

# Architectural Support for Building Automated Capture & Access Applications

Khai N. Truong and Gregory D. Abowd  
College of Computing & GVU Center  
Georgia Institute of Technology  
Atlanta, Georgia 30332-0280 USA  
{khai, abowd}@cc.gatech.edu

## Abstract

*Applications that automatically capture some details of a live experience and provide future access to that experience are increasingly common in the ubiquitous computing community. The exploration of this class of applications raises software engineering problems pertaining to proper software structuring and programming abstractions to support the design, development and evolution of these systems. In this paper, we present our experience in building a number of capture and access applications, sharing insights on relative successes and failures. These lessons learned are now embodied in a toolkit called Infrastructure for Capture and Access (INCA) which supports a distillation of the essential architectural features for this class of applications. We will demonstrate how this toolkit encourages a simplified model for designing, implementing, and evolving such applications—lowering the barrier for building automated capture and access applications and inspiring the construction of a larger variety of applications previously unexplored.*

## 1. Introduction

Our daily lives provide us with a great deal of records and memories that we often want to access again at some point in the future. Inspired by the progress of ubiquitous computing over the past decade, many researchers have demonstrated devices and applications that support the automated capture of live experiences and the future access of those records (see [31] for a full review).

Though there are many demonstrations of automated capture and access applications, it is still difficult to design, implement and maintain these applications. Those with the most creative ideas for automated capture and access must overcome too many accidental hurdles in realizing their vision, so the space of design possibilities is not adequately explored. In addition, there has been relatively little evaluation of capture and access systems

under authentic use, with the notable exceptions of Tivoli [21] and eClass [1, 5], largely because these demonstrations are hard to maintain and evolve over the course of a long-term study of use.

Prototyping an application rapidly, putting it into place and then modifying it according to continual feedback from the user population is not a new software engineering practice. However, this development model for ubiquitous computing applications requires necessary advances in the tools we provide to those creative designers who wish to realize and improve upon the visions of those such as Weiser [34] and Bush [7]. Such tools must support proper software structuring using correct separations of concerns and lightweight component integration techniques for the wide variety of devices and operating systems.

Abowd [2] previously outlined some software engineering challenges necessary for the advancement of research in ubiquitous computing, highlighting the need for support of context-awareness, automated capture and transparent (or natural, off-the-desktop) interaction. We have worked on tools to support context-aware computing [13, 14] and human-assisted error correction [17, 18] resulting from recognition-based interfaces. In these previous cases, our method has been to present the relevant design abstractions for a well-defined class of applications, develop an architectural solution to support the design and construction of these applications, implement a toolkit that embodies these abstractions and then validate the abstractions, architecture and toolkit by developing interesting and complex applications within the design space. We now apply this research method for the class of automated capture and access applications.

## Overview of paper

In Section 2, we review the related work to characterize existing automated capture and access applications from other research institutes—identifying the essential features of capture and access applications that define the relevant design space. In Section 3, we summarize our own research progress and share insights

on the development of three of these applications and relate the kinds of problems involved in the construction task. Having motivated the common functional and nonfunctional features of this class of application, we present in Section 4 the Infrastructure for Capture and Access (INCA) toolkit, embodying the relevant abstractions to support investigation within this domain. In Section 5, we present how this toolkit encourages a simplified model for designing, implementing, and evolving automated capture and access applications.

## 2. Related work

Over the years, a large number of prototype systems have been built as demonstration vehicles to illustrate the potential benefit of automated capture and access for a variety of situations, such as the classroom [1, 3, 5, 11, 22], meeting rooms [8, 23, 25, 28, 33, 36], conferences [15], and other general experiences (such as personal note-taking [27, 35]). The purpose of these applications is to preserve records of an event—typically through audio, video or some other artifacts produced during the live experience—to free the human users to better engage in the experience with high confidence that details will be available when needed later. In the simplest form, a single capture and access device can be constructed to provide this service; however, a collection of capture and access devices can also be used to automatically author documents integrating a number of captured streams of information [20].

Those systems that record just a single audio or video stream typically are built to provide short-term reminder services. The PhoneSlave and Xcapture systems were built to provide a “digital tape loop” of the audio produced in a single office or during telephone conversations [16]. The application itself runs on a workstation that captures from either a microphone input or the phone line and allows the user to quickly review audio content as well as to mark important snippets in the audio loop to save permanently. In MERL’s real-time audio buffering technique, the captured audio can persist for the duration of that phone conversation [12]. During the course of a conversation, a user may tap the phone against the ear to move backwards in the audio to relisten any portion of the discussion thus far. Audio processing techniques are applied to speed up the playback of the audio to allow the user to continue to hear the recorded audio and eventually catch up to the live content. The Where-Were-We application explored providing near synchronous video stream access in a conference room environment [19]. At any time during the meeting, participants can review scenes from that room minutes ago.

In situations when information is typically accessed a large period of time after the live experience, many systems have tried to increase the fidelity of the captured data by integrating additional types of input to just the audio or video. In classrooms and meeting rooms, systems such as Authoring on the Fly [3], the Lecture Browser [22], Tivoli [21], Dolphin [28], FiloChat [35] and NoteLook [8] temporally integrate hand written artifacts or other materials presented during the meeting with the audio and/or video to create a richer set of notes for that experience. The Audio Notebook demonstrates a personal, integrated solution for capturing and accessing multiple streams of information [27]. However, most of these systems divide the capturing responsibilities among different machines in the environment. Because a large number of heterogeneous devices are typically involved in the capture multiple streams of information, coordinating these distributed devices to collectively and reliably perform the capture and access task is a difficult challenge [20]. The STREAMS work introduced a common strategy for capturing multiple streams of information as separate, single medium streams that can be temporally integrated [10]; but other plausible access scheme (using additional integrating factors, such as matching subject matters or locations [9]) are seldomly explored.

## 3. Our investigation of capture & access

Since 1996, we began our own investigation of the area of automated capture and access by constructing a large number of applications investigating domains similar to those described above, such for classrooms [1, 5, 29], serendipitous meetings [6], software architectural discussions [26], distributed meetings [25], and domestic settings. In this section, we present a small set of these applications, highlighting the some of the successes and failures of their development effort.

### 3.1. eClass: The original motivating application

The eClass project (formerly known as Classroom 2000) was an experiment in which we created a classroom environment that captured much of the details from the university lecture experience on behalf of the students, automatically generating a set of Web accessible notes immediately available after class for student review [1, 5]. The task of capturing the various streams of information was divided among several specialized machines. An electronic whiteboard (such as the LiveBoard or SmartBoard) was used in place of a traditional whiteboard, recording slides presented in class as well as the instructor’s handwriting. A machine running a proxy server to log HTTP requests recorded Web pages visited

during lecture. Finally, a separate machine recorded the audio inside the classrooms.

A central server connected to each of these capture services, providing coordination for a given lecture, or capture session. This coordination included collection of prepared materials prior to a lecture, initiating and terminating the recording for all services for a given lecture, and the collection and post-production of all captured materials to create the Web-accessible notes. The system succeeded largely because this coordination was automatic, requiring very little extra instructor or student effort.

Over time, requests from users (both teachers and students) resulted in an extended whiteboard application (i.e., multiple display surfaces showing the history of a lecture), video capture, and a database of captured lectures to support server-side, dynamically generated Web notes. This growth and change in the system, occurring over a 4-year period, was not trivial. This evolution in eClass was possible largely because from very early on we adopted a structure to the capture problem that separated different concerns into four separate phases:

- pre-production to prepare materials for a captured lecture;
- live recording to capture and timestamp all relevant streams;
- post-production to gather and temporally integrate all captured streams; and
- access to allow end-users to view the captured information.

Clear boundaries between the phases allowed for the prototype to evolve with improved capabilities but minimal down-time.

### 3.2. StuPad: Personalizing public capture

One goal of eClass was to relieve students of the need to tediously copy down all the notes presented during class. However, only the instructor's actions were captured during the lecture, excluding students from being able to make the captured record more personally meaningful. As a result, some students continued to take a small amount of private notes with pen and paper, and they were forced to integrate the electronically captured notes manually with their own private notes on paper—a nontrivial task. To better support the integration of each student's notes with the eClass notes, the Student Notepad (or Stupad) system was added to the existing eClass system [30]. StuPad provided students with an interface integrating the prepared presentation, digital ink annotations and Web pages browsed from the public classroom notes into each student's private notebook for additional personal annotations.

From an implementation perspective, the requirements for student personalization of captured lecture notes meant that certain capture services had to be redistributed to a large number of clients. With the extended whiteboard, we were already redistributing recent lecture slides to a neighboring display to provide more persistence of the material presented during a live lecture. The StuPad system relied on a separate server to obtain all the material captured by the eClass system and to redistribute these streams of information to each student's notebooks. To obtain the lecture slides and ink annotations, this server spoke the same protocol as the extended whiteboard did within the eClass system. A similar scheme could not work for redistributing Web page visits to the StuPad clients, because visited Web pages were typically retrieved by the central server at the end of each lecture. Rather than risking adverse changes to the working eClass prototype, the proxy client was designed to periodically poll for the list of Web pages captured and computed what must be new Web visits since the previous list. This solution, while operative, had limitations.

Other solutions were possible for integrating the public classroom notes into each student's private notebook. The most obvious is to program each student notebook to directly obtain these information streams, bypassing the need for the proxy server altogether. However, scalability proved to be an issue. The eClass system was built in such a manner where clients communicated with the central server using a single channel and relied heavily on a network communication protocol where each message needed an acknowledgement before the next message was sent. Furthermore, when a slide is visited or an ink annotation is added, this information is sent to the central server in the same thread of execution as the rest of the whiteboard application. As a result, when multiple clients subscribed for the slides and ink annotations, scalability proved to be an issue. This forced us to create a multithreaded StuPad server capable of queuing up the captured information before redistributing them to subscribing clients (effectively acting much like the event queue in windowing systems).

Though the structuring of eClass into four phases facilitated much evolution and extension of the system, it was clear that certain decisions restricted scaling up the system, as seen with StuPad. The connectivity assumptions of eClass were largely to blame, and these assumptions cut across all phases of the system. As a result, despite discovering scalability problems in the way the eClass system was designed to communicate over the network, modification of this aspect of the system would have required a global redesign across all four phases. Such a change would have been too risky, and as a result, we deferred to the makeshift solution of the StuPad server which respected the original eClass design and connectivity assumptions.

Despite the student motivation to integrate in-class personal notes with the public capture of eClass, StuPad turned out to be a less useful application than expected. When study occurs outside of class, students were concerned that any additional note taking would not be easy to integrate with the captured notes.

At that time, the IBM CrossPad presented an affordable solution, allowing one to work with pen and paper while also capturing an electronic record. Such platform would ultimately require the integration of the private notes and the classroom notes using other contextual relationship beyond just temporal synchronization.

The eClass system stored captured information in a rigid hierarchical structure that consisted of course numbers, terms, and dates of the lectures. This storage scheme suggested a specific way through that information can be access, where students identify the course and then particular lecture date for which they wish to review. However, notes students could capture with the IBM CrossPad may come from outside the classroom. Forcing this record to fit inside the same directory structure was not a logical solution, as they required a more flexible storage.

How to provide access to the personal notes in a number of useful ways proved to be a second difficult challenge. The personal notes themselves can be viewed as a set of continuous pages captured over time. A part of these notes is captured during class, meaning they can be synchronized with the notes captured by eClass after class. Additionally, even if done outside of the classroom, notes written about a particular topic matching something presented during class should be integrated as well.

This project was ultimately terminated because of difficulty in overcoming the storage and access challenges described above. In the eClass system, a rigid storage system was used because only one access scheme was envisioned at the time. From this exploration, we learned the importance of a flexible storage to facilitate interesting access techniques that might be desired later as the system continues to evolve.

### 3.3. Current investigation effort

While we also investigated the automated capture and access of experiences in other domains, the three projects described above summarized well what succeeded and failed in many of the applications we developed. These three projects also motivated the need for a toolkit supporting the development of this class of applications. Toolkits offer uniformity in the ways applications will be built while decoupling (and abstracting away) many of the accidental tasks involved in the development of software from its essential features.

To develop a toolkit to support the development of automated capture and access applications, we began by identifying key lessons learned from the different applications we had built. We also performed an extensive review of the work in this area explored by other researchers as well to identify common architectural features that all developers must support in their applications. In the following section, we present a brief description of some of the related work in this area.

## 4. INCA: A toolkit for capture and access

From a design perspective, any capture and access application is expressible in terms of four basic functions:

- Part of the system is responsible for the **capture** of information as streams of data that are tagged with relevant metadata attributes. A given application may have any number of capture devices.
- Part of the system is responsible for the **storage** of information along with metadata.
- When information needs to be converted into different formats and types, part of the system must **transduce** the information.
- Part of the system is responsible for the **access** to multiple, related, or integrated, streams of information that are gathered as response to context-based queries; i.e., support for the integration of information can be wrapped directly into support for the access of the information, such that when information is requested, related streams of information are jointly provided.

From the implementation perspective, the INCA toolkit provides a direct way to translate the functions above into executable form. We will describe each of these functional parts and some of the other features of INCA that support other important, but nonfunctional, aspects of automated capture and access applications.

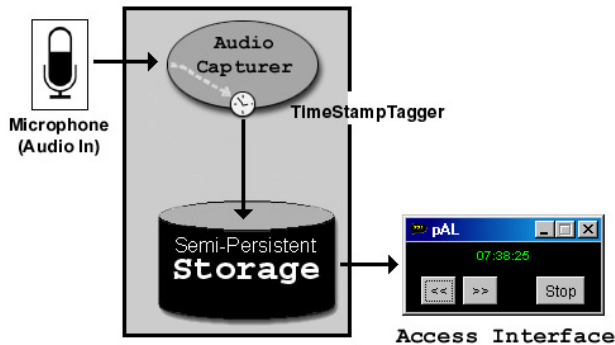
To better describe this toolkit, we will use a simple, but interesting example of a helpful capture and access application. This application, the Personal Audio Loop (PAL), is a near-term audio recording system designed to assist with the resumption of interrupted conversations. The initial version of PAL runs as a standalone application on a single device (laptop, or handheld) that users carry with. PAL constantly records the surrounding conversation, holding a buffer of 15 minutes of recorded audio (conversations older than 15 minutes are lost). When the user wants to be reminded of the content of a prior conversation, after perhaps some interruption, she uses a simple interface to jump backwards in the audio stream. The architecture of PAL and look at the simple access interface are shown in Figure 1.

## 4.1. Capturing and tagging information

INCA defines a *CaptureModule* object to support the capture of information; where capture is defined as the act of collecting data from the physical environment. Often, data is captured and digitized as raw bytes with tagged attributes that describe some properties about the data (such as its data type or format) and the context of the capture activity (such as when and where it was captured). These tags are used in later stages to make the captured data automatically available to those parts of the system that are responsible for storing, transducing, or otherwise accessing it and subscribed for a set of attributes satisfied by the captured information.

In the Personal Audio Loop application, the *AudioCapterer* component extends the behavior of the *CaptureModule* to support the capture of audio information. In a continuous loop, the *AudioCapterer* component reads audio data available from the microphone input into an array of bytes. Periodically, the capture function defined in the *CaptureModule* is invoked, collecting a buffer of audio data and preparing it to be tagged and stored. Every instance of a *CaptureModule* must define this capture function which creates an object that can be sent for storage, transduction or to serve some access service.

Tagging captured data with additional information, or metadata, is important activity. The capture function can be explicitly coded to add attributes to the captured data, but it is better to separate the tagging activity. This is done through the creation of *Tagger* objects that are registered to add metadata information automatically to the output objects from the capture function. For PAL, a *TimeStampTagger* object is registered with the



**Figure 1.** Personal Audio Loop architecture. A near-term audio capture and access reminder system, in which audio from a microphone input is captured, timestamped, and put into a semi-persistent storage (where information lasts for at most 15 minutes). Audio stored in the storage can be accessed by an interface which allows users to jump back 30 seconds at a time in the audio, 7 seconds forward, or to stop (and return to live).

*AudioCapterer* component and when the capture function is invoked, it automatically timestamps the audio chunk before it is distributed to other portions of system.

## 4.2. Storing information

A *StorageModule* represents the portion of the system responsible for storing information, allowing it to persist until it is later accessed. A *StorageModule* can specify a list of attributes for the kind of captured information it is interested in through a subscribe function. Similarly, a publish function can be invoked with list of attributes to let other parts of the system know about the kind of captured information that *StorageModule* can provide. When the capture function in a *CaptureModule* is invoked, the store callback function of any *StorageModule* that has registered a satisfied set of attributes is provided with the captured data. Similarly, when access to stored information is needed, the retrieve function is called. How this information is actually stored and retrieved is left up to the part of the application that actually extends the *StorageModule*

A *Repository* service is a service extending the basic *StorageModule* functionality. It provides a relational database and supports the storage and retrieval of any kind of data tagged with attributes. A *Repository* can be launched and left running, so that application developers can have storage performed as an existing service without additional development effort or modification. The *Repository* class can also be extended to meet a specific application need, such as storing only personal information or optimized for a specific captured data type.

The semi-persistent storage in the Personal Audio Loop application writes to file the byte array delivered to the store function of the *StorageModule* every time the *AudioCapterer* captures a chunk of audio. When information is requested, the retrieve function searches through the simple file system looking for audio files that satisfy the access query. Information matching the query is read in provided to the part of the application requesting the data. A thread is implemented to examine the files in the file system and to delete audio older than a certain amount of time (an arbitrary 15 minutes threshold limit).

## 4.3. Accessing information

Captured information can be accessed through an *AccessModule*, which supports handle, subscribe and request functions. When information is desired as it is being captured, an access interface can subscribe for information it wants; as information is captured, an *AccessModule*'s handle callback function is invoked providing that object with the captured data. A request

creates a query to the INCA runtime system consisting of attributes to be matched against stored metadata. Upon receiving this query, INCA checks with all existing *StorageModule* instances for data matching the specified query, resolves any cases of redundancy and then returns a list of data found back to the object.

The Personal Audio Loop's access interface extends an *AccessModule* and requests information as the user interacts with the graphical user interface. When the user interacts with the interface (see figure 1), she is essentially specifying a time-point at which to begin review. The user can jump back 30 seconds into the recently recorded audio stream. She can also nudge forward 7 seconds in the event of an overshoot. A query for audio information from that specified time-point forward for a minute is requested. INCA invokes the `retrieve` function in the semi-persistent storage and returns a list of matching audio chunks. The audio is then stitched together and then written to the speaker for playback.

#### 4.4. Transducing information

Just as there is a special component for capture, storage, and access, there is a *TransductionModule* for transforming information between different data types (such as from a video file to a series of image frames) and format (such as from a WAV file to a MPEG file). A *TransductionModule* instance subscribes with a list of attributes specifying the metadata for information that it can convert. When matching captured data is available, the `transduce` function of each *TransductionModule* is automatically invoked by the INCA runtime system. The transduced information is then passed on to those *StorageModules*, *AccessModules* or *TransductionModules* that have matching subscriptions for the newly generated data. Additional tagging of metadata to newly transduced data happens in a way similar to that described for the *CaptureModule*.

While not used in the Personal Audio Loop, we can envision several different *TransductionModules* that could have been used. Audio is captured as a WAV file; however, if storage space was a concern, the semi-persistent storage could have subscribed for only MPEG audio captured and a *TransductionModule* converting audio to MPEG files could have been added to the system. If relistening to audio is not appropriate, a *TransductionModule* performing speech to text recognition could have been created and instead of playback of audio through speakers, the access interface displays a transcript of the audio.

#### 4.5. Abstracted communication & coordination

For the functional parts described above to work with one another, there must be communication and coordination between them. INCA uses a *Registry* object to maintain the list of the available modules that handle the capture, storage, transduction and access of information. This object abstracts the communication between the various components of the system. In the Personal Audio Loop example, because these parts exist on the same device, a local and static *Registry* object is automatically created. In instances where different parts of the system exist on different machines, each capture, storage, access and transduction module must be directed to a remote *Registry* running at some known location.

Regardless of where the *Registry* resides, this feature allows application developers to develop each part of the application based on its function without needing to know where all the parts of the system reside and how they communicate with one another. The network communication protocol and specific IP addresses of each functional component are no longer an application issue. Thus, if changes need to be made to support different network transports, this change can occur without forcing application developers to modify the designed system's behavior.

One example of how this simplifies the programming task is seen in the relationship between an *AccessModule* and the rest of the run-time system. An *AccessModule* makes context-based queries (e.g., deliver all data owned by "Joe" originating from the location "Centennial Research Building 381" in the past week) but the application programmer does not need to know where any of this captured data resides. The runtime system of INCA resolves the query and delivers the information to the requesting *AccessModule*.

#### 4.6. Improved system reliability

In distributed systems, communication channels between different parts of the system can often terminate during network glitches, effectively removing these components from the system's distributed runtime state. In capture and access applications, the reliability of each application is crucial issue for two reasons. First, information can still be generated even though a *CaptureModule* has been removed from the rest of the system. Secondly, as a ubiquitous computing system, a large number of devices may be proliferated throughout many different physical spaces. Restarting each device and application would require much effort.

The *Registry* and all the specialized network modules are implemented with a watch-dog thread which monitors its network connectivity. When a network failure occurs, each will periodically attempt to restart itself. This mechanism would allow different parts of the capture and access system, such as a *StorageModule* to stay available

whenever there is network connectivity. Additionally each network module buffers application function calls. For example, a *CaptureModule* will buffer any information the application attempts capture while it attempts to reconnect with the Registry to deliver information to those functional parts able to consume this data.

#### 4.7. Protecting privacy concerns & more

In any applications development effort, there are many stakeholders involved. Features were included in INCA to allow designers to develop adaptable systems, for evaluators to better understand uses of the systems, and to better protect privacy for users.

First, there is an inherent concern for privacy as capture often involves the preservation of potentially important and personal information. One way to alleviate concern is to provide a way for users to understand the status of the system. Knowing what information can be captured in an environment, and what is currently capturing can allow users to adapt their behaviors in that environment.

Second, these observation services allow application developers to build context-aware capture and access applications. In knowing the potentially changing set of capture and access components available in a particular environment, it is possible to enable applications to leverage the available capture and access services.

Third, system evaluators can use observation services to keep a better log of how the system is used. The actual use of a system in an authentic setting is often different from that in laboratory settings. As ubiquitous computing systems are often built for anywhere-anytime use, it has been a problem in the past to gain an actual understanding of when the system is in use and how. This feature allows the evaluator to build an application to subscribe for changes in the status of the system in terms of when components are introduced, what kind of information it works with and how it manipulates that information.

Fourth, users may also want to control the capture and access of activities. This feature could potentially allow users to turn on or off the capture during a live experience to keep aspects of the experience private. Likewise, in a context-aware system, if a person has a preference of never having any information captured and is the only one in an environment, the system can have all capture temporarily disabled.

To support these concerns, INCA provides an *ObserveModule* a detailed description of the run-time state of the system can be gained through a `listModules` function call (which returns a list of descriptions of the existing modules). To control the behaviors of the state of any portion of the system, a

*ControlModule*'s `control` function can be invoked to specify the new state (such as stop capturing) or behavior (such as to discontinue the use of timestamp tags).

## 5. Evaluating INCA

Now that we have described and defined the key architectural features of INCA, we validate its usefulness by demonstrating that INCA:

- lowers the barrier for development;
- inspires exploration in the design space; and
- facilitates evolution of applications.

### 5.1. Lowering the development barrier

Our research is aimed at providing application developers with an infrastructure that makes it easier to build capture and access systems. To validate this claim, we have provided the infrastructure to members of our research group as well as several other research institutes—researchers new to the area of automated capture and access applications and for whom complex capture and access applications would be difficult to build. Thus far, the results are promising. These researchers are using INCA the following ways:

- INCA has been used to develop a set of reusable capture and access components [4]
- INCA has been used in the development of the next generation eClass.
- INCA has been used in the development of a synchronous meeting system that preserves artifacts generated during the meeting.
- INCA is being used to capture information generated over large-scale input surfaces.

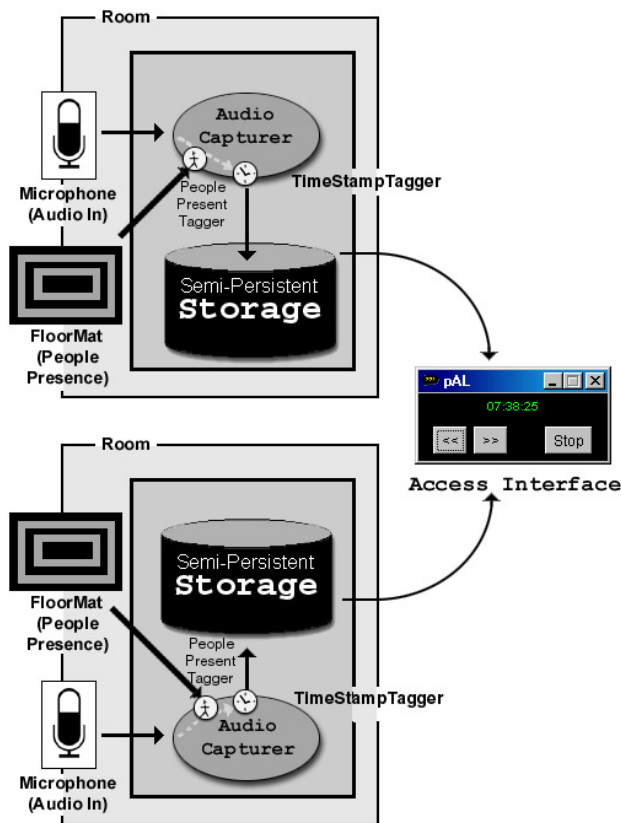
This variety of work by a number of different researchers is a promising demonstration of INCA's usefulness as a toolkit to allow many developers new to this area to be able to easily construct capture and access applications as well as to contribute to the toolkit library.

### 5.2. Enabling better design space exploration

To validate the second claim, we have mapped out the existing design space for capture and access applications and identified interesting gaps in this space that remain to be explored [31]. We have used INCA to develop several applications that explore a few of these gaps. Throughout the paper, we have already described how the Personal Audio Loop application was built using INCA. While many existing work have explored capturing information for long-term access, the Personal Audio Loop application explores the near-term capture and access of audio streams to provide users with a quick retrospective memory of recent events.

A second system we developed is WebMemex, an application built to support the continuous capture of a user's Web history. History mechanisms in standard Web browsers are impoverished, requiring users to recall hard-to-remember features of the Web page, such as its URL or title. The WebMemex service is provided through an augmented Web proxy server, resolving the problem that existing history mechanisms are local to a single machine. This work is an example of a general capture service that supports a task frequently performed everyday while providing several access services: the suggestion of related URLs that a user has seen in the past (in the spirit of the Remembrance Agent [24]); an explicit search over a user's complete Web history; and a collaborative filter to suggest relevant Web sites to friends (see [32]).

### 5.3. Facilitating application evolution



**Figure 2.** Personal Audio Loop architecture for multiple rooms. Each room is instrumented with microphones and floor mats (to identify who are present in the rooms). Each Audio Capturer captures audio that is then tagged with the name of those people present in the room and timestamped before a semi-persistent storage caches that data. Audio is then retrieved by the access interface from different storages (depending on where the user has been in the last fifteen minutes), stitched back and played as it is retrieved.

Though we found the version of PAL described throughout the paper useful in our own experience, we wanted to eliminate the need for users to carry any device. We wanted to evolve the application to offload the functionality to the environment. We will demonstrate how INCA facilitates application evolution by describing how the original PAL application is modified to meet these new requirements. The architecture of the distributed PAL application is shown in Figure 2.

Recording was distributed to a number of different rooms in our lab environment by extracting the *AudioCapturer* object from the original PAL application. Using an existing room-level indoor positioning system developed at Georgia Tech, it is possible to determine the people present during any interval of the recording. A *PeoplePresentTagger* object was created and added to the *AudioCapturer* component to automatically tag the list of names of people sensed in a location to each captured audio chunk. The effect for a single user is that her recent audio stream can be pieced back together through tagging, regardless of where she was and without having to carry around a recording device.

Many design possibilities existed with respect to how information could be stored. A single semi-persistent storage could have been used; however, we elected to keep the storage physically tied with the part of the system capturing audio. The now distributed storage still functions in the same manner as it did in the original prototype.

The access interface was on any device the user picks up from the environment. The user must specify her identity before she can use the access interface. Now, instead of just using time as the piece of information to integrate and synchronize previously captured audio, the user's name is also applied to filter for only her information. The rest of the access interface works the same as before, audio information is returned to the access interface and stitched together and then written to that device's speakers for playback.

Because the system is now distributed, an explicit *Registry* object must be running at some known location. The various capture, storage and access components direct all request to this *Registry* where their requests are resolved.

## 6. Conclusions

Many remain inspired by the visions of people like Vannevar Bush and Mark Weiser. We have entered an important stage in ubiquitous computing research. While there is still plenty of room for innovative technological advances, we are to the point where it is important to begin to understand how ubiquitous computing impacts the human experience in practice. However, serious

improvements to the tools we use are required in order for major advances in the human-centered research agenda of ubiquitous computing. We cannot observe what we cannot build and use reliably.

In this paper, we have targeted the specific challenge of building applications that support the automated capture of live experiences for later access. There is plenty of motivation to believe that this class of application can provide meaningful and useful services to end users, but there is not enough evidence to validate the claim. This is because capture and access applications are hard to build and keep operational, resulting in a limited variety of demonstrational applications and even more limited empirical observation of use.

Frameworks, toolkits, middleware solutions and infrastructures do exist to support the construction of distributed applications, mobile computing applications, and CSCW applications. However, these development facilities fail to address the specific concerns associated with automated capture and access applications. Simply allowing a printer to talk to a PC as JINI supports does not make the construction of automated capture and access applications any easier. Likewise, CSCW support typically typically caters to building synchronous, distributed applications. The design of INCA was greatly influenced by our own development of a CSCW capture system [25] for which existing CSCW toolkits were not helpful.

Because of a lack of appropriate existing support for building this class of ubicomp application, we have researched and identified key architectural insights into the creation of capture and access applications. Designing in terms of these architectural features—capture of attribute-tagged data streams, storage, transduction and access of related and integrated streams—allows a designer to focus on key distinguishing features of any capture and access application. We introduced the INCA toolkit as a means of transforming high-level designs into implementations that hide details of distribution and data from the programmer. We explained how an application can be built and evolved using INCA to demonstrate how such a tool can lower the barrier to entry into this domain, and facilitate creative and iterative exploration within a large and complex design space.

## 8. References

1. Abowd, G.D., *Classroom 2000: An experiment with the instrumentation of a living educational environment*. IBM Systems Journal, 1999. **38**(4). pp. 508-530.

2. Abowd, G.D. Software Engineering Issues for Ubiquitous Computing. In the *Proceedings of The 21st International Conference on Software*

*Engineering (ICSE'99)*. (May 16-22, Los Angeles, CA), 1999, pp. 75-84.

3. Bacher, C., *et al.* Authoring on the Fly. A New Way of Integrating Telepresentation and Courseware Production. In the *Proceedings of ICCE'97*. (December, Kuching, Sarawak, Malaysia), 1997.
4. Baldochi, L.A.B., Cattelan, R.G., and Pimentel, M.G.C. Automatic Generation of Capture and Access Applications. In the *Proceedings of SBMIDIA 2002*. (October 7-10, Forteleza, CE, Brazil), 2002.
5. Brotherton, J.A. *Enriching Everyday Activities through the Automated Capture and Access of Live Experiences - eClass: Building, Observing and Understanding the Impact of Capture and Access in an Educational Domain*. Ph.D. Thesis, College of Computing, Georgia Institute of Technology, 2001.
6. Brotherton, J.A., Abowd, G.D., and Truong, K.N. *Supporting Capture and Access Interfaces for Informal and Opportunistic Meetings*. Georgia Institute of Technology Technical Report GIT-GVU-99-06, 1998.
7. Bush, V., *As We May Think*, in *Atlantic Monthly*. 1945. p. 101-108.
8. Chiu, P., *et al.* NoteLook: Taking Notes in Meetings with Digital Video and Ink. In the *Proceedings of ACM Multimedia'99*. (October 30-November 5, Orlando, FL), 1999, pp. 149-158.
9. Chiu, P. and Truong, K.N. Interactive Space-Time Planes for Document Visualization. In the *Extended Abstracts of InfoVis'02*. (Boston, MA), 2002.
10. Cruz, G. and Hill, R. Capturing and Playing Multimedia Events with STREAMS. In the *Proceedings of ACM Multimedia'94*. (October 15-20, San Francisco, CA.), 1994, pp. 193-200.
11. Davis, R.C., *et al.* NotePals: Lightweight Note Sharing by the Group, for the Group. In the *Proceedings of CHI'99*. (May 15-20, Pittsburgh, PA), 1999, pp. 338-345.
12. Deitz, P. and Yerazunis, W. Real-Time Audio Buffering for Telephone Applications. In the *Proceedings of UIST'01*. (November 11-14, Orlando, FL), 2001, pp. 193-194.
13. Dey, A.K. *Providing Architectural Support for Building Context-Aware Applications*. Ph.D. Thesis, College of Computing, Georgia Institute of Technology, 2000.
14. Dey, A.K. and Abowd, G.D., *A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications*. Human-Computer Interaction (HCI) Journal, 2001. **16**(1-3).
15. Dey, A.K., *et al.* The Conference Assistant: Combining Context-Awareness with Wearable Computing. In the *Proceedings of ISWC'99*. (October 20-21, San Francisco, CA), 1999, pp. 21-28.

16. Hindus, D. and Schmandt, C. Ubiquitous Audio: Capturing Spontaneous Collaboration. In the *Proceedings of CSCW'92*. (October 31-November 4, Toronto, Canada), 1992, pp. 210-217.
17. Mankoff, J.C. *An architecture and interaction techniques for handling ambiguity in recognition-based input*. Ph.D. Thesis, College of Computing, Georgia Institute of Technology, 2001.
18. Mankoff, J.C., Hudson, S.E., and Abowd, G.D. Interaction techniques for ambiguity resolution in recognition-based interfaces. In the *Proceedings of UIST'00*. (November 5-8, San Diego, CA), 2000, pp. 11-20.
19. Minneman, S. and Harrison, S. Where were we: Making and using near-synchronous, pre-narrative video. In the *Proceedings of ACM Multimedia'93*. (August 1-6, Anaheim, CA), 1993, pp. 207-214.
20. Minneman, S., et al. A confederation of tools for capturing and accessing collaborative activity. In the *Proceedings of ACM Multimedia'95*. (November 5-9, San Francisco, CA), 1995, pp. 523-534.
21. Moran, T.P., et al. "I'll Get That off the Audio": A Case Study of Salvaging Multimedia Meeting Records. In the *Proceedings of CHI'97*. (March 22-27, Atlanta, GA), 1997, pp. 202-209.
22. Mukhopadhyay, S. and Smith, B. Passive Capture and Structuring of Lectures. In the *Proceedings of ACM Multimedia'99*. (October 30-November 5, Orlando, FL), 1999, pp. 477-487.
23. Pedersen, E.R., et al. Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings. In the *Proceedings of ACM INTERCHI'93*. (April 24-29, Amsterdam, The Netherlands), 1993, pp. 391-398.
24. Rhodes, B.J. *Just-In-Time Information Retrieval*. Ph.D. Thesis, Media Laboratory, MIT, 2000.
25. Richter, H., et al. Integrating Meeting Capture within a Collaborative Team Environment. In the *Proceedings of UbiComp 2001*. (September 31-October 2, Atlanta, GA), 2001, pp. 123-138.
26. Richter, H., Schuchhard, P., and Abowd, G.D. *Automated capture and retrieval of architectural rationale*. Georgia Institute of Technology Technical Report GIT-GVU-98-37, 1999.
27. Stifelman, L.J. *The Audio Notebook*. Ph.D. Thesis, Media Laboratory, MIT, 1997.
28. Streitz, N.A., et al. DOLPHIN: Integrated Meeting Support across Liveboards, Local and Remote Desktop Environments. In the *Proceedings of CSCW'94*. (October 22-26, Chapel Hill, NC), 1994, pp. 345-357.
29. Truong, K.N. and Abowd, G.D. StuPad: Integrating Student Notes with Class Lectures. In the *Proceedings of CHI 1999 Extend Abstracts*. (May 15-20, Pittsburgh, PA), 1999, pp. 208-209.
30. Truong, K.N., Abowd, G.D., and Brotherton, J.A. Personalizing the Capture of Public Experiences. In the *Proceedings of UIST'99*. (November 7-10, Asheville, NC), 1999, pp. 121-130.
31. Truong, K.N., Abowd, G.D., and Brotherton, J.A. Who, What, When, Where, How: Design Issues of Capture & Access Applications. In the *Proceedings of UBICOMP 2001*. (September 31-October 2, Atlanta, GA), 2001, pp. 209-224.
32. Truong, K.N., Abowd, G.D., and Pimentel, M.G.C. *Vicariously Sharing Captured Web Experiences through an Automated Recommendation System*. Georgia Institute of Technology Technical Report GIT-GVU-02-20, 2002.
33. Weber, K. and Poon, A. Marquee: A tool for real-time video logging. In the *Proceedings of CHI'94*. (April 24-28, Boston, MA), 1994, pp. 58-64.
34. Weiser, M., *The computer for the 21st century*. Scientific American, 1991. **265**(3). pp. 94-104.
35. Whittaker, S., Hyland, P., and Wiley, M. Filochat: Handwritten notes provide access to recorded conversations. In the *Proceedings of CHI'94*. (April 22-28, Boston, MA), 1994, pp. 271-277.
36. Wilcox, L., Schilit, B.N., and Sawhney, N. Dynamite: A Dynamically Organized Ink and Audio Notebook. In the *Proceedings of CHI'97*. (March 22-27, Atlanta, GA), 1997, pp. 186-193.