams, and *conferences*, which represent temporally related sequences of sessions. The model yields a rich environment and a precise understanding of structured collaborations for constructing multimedia applications. A prototype collaboration management system based on the model has been implemented at the UCSD Multimedia Laboratory.

As part of future work, we intend to extend the mechanization of our formal model to investigate a transformational approach to synthesis of multimedia applications from high-level formal specifications. An extended formal model would also provide a basis for rapid analysis of safety properties such as security, and liveness properties such as absence of deadlock in structured collaborations. We also intend to make use of our extended formal model to study how decisions at high-level architectural design can affect performance of multimedia applications at the implementation level.

References

[1] L. Aguilar, J.J. Garcia-Luna-Aceves, D. Moran, E.J. Craighill, and R. Brungardt. Architecture for a multi-media tele-conferencing system. *Proceedings of the SIGCOMM'86 Symposium on Communications Architectures and Protocols, Stowe, VT*, pages 126–136, August 5-7, 1986.

[2] S.R. Ahuja, J. Ensor, and D. Horn. The rapport multimedia conferencing system. In *Proceedings of COIS'88 Conference on Office Information Systems, Palo Alto, CA*, pages 1–8, March 23-25, 1988.

[3] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[4] L. Bannon and K. Schmidt. Cscw: Four characters in search of a context. *Studies in Computer Supported Cooperative Work: Theory, Practice and Design, Edited by J.M. Bowers and S.D. Benford, North-Holland*, pages 3–16, 1991.

[5] J.M. Bowers and S.D. Benford. *Studies in Computer Supported Cooperative Work: Theory, Practice and Design.* Norht-Holland, 1991.

[6] S. Casner, K. Seo, W. Edmond, and C. Topolcic. N-way conferencing with packet video. *Proceedings of the Third International Workshop on Packet Video, Morristown, NJ*, March 22-23, 1990.

[7] D. Cyrluk. Microprocessor verification in PVS : A methodology and simple example. *SRI-CSL-93-12, Technical Report Computer Science Laboratory, SRI International, Menlo Park, CA*, December 1993.

[8] D. Cyrluk, S. Rajan, N. Shankar, and M. Srivas. Effective theorem proving for hardware verification. *Proceedings of the 2nd International Conference on Theorem Provers in Circuit Design, Bad Heerenalb (Blackforest), Germany*, September, 26-29 1994.

[9] T. Danielsen, U. Pankoke-Babatz, W. Prinz, A. Patel, P. Pays, K. Smalland, and R. Speth. The amigo project: Advanced group communication model for computer-based communications environment. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW'86), Austin, Texas, ACM Press*, pages 115–142, 1986.

[10] P. Dewan. A tour of the suite user interface software. *In Proceedings of the 3rd ACM SIG-GRAPH Symposium on User Interface Software and Technology*, pages 57–65, October 1990.

[11] Charles Dunlop and Rob Kling. *Computerization and Controversy: Value Conflicts and Social Choices.* Academic Press, 1991.

[12] Jolene Galegher, Robert E. Kraut, and Carmen Egido. *Intellectual Teamwork: The Social and Technological Foundations of Cooperative Work.* Lawrence Erlbaum Associates, 1990.

[13] S. Greenberg. Personalizable groupware: Accommodating individual roles and group differences. *Proceedings of the European Conference on Computer Supported Cooperative Work (ECSCW) Amsterdam*, September 1991.

[14] W. Harrison, H. Ossher, and P. Sweeney. Coordinating concurrent development. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW'90), Los Angeles, California, ACM Press*, 1990.

[15] A.W. Holt. Diplans: A new language for the study and implementation of coordination. *ACM Transactions on Office Information Systems*, 6(2):109–125, April 1988.

[16] K.A. Lantz. An experiment in integrated multimedia conferencing. In *Proceedings of CSCW'86*, pages 267–275, December 1986.

[17] M. Leonard. Compression chip handles real-time video and audio. *Electronic Design*, 38(23):43–48, December 1990.

[18] Patrick Lincoln, Sam Owre, John Rushby, N. Shankar, and F. von Henke. Eight papers on formal verification. *Technical Report, SRI-CSL-93-4, Computer Science Laboratory, SRI International, Menlo Park, CA*, May 1993.

[19] S. Owre, J. Rushby, N. Shankar, and F.v. Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *to appear in IEEE Transactions on Software Engineering*, 1995.

[20] S. Owre, N. Shankar, and J.M. Rushby. User guide for the PVS specification and verification system, language, and proof checker (beta release). *Computer Science Laboratory, SRI International, Menlo Park, CA*, February 1993.

[21] W. Prinz and R. Speth. Group communication and related aspects in office automation. *Proceedings of the IFIP TC 6/WG 6.5 Conference on Message Handling Systems, Munich*, 1987.

[22] P.S. Rajan. Transformations in high-level synthesis: Formal specification and efficient mechanical verification. *SRI-CSL-94-10, Technical Report Computer Science Laboratory, SRI International, Menlo Park, CA*, October 1993.

[23] Srinivas Ramanathan, P. Venkat Rangan, and Harrick M. Vin. Designing communication architectures for inter-organizational multimedia collaboration. *Journal of Organizational Computing Systems*, 2(3&4):277–302, 1992.

[24] M. Robinson. Double level languages and cooperative working. In *Proceedings of Support, Society and Culture - Mutual Uses of Cybernatics and Science, Amsterdam, March 27 - April 1*, pages 79–114, 1989.

[25] M. Roseman and S. Greenberg. Building flexible groupware through open protocols. *Proceedings of the ACM Conference on Organizational Computing Systems (COCS-93)*, 1993.

[26] S. Sarin and I. Greif. Computer-based real-time conferences. *IEEE Computer*, 18(10):33–45, October 1985.

[27] H. Smith, P. Henessy, and G. Lunt. An object-oriented framework for modelling organisational communication. *Studies in Computer Supported Cooperative Work: Theory, Practice and Design, Edited by J.M. Bowers and S.D. Benford, North-Holland*, pages 145–158, 1991.

[28] Lee Sproull and Sara Kiesler. *Connections: New Ways of Working in the Networked Organization.* MIT Press, 1991.

[29] Harrick M. Vin, P. Venkat Rangan, and M.S. Chen. System support for computer mediated multimedia collaborations. *In Proceedings of the ACM 1992 Conference on Computer Suppported Cooperative Work (CSCW '92), Toronto, Canada*, November 2-4, 1992.

[30] Harrick M. Vin, P. Venkat Rangan, and Srinivas Ramanathan. Hierarchical conferencing architectures for inter-group multimedia collaboration. In *Proceedings of the Conference on Organizational Computing Systems (COCS'91), SIGOIS Bulletin, Vol. 12, No. 2-3*, pages 43–55, November 1991.

[31] Harrick M. Vin, P.T. Zellweger, D.C. Swinehart, and P. Venkat Rangan. Multimedia conferencing in the etherphone environment. *IEEE Computer - Special Issue on Multimedia Information Systems*, 24(11):69–79, October 1991.