INCA: A Software Infrastructure to Facilitate the Construction and Evolution of Ubiquitous Capture & Access Applications

Khai N. Truong and Gregory D. Abowd

Georgia Institute of Technology
College of Computing & GVU Center
Atlanta, GA 30332-0280, USA
{khai, abowd}@cc.gatech.edu

Abstract. People's daily lives provide them with memories and records that they often want to review later. They must expend time and effort to record these experiences manually for future retrieval. To address this issue, applications that automatically capture details of a live experience and provide future access to that experience have become an increasingly common theme of research in ubiquitous computing. In this paper, we present our experience building a number of capture and access applications, sharing insights on the successes and difficulties we encountered. These lessons inform the design of the INCA toolkit (Infrastructure for Capture and Access), which supports the construction of applications in this class. We will demonstrate how INCA facilitates the rapid prototyping and simplified evolution of increasingly complex capture and access applications.

1 Introduction

There are many everyday examples of people capturing information for later use. People often take pictures to capture a moment or write notes to record the important information from an experience. Reliance on manual methods of capturing information is not foolproof; people often fail to capture necessary details in a timely fashion. Many are not good at creating accurate records of an experience; as a result, these records are often biased, incomplete, and in some cases may even contain errors. The act of manual capture can distract people from fully engaging in the experience.

Increasingly, researchers are applying ubiquitous computing technology to capture details of a live experience automatically and to provide future access to those records. Automated capture and access applications leverage what computers do best —record information. In return, humans are free to fully engage in the activity and to synthesize the experience, without having to worry about tediously exerting effort to preserve specific details for later perusal.

There are many examples of automated capture and access applications, but they have explored only a few domains, such as the classroom, meetings, and other generalized experiences. Though there are significant social and cultural barriers that dictate against a world of continuous capture, experience shows many limited situations in which the value of capture can outweigh its cost. Therefore, continued rich exploration is appropriate, especially in a research context. Unfortunately, many new capture systems simply revisit ideas already explored in earlier work, and existing applications typically are not leveraged as platforms for further investigation because of the challenges of managing and evolving them. These issues also have prevented most researchers from evaluating their prototypes under authentic use and then modifying them to include interesting functionalities according to continual feedback from the user population.

To facilitate research in ubiquitous computing, advances are necessary to improve the tools we provide ourselves and other creative designers who wish to improve upon the vision of Mark Weiser. Many researchers have begun to provide such support for the development of physical [12], tangible [15], and smart devices/applications [11] and the collaboration between heterogeneous devices [28]. Previously, we created tools to support context-aware computing [9] and human-assisted error correction resulting from recognition-based interfaces [19]. In these previous cases, the common 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 an infrastructure/toolkit that embodies these abstractions and then validate the abstractions, architecture and toolkit by developing interesting and complex applications within the design space and exploring critical issues for deployed applications.

Using this research method, we introduce the Infrastructure for Capture and Access (INCA) toolkit, which encourages a simplified model for designing, implementing and evolving capture and access applications. We validate this infrastructure by demonstrating how it addresses design challenges, and more importantly how it supports the evolution of increasingly complex capture and access applications.

2 Related Work: The Capture & Access Design Space

We surveyed many of the existing and past projects that support the capture and access of various experiences (for a full review, consult [30]). This body of work can be organized based on the main domains/areas that have been explored: the classroom, meetings, and other generalized experiences.

2.1 Capture & Access in the Classroom

Classroom capture systems have experienced much success because they automatically record the activities of the instructor so a large number of students can directly benefit from the work. The eClass/Classroom 2000 system [1,5], the Cornell Lecture Browser [20], MANIC [24], AutoAuditorium [4], STREAMS [7], Authoring on the Fly [2], and work from Microsoft Research [13,17] all capture with varying degrees of automation significant streams of information presented during the lecture. Commonly captured streams include the instructor's presentations, audio, video, ink written on a physical or electronic whiteboard, visited Web pages, and arbitrary program executions. Access to these notes is typically provided through a Web interface that integrates the various captured streams and allows users to index into specific portions of the audio or video of the live experience.

A small number of projects support the capture of personal notes during lectures. The Audio Notebook [25] is an augmented paper notebook that records and integrates audio with ink written in the notebook. StuPad [29] and DEBBIE [3] are systems that use video display tablets to examine the integration of public lecture notes with private annotations. NotePals [8] is an example of a collaborative access system in which different users take separate notes during the experience and those notes are then merged during the access phase with the separately captured public presentation; this sharing can enable users to easily recognize the important points presented during class (as observed by multiple users at a time).

2.2 Capture & Access in Meetings

Meeting capture also has been a frequent subject of research in capture and access. Like the classroom domain, many meeting systems provide similar capabilities for reviewing presentations given during a meeting. Most support the public capture of meetings. Some of these systems support the collaborative capture of information using a shared whiteboard that a group of users may place and interact with artifacts. Examples of this class of system are DOLPHIN [26], TeamSpace [23] and Tivoli [22]. Dynomite [33] and FiloChat [32] are systems that support the recording and integration of personal notes with audio or video streams of the meeting. The NoteLook system [6] provides users control of an array of cameras to grab the images of the meeting they wish to store in their personal notebook and perhaps annotate.

2.3 Capture & Access in Other Environments

The potential benefits of capture and access have been in a few other settings such as offices, conferences, and museums. The Forget-Me-Not application [16] was perhaps the first to demonstrate the continuous capture of information for a user as she moves about an instrumented capture environment, the office, exhibiting the use of capture and access as a general memory prosthesis. This concept of personal mobile capture in a sensor-rich environment has been revisited in a number of recent applications. The Conference Assistant [10] allows the user to capture personal notes using a mobile device at a conference. Her notes are later integrated with content publicly captured based on her automatically sensed location information. The Comic Diary [27] automatically generates a comic strip recounting the conference attendee's experience based on sensed and manually inputted content. Similarly, the HP Remember [14] system allows a museum visitor to author an automatically generated Web page recounting her experience through both sensed and manually added content. A museum visitor is provided the ability to control cameras for capturing images of her during a museum experience is much like how a NoteLook user can specify what image from a meeting to include in her notes.

2.4 Summary

This review of research systems shows that although there are many existing and past projects in capture and access, only a small number of domains have been explored. Software products distributed with the Mimio (http://www.mimio.com) and Silicon Chalk (http://www.silicon-chalk.com/) reveal the same domains are investigated commercially as well. Despite the number of research and commercial efforts, there has been relatively little innovation in the past 5 years. Many new applications simply revisit ideas that have been previously explored. Furthermore, there has been relatively little research contribution in the way of understanding/evaluating these systems under authentic use, with the notable exception of Tivoli and eClass.

3 Lessons Learned from Classroom Capture & Access Systems

In 1995, we began our investigation of the automated capture of live university lectures so that students and teachers may later access them. In this section, we present the lessons learned from the successes and difficulties we experienced building these applications and evolving them to include necessary (and potentially complex) behaviors over the course of a longitudinal study of use.

3.1 eClass: A Successful Motivation for Classroom Capture and Access

The eClass project (formerly known as Classroom 2000) was an experiment in which we created a classroom environment to capture 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 different machine running a proxy server to log HTTP requests recorded the Web pages visited by the instructor during lecture. Finally, a separate machine recorded the audio inside the classrooms.

A central server connected to each of these capture services to provide coordination for each lecture, or capture session. This coordination included the collection of prepared materials prior to a lecture, initiating and terminating the recording for all services for a given lecture, and the integration and post-production of all captured materials to create the Web-accessible notes. The application required some initial set up and maintenance, which included the specification of all machines involved before runtime. However, the system succeeded largely because this coordination was transparent to the users, requiring very little extra instructor or student effort.

Over time, requests from users (both teachers and students) resulted in several changes to the system, including an extended whiteboard application (*i.e.*, additional display surfaces showing the history of a lecture), video capture, and a database of captured lectures to support server-side, dynamically generated Web notes and a search function. This evolution in eClass was possible largely because initially we adopted a structure to the capture problem that separated concerns into four 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 us to evolve the prototype to include the improved capabilities described above as small isolated changes to the software.

3.2 StuPad: Challenges in Extending eClass with Personal Capture & Access

One goal of eClass was to relieve students from the tedious task of copying notes during class. However, we observed that some students continued to take small amounts of notes on paper. To support the integration of each student's notes with the eClass notes, we added the Student Notepad (StuPad) system to the existing eClass system [29]. 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 added personal annotations.

Unfortunately, certain aspects of the existing eClass system lead to challenges implementing StuPad. Although the structuring of eClass into four phases facilitated much evolution to the system, these extensions to the system resulted in inconsistencies between how the eClass server communicated with the different clients —further weakening a communications scheme that already was not multithreaded. This communications structure prevented us from implementing StuPad in the most obvious way, where each student notepad directly obtains the various information streams in the classroom. Ultimately, we were able to create a working StuPad system, but it required redistributing the different data streams in a non-uniform manner that was considerably more difficult than anticipated.

Despite the student motivation to integrate their in-class personal notes with the public capture of eClass, StuPad turned out to be a less useful application than expected. When study occurred outside of class, additional note taking remained difficult for students to integrate with the captured notes. At that time, the IBM CrossPad presented an affordable solution that allowed one to work with pen and paper while also capturing an electronic record. Such a platform would have enabled students to capture inside or outside of the classroom. However, the eClass system stored captured information in a rigid hierarchical structure that consisted of course numbers, terms, and dates of the lectures. Records captured using the CrossPad required a more flexible organizational scheme, so forcing them to fit inside the same directory structure was not a logical solution. Furthermore, the storage scheme employed by eClass suggested specific ways information is accessed; students could identify the course and then the particular lecture date that they wish to review. Personal notes created during a lecture easily can be synchronized with the notes captured by eClass, but notes created outside of a lecture pertaining to topics addressed in lecture needed to be integrated as well. Providing flexible methods for reviewing the captured data proved to be a second difficult challenge, requiring the integration of the private notes and the classroom notes to happen through other contextual relationships beyond temporal synchronization.

3.3 Lessons Learned

The StuPad project eventually ended because the hurdles described above overwhelmed the development efforts. Through many makeshift solutions, we were able to avoid risky architectural changes involved in directly addressing the problems presented by the underlying communication structure of eClass. However, this issue could have been avoided if the essential application features were decoupled from this concern, making the system easier to build and extend. The rigid hierarchical data structure employed by eClass resulted in storage and access challenges that were too difficult to overcome. We learned the importance of information storage being flexible in order to support a growing set of captured information as the system continues to evolve. Information integration also occurs due to different contextual relationships between captured streams beyond just time. Both lessons may seem obvious, but in the case of eClass, they were overlooked as key design issues.

A primary reason why our classroom research has relative success was because we could evolve the system over a long period of evaluation. This was aided considerably by early architectural decisions we made to structure the system into four phases: pre-production, live capture, post-production/integration, and access. However, the phases of eClass imply a sequential ordering of activities that does not always happens. Instead, it is generally better to consider the functional components of the overall architectural solution. We also observed that post-production/integration activity can be further separated into: storage of information until it is later accessed; transduction (or transformation) between different data types; and integration, in which relationships between separately captured streams cause the multiple streams to be delivered collectively during access. Additionally, access happens on varying time-scales, depending on when information needs to be reviewed relative to when it was captured; therefore, different forms of access methods and interfaces are desired.

4 INCA: Infrastructure for Capture & Access

To help designers focus the development effort on the essential features of the capture and access application, we developed the INCA toolkit with a small set of key architectural abstractions in mind.

- Part of the system is responsible for the **capture** of information as streams of data that are tagged with relevant metadata attributes.
- 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;

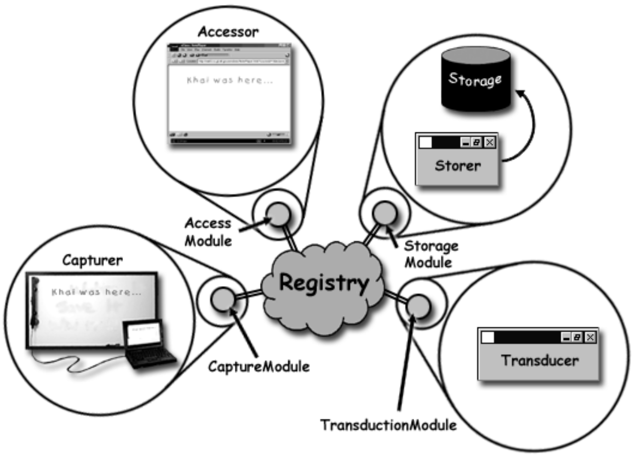


Figure 1. General architecture for systems built using INCA. A *Registry* runs at some well-known location and any number of applications acting as *Capturers, Accessors, Storers,* or *Transducers* can connect to it and share captured information through instances of specialized networked modules (such as a *CaptureModule, AccessModule, etc.*).

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.

For any given application, there may be more than one instance of each of the above functions. From the implementation perspective, INCA provides a direct way to translate applications designed the following way into executable form. For each functional component above, INCA provides an encapsulated module that a programmer extends or uses as part of the application code. We next present the different components within the infrastructure. There are additional features of INCA that simplify other aspects of application development that stem from the inherent distributed nature of these applications, and common data types and features that allow programmers and end-users to inspect and control the run-time system.

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. 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.

The `capture` function defined in the *CaptureModule* is invoked when the application attached to a device (such as a camera, microphone, or electronic whiteboard) has data that is available to be tagged and stored, transduced or provided to some access service. A *CaptureModule* can register various *Tagger* objects to add metadata information automatically to the output objects from the `capture` function.

INCA provides a number of reusable *Tagger* objects in its toolkit library for adding attributes specifying people present in a location, the current time, the data type being captured and the location of the captured activity.

4.2 Storing Information

A *StorageModule* provides persistence for captured data. A *StorageModule* can specify a list of attributes for the kind of captured information it is interested in automatically receiving via a `subscribe` function. 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, a `publish` function announces to other components an attribute list describing the kind of information it stores. 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 defines all 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 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 *Repository* provided by INCA uses MYSQL as the back-end database. A *FileRepository* service is also available and provides the same functionality as the *Repository* object without using MYSQL.

4.3 Accessing Information

An *AccessModule* supports `handle, subscribe` and `request` functions. When information is desired as it is being captured, an access interface can `subscribe` for information it wants (*e.g.*, subscribe for all data created by "John" originating from "Building 4: Room 106"). As information is captured, an *AccessModule*'s `handle` callback function is invoked providing that object with the captured data. A `request` creates a context-based 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* and *Repository* instances for data matching the specified query (by invoking the `retrieve` function in the storage components) to obtain all matching data, resolves any cases of redundancy and then returns a list of data found back to the object. Information is integrated based on how it is requested through context-based queries. In its simplest form, the query match is based on attribute name-value pairs and can grow to include more general data retrieval operations that more effectively filter and mine large distributed repositories.

One example of INCA simplifying the programming task is seen in the relationship between an *AccessModule* and the remaining run-time system. An *AccessModule* makes context-based queries for information, but the application programmer does not need to know the location of any of this captured data. The INCA run-time system resolves the query and delivers the information to the requesting *AccessModule.*

4.4 Transducing Information

A *TransductionModule* supports the transformation of information between different data types (such as from a video file to a series of image frames) and formats (such as from a WAV file to an MP3 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.* INCA provides a number of *Transducer* services, such as transforming a video file into a series of image frames and vice versa, transcribing handwritten ink, or converting text to speech.

4.5 Additional Abstractions & Features

The previous features of INCA provided abstractions meant to guide a designer's thinking about how to create a specific capture and access application. In addition to those essential application features, there are a number of other concerns that INCA supports to simplify the development and evolution process:

- Attribute-triggered automated garbage collection allows the system to discard unwanted data.
- An *ObserveModule* provides a detailed description of the run-time state of the system. A *ControlModule* allows for the modification of this run-time state. Together, these features could allow for implementation of privacy and security features, instrumentation for extended evaluation purposes and dynamic adaptation of application features.
- An extensible library of reusable components supports the capture and access common data types, currently including audio, video, ink and Web visits.

As seen with eClass, the communications structure of a system cuts across all the architectural concerns of a capture and access application and is an important factor in evolving these applications. We implemented a network abstraction layer to separate network concerns from the application code. This layer supports a general client-server architecture, where the server binds to two ports. The server uses one of the ports to support connections for synchronous communication between the clients and the server. The second port is used to support asynchronous communication. We build the four functional modules described previously (the *CaptureModule*, *AccessModule*, etc.) as clients in this architecture. A *Registry* object is built as a server maintaining a list of the available modules that handle the capture, storage, transduction and access of information (see Figure 2). The *Registry* and all the specialized network modules are implemented with a watchdog thread which monitors its network connectivity, increasing reliability and self-maintenance.

To support a variety of applications designed for different domains, the server and it clients exchange serialized message objects. By viewing captured data as only raw bytes with tagged attributes, the infrastructure is able to handle many different kinds of data in the same fashion.

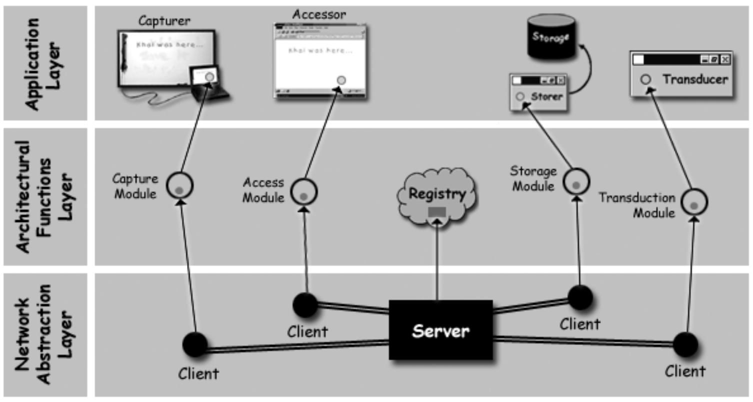


Figure 2. INCA includes a network abstraction layer on which the specialized networked modules (such as the *CaptureModule, AccessModule, etc.*) and the *Registry* are built. Developers create applications without worrying about details of the underlying network code.

5 Building a Simple Capture & Access Application with INCA

Now that we have defined the key architectural abstractions and other useful services provided by INCA, we will demonstrate how this infrastructure can be used to develop a simple audio capture application that supports the access of near-term recorded conversations (shown in Figure 3). This Personal Audio Loop application (PAL) is intended to run on a single, mobile device that travels with its user. Unlike a tape recorder, this service continues to capture audio even when playback of previously recorded information is accessed. Furthermore, the application automatically discards portions of the captured audio that are older than fifteen minutes.

To begin, we instantiate a *Registry* component in our main program:

```
Registry registry = new Registry();
```

Capture behavior

We use a predefined *WaveCapturer* component, an extended *CaptureModule* from INCA, to capture audio and register *Tagger* objects to add attributes facilitating the future retrieval of the captured audio (e.g. time stamps). Once initialized, we start the *WaveCapturer*.

```
WaveCapturer wave_capturer = new WaveCapturer();
CaptureModule capture_module = new CaptureModule(wave_capturer);
```

```
capture_module.addTagger(new TimeStampTagger());
wave_capturer.startCapture();
```

Storage behavior
To store the audio, we simply instantiate a *Repository*.

```
Repository repository = new Repository();
```

Access behavior
We use a predefined *AudioPlayer* component, an extended AccessModule from INCA, to play back requested audio.

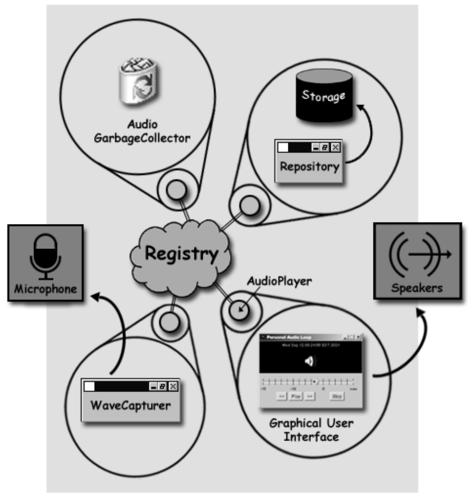
```
AudioPlayer audio_player = new AudioPlayer();
```

We developed a GUI for the user to specify the time in seconds, *t*, back from the present, at which audio should begin playback (see Figure 3). We developed our application to request 1 minute of audio from that point for playback.

```
Query q_start_time = new Query();
q_start_time.greaterThan(new Attribute("TimeStamp",new Long(t).toString()));
Query q_stop_time = new Query();
q_stop_time.lessThan(new Attribute("TimeStamp", new Long(t+(1*60*1000)).toString()));
Query q_main = new Query();
q_main.and(q_start_time);
q_main.and(q_stop_time);
audio_player.playback(q_main);
```

To discard audio, we create a special *GarbageCollector* object that periodically discards information.

```
class AudioGarbageCollector extends GarbageCollectionModule implements Runnable {
   protected Thread thread;
   public AudioGarbageCollector() {
      thread = new Thread(this);
      thread.start();
   }
   public void run()   {
      while(true) {
         try {
            Thread.sleep(1000 * 60);
            Query q_time = new Query();
            q_time.lessThan(new Attribute("TimeStamp",
```



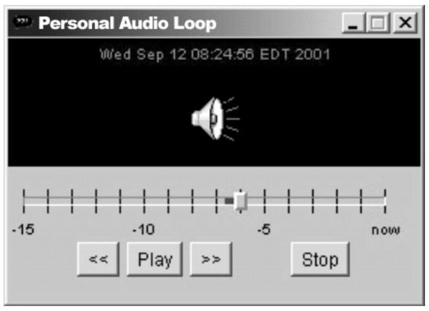


Figure 3. The Personal Audio Loop application's high-level architecture (left) and user interface (right).

```
               new Long(System.currentTimeMillis() - (15 * 60* 1000)).toString()));
             gc(q_time);  // remove data older than a certain time from storage
           } catch(Exception e) {}
         }
       }
    }
```

6 Uses of INCA

We now present three more complex systems built with INCA and evolved to investigate interesting issues and new features. Finally, we discuss applications built by others using INCA as evidence to support the general applicability of this toolkit and its value to the ubiquitous computing community.

6.1 Building & Evolving a Classroom System

The foundation of the classroom capture and access experience is that of the public information available before, during, and after the actual lecture. A minimally useful system must capture the instructor's writing and prepared notes, the instructor's speech while communicating with the class, and outside information brought into the lecture in the form of visited web pages. Furthermore, all of this raw data must be stored in a logical manner for easy access at a later time. When users choose to access the information from any particular lecture, they must be able to do so from any other location and using a variety of techniques.

To support the capture of the instructor's writing, we developed a custom application, known as e-Board (see Figure 4a) that is installed on an electronic whiteboard in the classroom. For instructors who teach with prepared presentations, the e-Board application allows users to load a presentation for display on the electronic whiteboard. e-Board is built with a *BoardSurface* component, a specialized *CaptureModule. BoardSurface* provides a blank writing surface that listens to pen events and also displays a slide image. As the instructor creates a new blank slide or chooses an existing slide to present, *BoardSurface* captures that slide and automatically tags it with a unique ID (a string containing the name of the machine and the time the slide was loaded or created in the classroom capture and access system). Similarly when an instructor visits a slide, *BoardSurface* captures a visit event tagged with that slide's unique ID. *BoardSurface* also tags ink strokes captured during the slide's presentation with the slide's unique ID. Tagger objects, registered by e-Board, automatically associate time, physical location (*e.g.*, classroom number), relevant course information (*e.g.*, class name and instructor name), and relevant system information (*e.g.*, application name and IP address of the machine running the application) to the captured ink strokes, slides, and slide visits. *BoardSurface* is distributed with INCA, so other developers can use and extend it as desired.

INCA includes *WaveCapturer*, a reusable *CaptureModule* specialized for capture of low-bandwidth audio in the .WAV format. An instance of *WaveCapturer* is installed on a suitable machine attached to a recording device in a particular space (a classroom, in this case). The INCA *Registry* is informed automatically of this instance. The e-Board application uses an *ObserveModule* to determine the list of available nearby capture services and a *ControlModule* to start those services. Starting and stopping the audio recording can be coupled with the starting and stopping of the e-Board application. It also can be tailored to start and stop recording at different times during the use of e-Board, either automatically or through human intervention.

WebMemex is a specialized *CaptureModule* provided by INCA to record Web pages requested by a client browser. Acting as a Web proxy, it handles the HTTP requests and logs pages visited. INCA provides this Web proxy application as a general capture service. In eClass, *WebMemex* helps capture pages viewed in class.

All captured classroom activity is stored in the property-based *Repository*. The access application runs as a standard Web interface (see Figure 4e) allowing users the freedom to access classroom data from any web-enabled machine. The application, composed of custom built Java Server Pages (JSP), instantiates an *AccessModule* to request captured information tagged with the course name and date specified by the student reviewing the lecture. It then presents the lecture as a sequence of discrete slides in the same order used by the instructor during the lecture, regardless of what ordering might have existed in the prepared slides. Users can examine a timeline that indicates the important events taking place during the lecture, such as a slide being created or visited or a Web page being viewed. For further details, a user may click

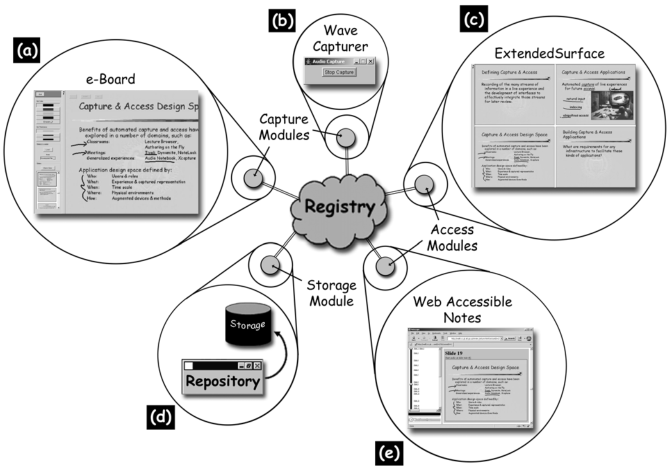


Figure 4. The simplified architecture of the basic eClass system. (a) The e-Board application built using the *BoardSurface* capture module provides a writable presentation surface that allows the instructor to present prepared slides and/or write on a blank surface. (b) A separate component built using the *WaveCapturer* capture module records audio during the lecture. (c) The *ExtendedSurface* application shows a history of slides captured during the lecture. (d) A Web access interface that includes an audio player, a timeline of the captured lecture, and the slides and their annotations automatically generate using JSP allows students to review the captured lecture. (e) A generic *Repository* object stores all the captured data.

any portion of the timeline or the handwritten ink to begin playback of the audio corresponding to that portion of the lecture. In this case, the *AudioPlayer*, a toolkit component for playing back audio (using an *AccessModule*) requests audio chunks and begins playback until the user clicks the stop button.

Playback of the ink stream at variable speed is also a desirable application feature [21]. INCA supplies this functionality with a reusable *NotesPlayer* component that provides an interface for requesting captured slides and ink annotations and methods for invoking and stopping playback. Using an *AccessModule*, the *NotesPlayer* object requests the information specified by the user. INCA delivers information back to the *NotesPlayer* in time sequential order allowing custom rendering over time. A *Clock* object is used to control the playback of the captured notes in the NotesPlayer. This *Clock* object can be paused or even programmed to run at different rates. The *AudioPlayer* and the *NotesPlayer* can share a *Clock* object to synchronize playback of the captured notes augmented with audio.

A useful near-term access application in the classroom to both students and teachers is an extended whiteboard to show the history of the lecture. INCA provides a reusable *ExtendedSurface* component. This specialized *AccessModule* provides an interface supporting the subscription queries for captured slides and ink annotations. When slides are created or visited, they are added to the surface. The surface can be defined to show the current slide, the previous slide, the current and previous three slides or the previous four slides (see Figure 4c). The surface displays not only the slides but any ink annotations of the slides as well.

A potential evolution of the system is to change the technique to capture presentation slides and ink annotations. While most of the related public capture systems support the explicit capturing of information through electronic whiteboards, the Cornell Lecture Browser demonstrates the ability to do this capture with cameras and vision techniques [20]. INCA supports this behavior through a component that frequently captures frames from a camera or display signal to a projector. This component uses an algorithm that performs image differencing on specified regions of the frames representing the actual presentation slides. By scaling the captured frames to 1/8th of their original resolution and computing the percent of pixels that are different between the two frames, this component is able to determine a new slide or a slide visit event. The separation of concerns supported by INCA minimizes the impact of the change in the underlying capture technique to the rest of system.

Not all instructors have electronically prepared presentation slides. As a result, the hurdle for using eClass can be lowered with support for scanning materials and making it available on a blank slide that an instructor could then annotate. This feature is also desirable at times during class when illustrations or examples are written on paper and needs to be integrated with the rest of the captured electronic lecture notes. We developed a custom application, *eScanner* to run on the computer connected to a scanner. This application continuously scans material until a blank image is detected. As images are scanned in, they are captured and made available to any application interested in it. This application publishes that it is a scanner application and it is located in a particular room. The e-Board application is modified to use an *AccessModule* and subscribes for slides created in the classroom. We added to the e-Board application a GUI button that activates the loading of slides from the scanner. The e-Board application uses an *ObserveModule* and *ControlModule* pair to request the scanning of any material the instructor has placed on the scanner.

Finally, most instructors occasionally want the ability to suspend audio recording. We modified the e-Board application to include in its interface a button to stop and resume audio recording when it observes than an audio capture service is available. The button invokes the *ControlModule* to control audio capture in this way.

6.2 Capturing & Sharing Web Experiences

To facilitate a content and/or context based history search mechanism, we used the *WebMemex* component to capture an annotated Web history. We registered a number of existing and custom *Tagger* objects to help associate the user's ID, time and location to each captured Web visit (in addition to the keywords). The access interface consisted of custom Java Server Pages that used an *AccessModule* to handle search queries for previously captured Web visits.

This annotated Web history also enables a number of other access features, such an automatic recommendation capability. As a user browses new Web pages, a different access application suggests related URLs that the user has visited from the past. An *AccessModule* requests the last Web visit handled by the proxy server (which is currently viewed by the client browser). By taking the keywords of this Web page, the access application can query for previous URLs that she visited with matching keywords. This information is displayed in a pop-up window.

Although we originally developed this service to support individual users as they surf the Web, it has since been extended to investigate the sharing of Web histories within a social network [18]. We developed a component that communicates with Yahoo's Messenger service to authenticate the user's login and obtain her list of friends, which we considered the user's social network. This component simply replaced the less sophisticated user authentication component we previously used in the system; *i.e.*, this modification happened as an isolated change from the rest of the capture and access application. We then modified the access behavior to allow information sharing between users if they exist in each other's buddy list.

6.3 Recording & Analyzing Developmental Behavioral Patterns

To better support the collection and analysis of developmental behavioral data of children with autism or other disorders, we developed a mobile capture application that runs on a Tablet PC and integrates the notes taken during an observation session with the corresponding video clip automatically captured [31]. Using an *InkSurface* component, which the *WhiteBoard* component used in the classroom system extends, we recreated the paper forms used by the teachers. We added the ability to tag the markings the teachers create on this form with their semantic meaning (such as if a behavior is observed, not observed, *etc.*) by recognizing the gestures. This electronic form includes a *VideoCapturer* component that automatically records video clips during a session. After each session, the captured information is stored and then made available again when the teacher reviews the data with the parents. We used an

AccessModule to query for all behaviors captured over time based on their tagged values (if a behavior is observed, not observed, *etc.*). This data is color-coded and then plotted on two timelines to provide a macro view of all the behaviors observed by the teacher and a micro view that shows details of a particular session. Clicking on marks in the micro timeline causes a *VideoPlayer* component embedded in the access interface to access a video clip of that behavior.

By iteratively designing this application with members of the Emory Autism Centers, we uncovered major usability problems in the capture application. Writing on a Tablet PC was too different from writing on paper in two important ways: calibration and resolution. Furthermore, imperfect gesture recognition resulted in too much time and effort spent correcting the data. As a result, we needed to modify the capture interface. We replaced the *InkSurface* components found in the capture interface with buttons that users can push to specify an observed behavior. This event is captured and tagged with the same semantic meaning as the strokes were before.

6.4 Other Uses of INCA

INCA was developed as part of a joint research effort between the Georgia Institute of Technology and Universidade de São Paulo (USP). We provided INCA to developers at our own institution and at the partner university. In addition to the applications described above, developers at Georgia Tech used INCA for the following projects:

- An e-Board application for another department on campus.
- A very large-scale input surface, covering two entire walls of a meeting room using six chained mimio recording devices.
- A video recording application that automatically captures and tags home videos based on room-level entry and exit of individuals in the home.

At the partner university, Universidade de São Paulo, the following uses of INCA were reported:

- The iClass system (http://iclass.icmc.usp.br) to support the capture of lectures and seminars in order to generate a varied of web-based multimedia documents.
- An application to facilitate exchange of notes between a Palm-based PDA and the normal electronic whiteboard.
- A component for Web capture that could do specialized processing on the content of the URL. The same student built an ink capture module linked with a handwriting recognition engine he implemented. This represents an interesting use of the transduction capabilities of INCA.
- A distributed XML service that handles transparent storage and retrieval of XML documents reporting session-level information of captured data.

7 Conclusions

Previous research demonstrated the value of automated capture and access as a significant class of ubiquitous computing systems. Despite this, we observe that features of capture and access have not been sufficiently explored in many domains.

We researched and identified key architectural insights into the creation of this class of 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 applications. We introduce the INCA toolkit as a software infrastructure for transforming high-level designs into implementations while hiding from the programmer details of certain development tasks incidental to the software. INCA separates various application concerns into individual functional building blocks and decouples features that cut across all aspects of the application from system code. We validated that INCA simplifies the development and evolution of complex capture and access capabilities through use in a number of applications. The successful uses of INCA by us and others demonstrated that we identified the proper software structuring for this class of applications and gives us confidence that we created an important tool for others to build upon.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. 0070345 and 0121661. The authors thank Maria da Graça Pimentel, Tom Barnwell and Lonnie Harvel for their continued collaboration on this work. We also deeply appreciate the endless help and support provided by Anind Dey, Gillian Hayes, Elaine Huang and members of the Georgia Tech's Ubiquitous Computing group.

References

1. Abowd, G.D., Classroom 2000: An Experiment with the Instrumentation of a Living Educational Environment. IBM Systems Journal. **38**(4) (1999) pp.508-530.
2. Bacher, C., Muller, R., Ottmann, T., and Will, M. Authoring on the Fly. A New Way of Integrating Telepresentation and Courseware Production. In the *Proceedings of International Conference on Computers in Education (ICCE'97).* Kuching, Sarawak, Malaysia (1997)
3. Berque, D. Using a Variation of the WYSIWIS Shared Drawing Surface Paradigm to Support Electronic Classrooms. In the *Proceedings of HCI International 1999.* Munich, Germany (1999)
4. Bianchi, M. AutoAuditorium: A Fully Automatic, Multi-Camera System to Televise Auditorium Presentations. In the *Proceedings of DARPA/NIST Smart Spaces Technology Workshop.* (1998)
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. Chiu, P., Kapuskar, A., Reitmeier, S., and Wilcox, L. NoteLook: Taking Notes in Meetings with Digital Video and Ink. In the *Proceedings of ACM Multimedia'99.* Orlando, FL (1999) pp.149-158.
7. Cruz, G. and Hill, R. Capturing and Playing Multimedia Events with STREAMS. In the *Proceedings of ACM Multimedia '94*. San Francisco, CA (1994) pp.193-200.
8. Davis, R.C., Landay, J.A., Chen, V., Huang, J., Lee, R.B., Li, F.C., Lin, J., III, C.B.M., Schleimer, B., Price, M.N., and Schilit, B.N. NotePals: Lightweight Note Sharing by the Group, for the Group. In the *Proceedings of CHI'99.* Pittsburgh, PA (1999) pp.338-345.
9. Dey, A.K., Salber, D. and Abowd, G.D. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Human-Computer Interaction (HCI) Journal. **16**(2-4) (2001) pp.97-166.

10. Dey, A.K., Futakawa, M., Salber, D., Abowd, G.D. The Conference Assistant: Combining Context-Awareness with Wearable Computing, In the *Proceedings of the 3rd International Symposium on Wearable Computers (ISWC '99)*. San Francisco, CA (1999) pp.21-28.
11. Gellersen, H.W., Schmidt, A. and Beigl, M. Multi-Sensor Context-Awareness in Mobile Devices and Smart Artefacts.. Mobile Networks and Applications (MONET). **7**(5) (2002) pp.341-351.
12. Greenberg, S. and Fitchett, C. Phidgets: Easy Development of Physical Interfaces through Physical Widgets. In the *Proceedings of UIST 2001.* Orlando, FL (2001) pp.209-218.
13. He, L., Sanocki, E., Gupta, A., and Grudin, J. Auto-Summarization of Audio-Video Presentations. In the *Proceedings of ACM Multimedia 1999.* Orlando, FL (1999) pp.489-498.
14. Fleck, M., Frid, M., Kindberg, T., O'Brien-Strain, E., Rajani, R. and Spasojevic M. Rememberer: A Tool for Capturing Museum Visits. In the *Proceedings of UBICOMP 2002.* Goteberg, Sweden (2002) pp.48-55.
15. Klemmer, S.R., Li, J., Lin, J. and Landay, J.A. Papier-Mâché: Toolkit Support for Tangible Input. In the *Proceedings of CHI 2004.* Vienna, Austria (2004)
16. Lamming, M., and Flynn, M. "Forget-me-not" Intimate Computing in Support of Human Memory. In the *Proceedings of FRIEND21: Symposium on Next Generation Human Interfaces.* Tokyo, Japan (1994)
17. Liu, Q., Rui, Y., Gupta, A., and Cadiz, J.J. Automating Camera Management for Lecture Room Environments. In *Proceedings of CHI 2001.* Seattle, WA (2001)
18. Macedo, A.A., Truong, K.N., Pimentel, M.G.C., and Camacho, J.A. Automatically Sharing Web Experiences through a Hyperdocument Recommender System. In the *Proceedings of ACM HyperText 2003.* Nottingham, UK (2003)
19. Mankoff, J.C., Hudson, S.E. and Abowd, G.D. Interaction Techniques for Ambiguity Resolution in Recognition-Based Interfaces. In the *Proceedings of UIST 2000.* San Diego, CA (2000) pp.11-20.
20. Mukhopadhyay, S. and Smith, B. Passive Capture and Structuring of Lectures. In the *Proceedings of ACM Multimedia'99.* Orlando, FL (1999) pp.477-487.
21. Omoigui, N., He, L., Gupta, A., Grudin, J., and Sanocki, E. Time-Compression: Systems Concerns, Usage, and Benefits. In the *Proceedings of CHI'99.* Pittsburgh, PA (1999) pp.136-143.
22. Pedersen, E. McCall, K., Moran, T.P. and Halasz F. Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings. In the *Proceedings of INTERCHI'93.* Amsterdam, The Netherlands. (1993) pp.391-398.
23. Richter, H., Abowd, G.D., Geyer, W., Fuchs, L., Daijavad, S. and Poltrock, S. Integrating Meeting Capture within a Collaborative Team Environment. In the *Proceedings of UBICOMP 2001*. Atlanta, GA (2001) pp.123-138.
24. Stern, M., Steinberg, J., Lee, H., Padhye, J., and Kurose, J. MANIC: Multimedia Asynchronous Networked Individualized CourseWare. In the *Proceedings of Educational multimedia and Hypermedia*. (1997).
25. Stifelman, L.J. The Audio Notebook.Ph.D. Thesis, Media Laboratory, MIT (1997)
26. Streitz, N.A., Geibler, J., Haarke, J., and Hol. J. DOLPHIN: Integrated meeting Support across Local and Remote Desktop Enviroments and LiveBoards. In the *Proceedings of CSCW'94*. Chapel Hill, NC. (1994) pp. 345-357.
27. Sumi,Y., Sakamoto,R., Nakao,K., and Mase,K. ComicDiary: Representing Individual Experiences in a Comics Style. In the *Proceedings of UBICOMP 2002*. Goteberg, Sweden (2002) pp.16-32.
28. Tandler, P. Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices. In the *Proceedings of UBICOMP 2001.* Atlanta, GA (2001) pp.96-115.
29. Truong, K.N., Abowd, G.D., and Brotherton, J.A. Personalizing the Capture of Public Experiences. In the *Proceedings of UIST'99.* Asheville, NC (1999) pp.121-130.
30. 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.* Atlanta, GA (2001) pp.209-224.
31. White, D.R., Camacho-Guerrero, J.A., Truong, K.N., Abowd, G.D, Morrier, M.J., Vekaria, P.C. and Gromala, D. Mobile Capture and Access for Assessing Language and Social Development in Children with Autism. In the *Extended Abstracts of UBICOMP 2003.* Seattle, WA (2003) pp.137-140.
32. Whittaker, S., Hyland, P. and Wiley, M., FiloChat: Handwritten Notes Provide Access to Recorded Conversations. In the *Proceedings of CHI'94*. Boston, MA (1994) pp.271-276.
33. Wilcox, L., Schilit, B.N., and Sawhney, N. Dynomite: A Dynamically Organized Ink and Audio Notebook. In the *Proceedings of CHI'97.* Atlanta, GA (1997) pp.186-193.