Touring Machine: A Software Platform for Distributed Multimedia Applications[*]

M. Arango P. Bates R. Fish G. Gopal N. Griffeth G. Herman
T. Hickey W. Leland C. Lowery V. Mak J. Patterson L. Ruston
M. Segal M. Vecchi A. Weinrib S. Wuu

Bellcore[†]
445 South St.
Morristown, NJ 07962-1910

Abstract

The goal of the Touring Machine project is to provide a reliable and extensible software platform that supports independently-developed distributed multimedia applications. The project includes an experimental testbed composed of a network of desktop video and audio devices controlled via user workstations. Touring Machine is more than a research testbed; it is the basis of the communications tools used daily by 100 users in two Bellcore locations 50 miles apart. It supports multimedia conferencing and information services as well as point-to-point communications. This paper describes Touring Machine, its system model and its software architecture.

1 Introduction

With the technological advances in computing and communication networks in recent years, we have seen the emergence of a variety of distributed multimedia applications[7, 22]. Examples of such applications include Computer Supported Cooperative Work (CSCW)[11], medical applications[9], multimedia conferencing[19, 21], and distance education[20]. Although most research in this area centers on individual multimedia applications on stand-alone hardware, great potential for multimedia computing lies in the widespread development and use of distributed multimedia applications on top of a common platform or system infrastructure. Without the system infrastructure, distributed multimedia applications would have to deal with the complexities of multimedia communications: routing, resource allocation, session control, network management, presentation control, multimedia device control, and media synchronization, in addition to the complexities of managing the application itself. The system infrastructure can reduce the burden on the application developer by providing the basic common functionalities.

A primary goal of the Touring Machine project is to provide an open software platform that facilitates the widespread development and use of distributed multimedia applications on a variety of wide area networks. Touring Machine is designed to separate the concerns of application

[*]Submitted to the 1992 IFIP International Conference on Upper Layer Protocols, Architectures, and Applications, Vancouver, Canada, May 1992.

[†]Contact author: Victor Mak, 445 South St., 2P-254, Morristown, NJ 07962-1910, (201) 829-4470, Internet: vicmak@bellcore.com

development from the complexities and details of establishing and modifying multimedia communication sessions. A general Application Programming Interface (API), supporting a rich set of capabilities of the system, is provided to the application developer. The API defines the set of messages passed between user applications and Touring Machine. This API has been designed in close collaboration with distributed multimedia application developers from the CRUISER[1][6, 8] and RENDEZVOUS[2][17] projects within Bellcore, and has incorporated many concepts from work on Broadband ISDN (B-ISDN) signaling[16] from the EXPANSE project[1]. The CRUISER service is a multimedia communications application designed to support informal communications among remotely located co-workers through the medium of an audio and video network. It allows users in different physical locations to construct and participate in a *virtual workplace* without leaving their own desks. The RENDEZVOUS system is an architecture for creating synchronous multi-user applications, such as a multi-user whiteboard and multi-user games. The EXPANSE signaling protocol supports the establishment and modification of complex multimedia services for B-ISDN in light of the limitations of Q.931, the user-network signaling protocol for narrowband ISDN.

An important objective in designing Touring Machine has been to impose a strict separation of *policy* from *mechanism* for communication session management. Policy is set by applications; Touring Machine does not impose any particular policy. This separation of policy from mechanism allows Touring Machine to support independently-developed distributed multimedia applications that may have very different policies in their communication models.

Touring Machine provides a set of logical abstractions for specifying the transport topology of a communication session. These abstractions hide the details of the underlying physical network from the application developer, and allow arbitrary configurations to be specified. The Touring Machine API provides the capability for an application to set up and modify multiple concurrent multi-user multimedia communication sessions. These sessions can share the same set of station resources. An application has separate control of all supported media: audio, video, and data. Each component of a communication session can be individually suspended or resumed. Touring Machine also includes an integrated name server that acts as the repository of both static and dynamic information about the system. The information listed in the name server can be browsed by user applications, subject to privacy policies, to implement various intelligent services that require knowledge of the current system state.

Beyond defining the API, a second goal of the project is to investigate research issues important to the development and maintenance of a large communication control system that operates in a public environment. Issues such as scalability, extensibility, fault tolerance, maintenance, multiple administrative domains, and heterogeneity must be addressed. The internal architecture of Touring Machine has been designed to address some of these issues. In this paper, however, we focus on describing the system functionalities provided to applications; more details on the motivation for the internal architecture are provided in [10].

The current version of Touring Machine software is structured as a set of distributed objects working cooperatively to provide the services supported by the API. The system controls desk-top video and audio devices connected through a network of multiple switches and other specialized hardware resources such as audio and video bridges and mixers. The current version uses analog audio and video hardware; while not sophisticated, this choice allows us to support a large and active user population (about 100 users across two Bellcore locations 50 miles apart with digital links), and concentrate our efforts on developing the software platform and multimedia applications. Touring Machine is an evolving project. The current version represents the second iteration of system design

[1]CRUISER is a trademark and service mark of Bellcore.

[2]RENDEZVOUS is a trademark of Bellcore.

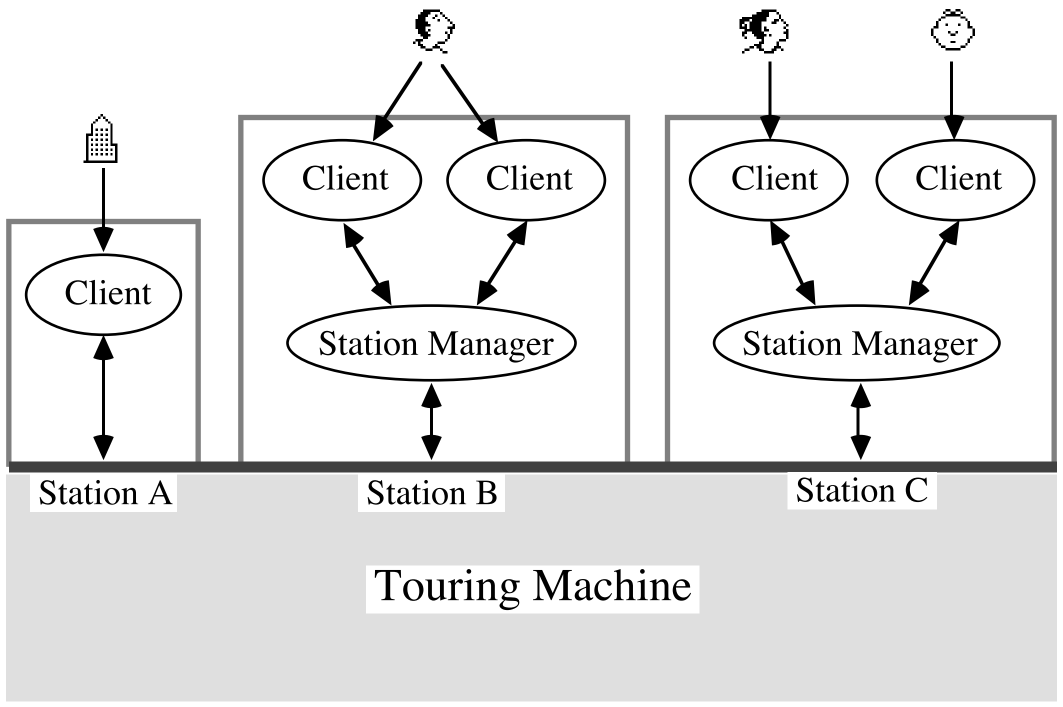

Figure 1: System Model of Touring Machine

and is implemented by 100K lines of C code; the first version of the system was described in [2].

The rest of this paper is organized as follows. Section 2 describes Touring Machine's system model and the API provided to the application developer to develop distributed multimedia applications. This section describes the relationships among the various abstractions used in the system model and summarizes the services provided by the API. Section 3 presents the software architecture of the current version of Touring Machine. This section describes the functionality of and relationships among the set of distributed objects that make up the platform. Section 4 concludes the paper by contrasting Touring Machine with related work and discussing ongoing research.

2 Touring Machine System Model

A primary goal of the Touring Machine project is to provide a platform upon which applications requiring complex multimedia communications can be developed, independent of the actual network fabric used to provide transport. To achieve this goal, Touring Machine provides an abstract model of communication that enables the application developer to concentrate on the logical specification of the application itself, instead of worrying about the physical realization of the communication service, such as routing across multiple switches and bridging for multi-party sessions. The complexities in session management, resource allocation, and network management are hidden from the application.

The relationship between application software and Touring Machine is of the *client-server* type. Touring machine is a *server* providing multimedia communication services to a number of *clients* (application software that act as agents of users in the system) at various geographically distributed *stations* (see Figure 1). A user represents the responsible entity on whose behalf service is authorized. Typically, a user instantiates a person, but it might be a network service provider which

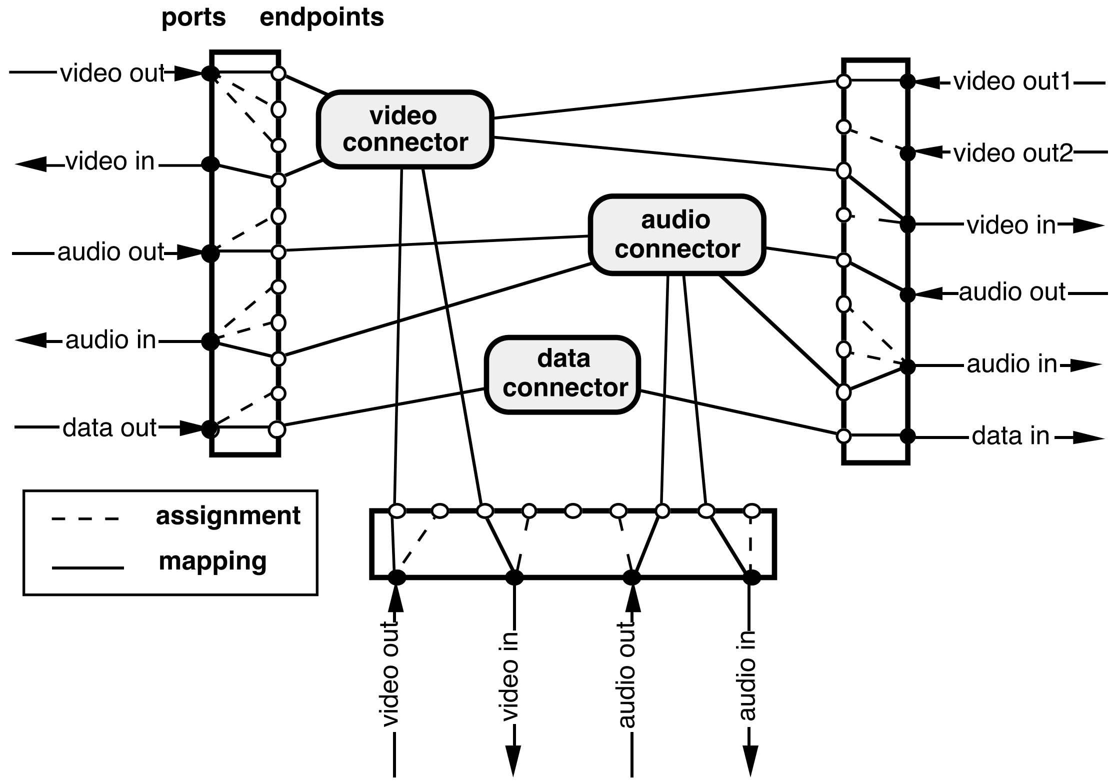

Figure 2: The relationships among Touring Machine abstractions for a three-party multimedia session

offers some communication applications to other users. A station in a user's office consists of an arbitrary configuration of audio/video equipment; network access for a station is controlled by Touring Machine, and can be shared among the clients (belonging to the same user or different users) at that station. A user may have one or more clients requesting services from Touring Machine simultaneously. For instance, a user can have one client handle video conferences and another handle a shared whiteboard application. An optional specialized client called the *station manager* may be used to coordinate among the other clients at that station. The station manager implements various *policies* for resource sharing among clients at that station. It also provides common features (such as session screening and forwarding) to the other clients.

One of the most important aspects of Touring Machine is the separation of the management policy from the mechanism that enforces the policy, both to support a diverse set of applications and to more easily evolve the system as the number of supported application grows. This design decision is based on the observation that dealing with embedded policy gets complex as system functionality evolves and can lead to conflicts among applications, feature interactions, which are hard to detect and resolve[5]. The explicit assertion of policy as in Touring Machine allows mechanized detection of conflicts.

A *session* represents a multi-client, multimedia communication. Touring Machine provides the mechanism to enforce different session policies that are specified and agreed to by the participating clients on a per-session basis. Currently supported policies include privacy and permission. Privacy policy controls whether other clients can find out the session's existence and its attributes such as a member list. Permission policy controls which clients can issue change requests to the session.

The transport topology of a session is specified logically as a set of typed *connectors* (see

Figure 2). A connector represents a multi-way transport connection between *endpoints* (logical ports). A connector is an abstraction of a communications bridge, including point-to-point (two-way) as well as multi-point connections. Since bridging is a medium-specific operation, connectors are typed by medium. A session may have one or more connectors per medium, with source and sink endpoints from participating clients. An endpoint represents a connector termination point. Endpoints are distinguished by medium, direction of flow, and client receiving or providing transport. An endpoint represents only a logical transport termination point; it yields real transport when mapped onto a *port*. A port is typed by its medium and direction of flow, and represents a network access channel to which a station's audio/video/data equipment may be attached. A client specifies the port to which each of its endpoints is *assigned* in a session, and may then *map* and *unmap* its endpoints to the ports to share the access channels among multiple concurrent sessions. Unmapping an endpoint-to-port assignment places a component of a session on hold, while mapping an assignment resumes that component.

The Touring Machine Application Programming Interface (API) defines the set of messages passed between a client and Touring Machine for requesting services from Touring Machine. The functionality of the API can be divided into the following five categories: client registration, session management, network access control, name server query, and inter-client message passing.

2.1 Client Registration

A client must register at a station before it can issue a request to Touring Machine or participate in a session. Touring Machine authenticates the client and allows the client's current physical location to be known to other clients. A client can specify at registration, and later change, information such as its set of endpoints, privacy, and other attributes that may be of interest to other clients.

2.2 Session Management

A client creates a session by specifying the initial attributes (the set of participating clients, the transport topology, and the policies) for the session. Creation of a new session involves two stages: negotiation of session management policies and resource allocation. After all clients agree on the session, Touring Machine translates the logical specification of the transport topology, in terms of connectors, into network resource requirements, such as trunks and bridges, and enters the resource allocation stage. If the second stage is also successful, a new session is created. Clients may leave or join a session, and may modify the transport topology and/or policies of an existing session using messages defined by the API.

2.3 Network Access Control

The API allows a client to specify and change its endpoint-to-port assignments as well as its endpoint mappings during a session. For example, a client may change the assignment of its video source endpoint from a camera pointing to the user, to a camera pointing to a document, or may change the assignment of its audio sink endpoint from a speaker to an audio recorder. The flexibility in changing the endpoint to port assignment dynamically allows a user to change the local termination of a connector without changing the session's logical topology. The endpoint-to-port assignments can be mapped and unmapped to resume and suspend certain components of a session. However, network resources are not released when an endpoint is unmapped so that connectivity is guaranteed when the endpoint is later mapped.

A client may dynamically create a new port at the local station. Since the transport mechanisms for audio and video in the current release of Touring Machine are analog (and thus audio and video ports are pre-created and fixed in number), only data port creation is currently supported. However, in the future when the system migrates to a digital domain, ports of all media will be created and destroyed dynamically, limited only by the available network access bandwidth.

2.4 Name Server Query

The name server acts as the central repository of both static and dynamic information for Touring Machine. It contains information of the following categories: authorized users, registered clients, ongoing sessions, Touring Machine stations and their ports. A client can query the name server to find out, for instance, all registered clients supporting a particular application, all interesting sessions to join, or all clients in a particular session, subject to the privacy policies set on the stored information. The name server enables the development of intelligent applications that require knowledge of both static and dynamic information of the system.

2.5 Inter-Client Message Passing

Touring Machine provides a datagram service for inter-client message passing. Unlike the data medium in a session that provides a stream connection and requires session setup procedures, the inter-client message passing service provides an efficient way to send datagram messages between clients without the overhead of session setup.

3 Touring Machine Software Architecture

The current version of the Touring Machine software is structured as a set of distributed objects that work cooperatively to realize the API. The software architecture includes two types of objects: permanent objects, which exist as long as the system remains available to users, and transient objects, which exist for a limited amount of time. Touring Machine does not require any object to execute at a particular physical location, unless it controls a piece of hardware that is physically connected to a particular processor.

All Touring Machine objects communicate by asynchronous message passing. Inter-process communication among objects in Touring Machine is provided by the Connection Manager[3]. The Connection Manager is a high-level message-based inter-process communication interface layered on top of BSD UNIX[3] sockets. It also provides a set of routines and support programs that simplify the creation of complex communication protocols.

Touring Machine exhibits a layered architecture (see Figure 3). Above Touring Machine lie applications that use the API to request services provided by the infrastructure. The Touring Machine infrastructure itself is separated into two levels: session control and transport control[4]. The session control level provides logical control of communication sessions. It contains the station object and the session object. The transport control level takes care of physical resource allocation and provides network transparency. It contains the transport object, the resource manager, and various resource objects. A special object, the name server, is accessible by objects in both levels and maintains system information. The subsequent sections describe each of these objects in more detail.

[3]UNIX is a registered trademark of UNIX System Laboratories, Inc.

[4]The Touring Machine's session control and transport control levels are not the same as the session and transport layers defined in the ISO reference model. Our session control level belongs to the application layer in the ISO reference model, whereas our transport control level belongs to the network and transport layers in the ISO reference model.

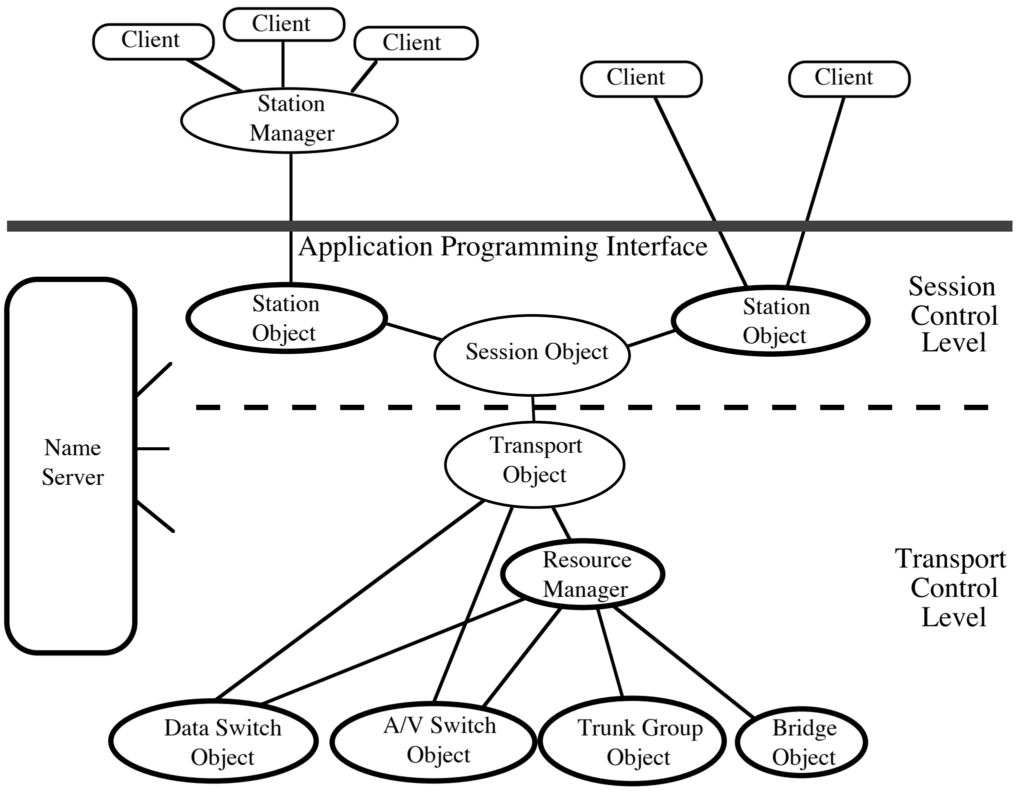

Figure 3: Touring Machine Software Architecture

3.1 Station Object

The station object is the interface point between clients and Touring Machine. Prior to making any requests, a client must first be authenticated and registered with a station object. Once registered, a client sends API requests to that station object. Explicit registration and deregistration requests allow support of mobile users by Touring Machine. The station object checks each message received from a client for syntactic and semantic errors, and routes it to the appropriate Touring Machine object. If a request involves the creation of a new session, the station object instantiates a new session object that handles subsequent messages related to that session. The station object also manages the sharing of network access ports among multiple clients registered at the station using a default policy of first-come-first-served. If other policies are desired, they can be specified and enforced outside Touring Machine, possibly in a station manager.

3.2 Session Object

The session object is a transient object created by the station object upon a client's request to create a new session. The session object coordinates negotiation among clients and maintains the logical state of the session. Currently, the negotiation stage involves only reaching agreement on the session management policies specified in the request by the originating client. A more elaborate form of negotiation that can provide compromises as well as consensus is under investigation[12]. Because session is a named object and is listed in the name server, clients can join an existing session by sending a *join* request to the appropriate session object, subject to the permission of the session.

The session object lacks knowledge of the underlying physical network. It maintains the logical state of the transport in terms of a set of connectors and their endpoints. Any changes in the transport, for example, as clients are added to a multi-user conference, are passed on to an available resource manager for physical realization. Currently, resource allocation is performed only after the negotiation stage. It is possible in the future to use other policies that allow pre-allocation of resources.

3.3 Resource Manager

The resource manager coordinates the allocation and deallocation of system resources, such as trunks and bridges, to realize the physical transport of a session. Requests from a session object can be served by any one of the available resource managers in the system. Maintaining exact, timely, and consistent states of system resources among multiple resource managers is expensive. Instead, an optimistic strategy is employed as a performance optimization: each resource manager maintains approximate information on the system resources and uses these data as *hints* for the heuristic routing and resource allocation algorithms it employs. (Only the resource objects themselves maintain the exact state of network resources.) A resource manager allocates, deallocates, and updates its state information about a given resource by sending requests to the resource object controlling that resource. If a request is rejected, the rejection supplies updated state information to allow the resource manager to update its state model and, if desired, re-evaluate its routing and allocation decisions.

3.4 Transport Object

While the session object maintains the logical state of the transport of a session, the transport object, the second transient object, maintains the logical-to-physical mapping. The transport object is instantiated by the resource manager upon initial successful allocation, and keeps track of the results of subsequent allocation and deallocation requests pertaining to that session. The instantiation of a transport object for each session is designed to alleviate the fault-tolerance problem: the failure of a resource manager does not result in the loss of any session, and the failure of the transport object results in a loss of at only one session.

3.5 Resource Objects

Resource objects manage specific sets of physical resources. They keep track of the exact state of the physical resources, and are the only objects that communicate directly with network hardware. The use of resource objects allows the rest of Touring Machine to remain isolated from the details of particular pieces of hardware. In general, these objects have two interfaces: an interface for communicating with other Touring Machine objects, and a specialized interface for communicating with their associated hardware.

The current release of Touring Machine includes four types of resource objects. The A/V switch object makes audio/video connections. The data switch makes data stream connections. The bridge object allocates and releases audio and video bridges for multi-way communication. The trunk object allocates trunks between different switches.

3.6 Name Server

The name server acts as a repository of static and dynamic information for Touring Machine. The stored information is accessible by clients using the API via the station objects, as well as by objects at all levels. Queries to the name server may include conjunction (A AND B), disjunction (A OR B), and wildcards (user = bob*). The name server provides mechanisms to implement various privacy policies. Access to the information listed in the name server is controlled. For example, information on a client or a session can be kept private, or known only to a user-defined group of clients. The use of the name server to keep client-to-station associations enables personal communication services where clients are called by names rather than by location specific addresses.

4 Discussion

Touring Machine is distinct from other work in the area in its emphasis on defining and supporting an Applications Programming Interface. The API makes available various network-provided capabilities, and core services to a large class of multimedia applications. Related research includes Xerox PARC's work on the software for the Etherphone system[23] and later extensions to include video[21]. Their software architecture is object-based with negotiation among peer objects, but they focus on a single powerful desk-top conferencing application, rather than on providing an open platform that supports multiple applications with their own individual policies and requirements. The multimedia conference control program (MMCC)[19], is designed to control a multimedia conferencing system based on the TWBnet experimental testbed; in its current realization MMCC is also a single conferencing application that enforces a predetermined set of control policies. Leung et al.[15] concentrate on abstractions for handling digital multimedia streams. Blair et al. at Lancaster University[4, 18] start with models emerging from the Open Distributed Processing (ODP) standardization process in Europe, extending them to incorporate multimedia services; they propose abstractions similar to some of ours. The Pandora's Box system experiment[13] comprises a combination of hardware and software to realize video- and audio-based applications on an underlying packet network; they emphasize custom-built hardware to accelerate video processing. Finally, the PX project[14] focuses on voice communications; their focus, similar to ours, is on developing a collection of toolkits that provide common functions to voice applications.

A unique value of the Touring Machine project is that it is a large system experiment with a significant user community, a number of independent applications developers who plan to use the system to conduct sociological and other experiments, and support for heterogeneous multimedia hardware. This mix of features allows (and forces) us to address numerous interesting topics for research, and provides an environment to test our research ideas in practice.

The Touring Machine project is still at an early stage, with many areas of ongoing work on both API and system-software issues. Examples of areas of active API-related research include:

- Support for hybrid (analog and digital) networks, to evolve the API to support both analog networks as well as the rich abstractions available in an digital environment. An important example is support for synchronization of different media types.
- Presentation control, to give clients the ability to specify the presentation of information at their stations. For example, a client ought to be able to specify the positions on the screen of separate video sources being bridged together, and set the volumes of audio streams.
- Applications structuring issues relating to managing multiple clients registered at the same station that have conflicting needs. We are investigating mechanisms for discovering and controlling such interactions between clients. Some of these issues have surfaced only recently, through actual experience with developing multiple applications for Touring Machine.

Systems-software research includes, among others, support for multiple administrative domains, enhanced fault tolerance and availability, privacy and security, naming and addressing, network management, software maintenance, and system instrumentation and observation. Beyond work on the system itself, we are actively developing collaborative relationships with other institutions to expand and enrich the Touring Machine user community.

References

[1] Albanese, A., Bussey, H., Weinstein, S., and Wolff, R., "A Multi-Network Research Testbed for Multimedia Communications Services," *Proc. IEEE ICC'91*, June 1991.

[2] Bates, P., and Segal, M., "Touring Machine: A Video Telecommunications Software Testbed," *Proc. First International Workshop on Network and Operating System Support for Digital Audio and Video*, Berkeley, CA, November 1990.

[3] Bates, P., "A Connection Manager for Rapid Prototyping of Distributed Systems," *in preparation*, 1992.

[4] Blair, G.S., Coulson, G., Davies, N., and Williams, N., "Incorporating Multimedia in Distributed Open Systems," *Proc. EUUG Spring '91 Conference on Distributed Open Systems in Perspective*, Tromso, Norway, May 1991.

[5] Bowen, T.F., Dworak, F.S., Chow, C.-H., Griffeth, N.D., Herman, G.E., Lin, Y.-J., "Views on the Feature Interaction Problem," *Proc. of the 7th International Conference on Software Engineering for Telecommunications Switching Systems*, July 1989.

[6] Clayton, R., "Patch Cords — An Infrastructure for Distributed, Multimedia Applications," submitted for publication, 1991.

[7] Davies, N.A., and Nicol, J.R., "A Technological Perspective on Multimedia Computing," *Computer Communications*, Vol. 14, No. 5, June 1991.

[8] Fish, R. S., "Cruiser: A Multi-media System for Social Browsing," *The ACM SIGGRAPH Video Review Supplement to Computer Graphics*, Vol. 45, No. 6, Videotape, 1989.

[9] Goldberg, M., Georganas, N.D., Robertson, J., Mastronardi, J., and Reed, S., "A Prototype Multimedia Radiology Communication System," *Proc. 2nd IEEE International Workshop on Multimedia Communications*, Ottawa, Canada, April 1989.

[10] Gopal, G., Herman, G., Vecchi, M.P., "The Touring Machine Project: Toward a Public Network Platform for Multimedia Applications," *Proc. 8th International Conference on Software Engineering for Telecommunications Systems and Services*, Florence, Italy, March 1992.

[11] Greenberg, S., "Sharing Views and Interactions with Single-User Applications," *Proc. Conference on Office Information Systems (COIS '90)*, April 1990.

[12] Griffeth, N. and Velthuijsen, H., "The Negotiating Agent Model for Rapid Feature Development," *Proc. 8th International Conference on Software Engineering for Telecommunications Systems and Services*, Florence, Italy, March 1992.

[13] Hopper, A., "Pandora - an Experimental System for Multimedia Applications," *Operating Systems Review*, Vol. 24, No. 2, April 1990.

[14] Kamel, R., Emami, K., and Eckert, R., "PX: Supporting Voice in Workstations," *Computer*, August 1990.

[15] Leung, W-H. F., Baumgartner, T.J., Hwang, Y.H., Morgan, M.J., Tu, S.C., "A Software Architecture for Workstations Supporting Multimedia Conferencing in Packet Switching Networks," *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 3, April 1990.

[16] Minzer, S.E., "Signaling and Control for Multimedia Services," *IEEE Multimedia '90*, November 1990.

[17] Patterson, J.R., Hill, R.D., Rohall, S.L., and Meeks, W.S., "Rendezvous: An Architecture for Synchronous Multi-User Applications," *Proc. CSCW '90*, October 1990.

[18] Ruston, L., Blair, G., Coulson, G., and Davies, N., "Integrating Computing and Telecommunications: A Tale of Two Architectures," *Proc. 2nd International Workshop on Network and Operating Support for Digital Audio and Video*, Heidelberg, Germany, November 1991.

[19] Schooler, E.V., "A Distributed Architecture for Multimedia Conference Control," ISI Technical Report No. ISI/RR-91-289, November 1991.

[20] Smith, R.B., "A Prototype Futuristic Technology for Distance Education," in *NATO Research Workshop on New Directions in Education Technology*, November 1988.

[21] Vin, H.V., Zellweger, P.T., Swinehart, D.C., and Rangan, P.V., "Multimedia Conferencing in the Etherphone Environment," *IEEE Computer*, Vol. 24, No. 10, pp. 69–79, October 1991.

[22] Weinstein, S., "ISDN Multimedia Services," in *ISDN Systems: Architecture, Technology, and Applications*, Prentice-Hall, 1990.

[23] Zellweger, P.T., Terry, D.B., and D.C. Swinehart, "An Overview of the Etherphone System and its Applications," *Proc. 2nd IEEE Conf. Computer Workstations*, IEEE, New York, pp. 160–168, 1988.