Semantic Representation of Digital Ink in the Classroom Learning Partner

By

Michel A. Rbeiz

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 26, 2006

Author . . . .
; and Computer Science
May 26, 2006

Certified by . . . .
Kimberle Koile
search Scientist
esis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

BARKER

Semantic Representation of Digital Ink in the Classroom Learning Partner

By Michel A. Rbeiz

Submitted to the
Department of Electrical Engineering and Computer Science

May 26, 2006

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The research presented in this thesis addresses a critical issue in the introduction of new tablet-PC-based educational technology in the classroom: interpretation and semantic representation of digital ink. The teaching paradigm being investigated is that of in-class exercises, which have proven beneficial to student learning. Recent advances in educational technology support such exercises by means of distributed wireless presentation systems. Such systems have proven successful, but are limited in their scope because they rely on digital ink as the communication medium between an instructor and his or her students.

The work done in this thesis extends the use of such systems and makes the following contributions towards the creation of a learning partner in the classroom:

- an ink interpreter capable of text and arrow interpretation which can rival PRS on multiple choice and true-false questions
- a framework for sketch and text interpretation inspired from state of the art research in handwriting and sketch interpretation
- an infrastructure necessary for creating in-class exercises as part of Classroom Presenter, interpreting student answers, and aggregating them

Thesis Supervisor: Kimberle Koile, PhD
Title: Research Scientist

Acknowledgements

Like any large project, this work could not have been done alone, so I would like to thank the people who made it possible and who supported me throughout this last year.

I would like to first thank my thesis advisor, Kimberle Koile, for her mentoring, her guidance, and her support for my work. I owe to her a developed understanding of AI and a passion for "cool" projects.

I want to thank the members of the CLP group in general for their hard work, camaraderie, without which this work would not be as useful. I would like to especially thank Kevin Chevalier, Adam Rogal, and Kah-Seng Tay for staying late nights helping and contributing ideas to this thesis. I also want to thank more broadly members of the AIRE group for their companionship over the last year and members of the DRG group at MIT for their conversations about sketch recognition and their help, especially Tracy Hammond, Aaron Adler, and Michael Oltmans.

Being a collaborative work with Microsoft and the University of Washington, I would like to thank the ConferenceXP research team and the UW Classroom Presenter Team for their help, especially Todd Landstat at Microsoft who provided me with incredibly detailed emails to help me out.

I would like to thank all professors at MIT who have made my experience memorable, most especially, Professor Boleslaw Wyslouch for countless conversations about modeling the world with physics, Professor James Paradis for great conversations about history and civilization, Professor Patrick Winston for deepening my interest in AI and cognition, and teaching me how to write better, and finally Professor Harold Abelson for his guidance as my academic advisor.

I would also like to thank my friends for their support through this last year, most especially Bassam, Ziad, Jad, Bernardo, and my girlfriend Sharon who contributed to my thesis.

Finally, I would like to thank my family, especially my father Aziz, my mother Joyce, my brother Nicolas, my aunts Myriam and Nayla, my grandmothers Marcelle and Fifi, my uncle George for their unconditional support and love.

This work was supported by the MIT-Microsoft iCampus alliance (http://icampus.mit.edu)

Table of Contents

List of Figures

List of Tables

1 Introduction

The research presented in this thesis addresses a critical issue in the introduction of new tablet-PC-based educational technology in the classroom: interpretation and semantic representation of digital ink. The teaching paradigm being investigated is that of in-class exercises, which have proven beneficial to student learning [Bransford, et. al 2000; Hake 1998]. Recent advances in educational technology support such exercises by means of distributed wireless presentation systems. Such systems have proven successful, but are limited in their scope because they rely on digital ink as the communication medium between an instructor and his or her students. Extending the scope of such systems involves overcoming the following challenges:

- creating in-class exercises
- interpreting student answers
- aggregating student answers
- building communication infrastructure

This thesis addresses two challenges in extending the use of such systems to a wide variety of classroom settings. It makes the following contributions towards the creation of a learning partner in the classroom:

- an ink interpreter capable of text and arrow interpretation that can rival PRS on multiple choice and true-false questions
- a framework for sketch and text interpretation inspired from state of the art research in handwriting and sketch interpretation
- an infrastructure necessary for creating in-class exercises as part of Classroom Presenter, interpreting student answers, and aggregating them

The thesis first describes in Chapter 2 how the research fits into falls into the larger picture of creating a learning partner for the classroom. In Chapter 3, it describes the data model supporting the use of interpreted ink in the new tablet-PC-based system called the Classroom Learning Partner (CLP) [Koile and Shrobe 2005]. In Chapter 4, it describes a technique for interpreting digital ink, i.e., translating ink into a semantic representation, as well as the implementation of a text and arrow interpreter. This thesis demonstrates that the accuracy of interpretation of such a system rivals PRS on multiple choice and true-false questions. A system such as CLP, however, is not limited to multiple choice and true-false in-class questions but enables open-ended questions. Finally, the thesis presents in chapter 5 salient ideas for adding functionality to CLP and improving accuracy of the interpreter.

2 Classroom Learning Partner

2.1 Scenario

The goal of the Classroom Learning Partner (CLP) is to improve classroom interaction for both students and instructors, especially in large classes. The following scenario, from [Koile and Shrobe, 2005], illustrates its envisioned use in a classroom in which an instructor lectures via slides projected from his tablet-PC, and the students view the slides on their tablet-PCs.

> An instructor is lecturing in an introductory computer science class on environment diagrams. She puts up a slide that discusses the difference between defining a new variable and setting the value of an existing variable. She goes through an example: Using digital ink on a slide, she "(define z 20)" and draws a line through "z: 10", replacing it with "z: 20". She continues with the example, then says, "It's your turn now." She displays a slide on which she asks the students to evaluate four particular expressions in order and modify the accompanying environment diagram accordingly. The students work the problems using ink on their tablet-PCs, and then wirelessly send their answers to the instructor's machine by clicking on a button labeled "submit". The Classroom Learning Partner interprets the digital ink answers, tagging them with semantic information, then aggregates the answers, and shows the instructor a histogram of the responses, along with representative answers for each category in the histogram. The instructor then discusses the problems, displaying the representative student submissions for each.

This thesis focuses on two important components of the Classroom Learning Partner: an interpreter capable of tagging digital ink with semantic information and a data model that supports the use of such an interpreter in aggregating student answers. To understand the importance of semantic information, consider Noam Chomsky's famous quote: "colorless green ideas sleep furiously." This sentence is syntactically correct but semantically nonsensical. Similarly, digital ink is syntactically a series of points interconnected together. Without an interpreter to "understand" the ink –i.e., tag the bits with semantic representation– students' answers cannot be aggregated as described in the above scenario. Without aggregation, an instructor cannot assess how well her students understand a topic; she instead will be overwhelmed by possibly hundreds of student answers.

2.2 Background

Previous research in educational psychology has shown that students learn better when they are actively engaged in the learning situation [Bransford, et. al. 2000, Bresler 2004]. It has also shown that students learn better when engagement takes the form of hands-on activities that yield immediate feedback through interaction with peers and/or instructors [Hake 1998]. CLP takes inspiration from two different teaching methodologies, each of which engages students actively in the learning situation by means of in-class exercises— the use of Personal Response Systems (PRS) in large classrooms, and the use of in-class exercises in small classrooms.

Personal Response Systems enable students to answer questions using infrared wireless transmitters with a click of a button [Duncan 2005]. Results are instantly charted and displayed for real-time student feedback and lesson refinement, allowing an instructor to make the most of her class time and students to make the most of their learning. PRS, however, has some limitations: it only supports the use of multiple choice, true or false, or matching questions.

Using a wider variety of in-class problems, particularly what are called open-ended questions, has proven beneficial to student learning [Hake 1998]. This style of teaching is well-supported by a tablet-PC-based system called Classroom Presenter [Anderson, et. al, 2004]. Classroom Presenter is a tablet-PC-based distributed presentation system that enables instructors to annotate slides with diagrammatic ink [Anderson et. al, 2005]. The system allows the instructor to broadcast ink almost instantaneously to student tablets. Furthermore, an instructor can display a slide containing an exercise for students to work.

The students work the problem, using digital ink on their tablet-PCs, and then anonymously submit their solutions to the instructor via either a wired or wireless network. The instructor can examine the students' submissions individually. Using Classroom Presenter in this way works well in classes of eight or smaller [Anderson 2005], as instructors are overwhelmed easily by more than eight solutions.

CLP combines the benefits of PRS and Classroom Presenter by enabling the use of non-multiple-choice exercises in large classes. It does so using the technique proposed in this thesis: It interprets the student ink answers, translating them into a semantic representation, and then it aggregates the answers. A recent pilot study with Classroom Presenter has shown that this engagement style of teaching resulted in more students scoring in the top 10% of the class than expected, and fewer scoring in the bottom 10% in the introductory computer science class. [Koile and Singer, 2006a] [Koile and Singer, 2006b]

2.3 Classroom Learning Partner Architecture

We propose to divide the CLP into three main components: [Koile and Shrobe, 2005]

- *Instructor authoring tool*: which enables an instructor to create a presentation with in-class exercises and answers [Chen 2006]
- *Interpreter*: a semantic ink interpreter for the tablet-PC that extracts semantic information from student answers to in-class exercises
- *Aggregator:* which aggregates student answers into equivalence classes and display a summary [Smith 2006]

The following diagram illustrates the system architecture of the CLP and the interactions between the different modules.

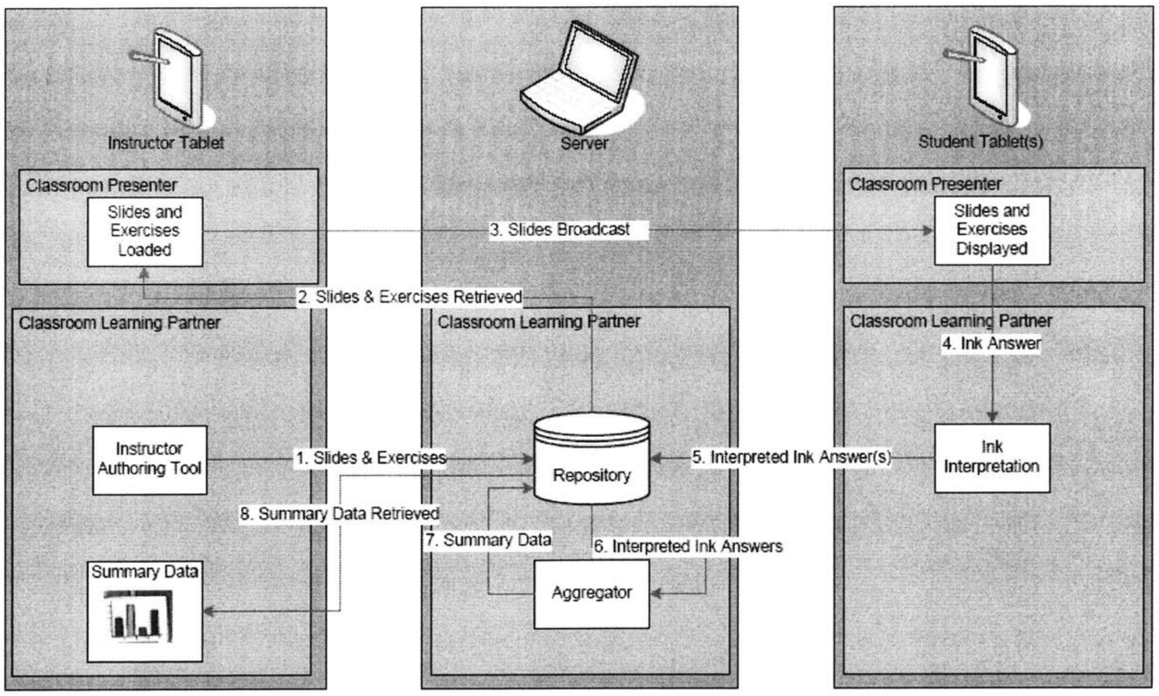

Instructor Tablet
Server
Student Tablet(s)
Classroom Presenter
Slides and Exercises Loaded
3. Slides Broadcast
Classroom Presenter
Slides and Exercises Displayed
2. Slides & Exercises Retrieved
Classroom Learning Partner
Classroom Learning Partner
Classroom Learning Partner
4. Ink Answer
Instructor Authoring Tool
1. Slides & Exercises
Repository
5. Interpreted Ink Answer(s)
Ink Interpretation
8. Summary Data Retrieved
7. Summary Data
6. Interpreted Ink Answers
Summary Data
Aggregator

Figure 2.1 CLP System Architecture

Before class:

1. The instructor creates a PowerPoint presentation along with a set of exercise objects. The exercise information is both embedded in the slides and stored as a separate object in the repository. The presentation slides and exercises are stored in the database as a single lecture object.

During class:

2. The instructor retrieves the PowerPoint presentation from the database.
3. The presentation slides are broadcast to student machines or retrieved from the central repository.
4. When a slide containing an exercise is displayed, each student enters a digital ink answer, which is interpreted on her machine.
5. Each student's interpreted answer is submitted and stored in the database.
6. When the instructor indicates end of student exercise period, the aggregator retrieves the interpreted ink answers, aggregates them, and produces summary data.
7. The summary data is stored in the database.
8. The summary data is displayed on the instructor's machine.

This thesis focuses primarily on semantic representation of digital ink answers and on establishing an asynchronous data model for the system using the repository as the core of the system.

We are building the CLP on top of Classroom Presenter, which in turn is built on top of the ConferenceXP platform. The following diagram gives the high-level software architecture of our project.

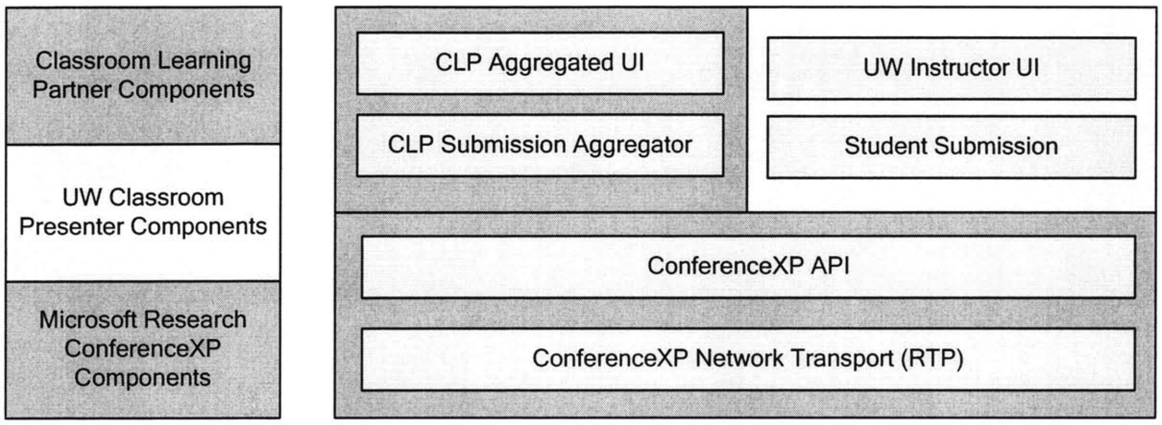

Classroom Learning Partner Components
UW Classroom Presenter Components
Microsoft Research ConferenceXP Components
CLP Aggregated UI
CLP Submission Aggregator
UW Instructor UI
Student Submission
ConferenceXP API
ConferenceXP Network Transport (RTP)

Figure 2.2 CLP Software Architecture [Koile and Shrobe, 2005]

3 Data Model

This chapter focuses on the framework[1] put in place to enable communication between CLP's different software modules. As outlined in the architectural diagram (Figure 2.1), we have added a central repository and three main modules to Classroom Presenter: *the instructor authoring tool, the interpreter, and the aggregator.* The data model provides a shared language that supports communication between the modules. The central repository, which implements the data model, provides a shared memory that also supports the communication. The data model and repository are necessary because Classroom Presenter state saving capability does not satisfy our needs. Student submissions in Classroom Presenter, for example, are transmitted and stored on the instructor machine in an internal representation. It would be computationally inefficient for the CLP modules to retrieve information from the Classroom Presenter files; Classroom Presenter does not have simple mechanisms to enable access to its internal representations. Furthermore, for the aggregator to rely on submission data on the instructor machine would create a networking bottleneck with hundreds of submissions being sent to the instructor machine[2]. The separation of the CLP modules from Classroom Presenter also is essential for enabling independent upgrades of the different modules, given that the research group at University of Washington maintains Classroom Presenter.

The CLP modules and central repository run on different machines; the interpreter runs on student machines. The alternative would have been to collect digital ink and run interpretation in batch on a server. The parallel processing of ink on different student

[1] The author designed this system; much of the implementation has been done by Kah-Seng Tay, an undergraduate researcher in our group.

[2] During our testing in classes of size 18 or fewer, student submissions were sent to both the instructor machine and to the central repository. This dual storage was primarily used for debugging purposes.

machines, however, turns out to be computationally more efficient. The central repository runs on its own server which is advantageous when scaling to a class of three hundred; students submitting 300 answers almost simultaneously would likely consume most of a machine's resources. Profiling the aggregator [Smith 2006] has shown that we can put the aggregator on the server or the instructor machine without a significant hit in performance. Finally, the instructor authoring tool [Chen 2006] can run on any machine and is used by the instructor outside of class to create presentations used with CLP.
The remainder of this chapter focuses on detailing more issues on the abstractions in our data model and on the implementation details of a database that serves as our central repository.

3.1 Abstraction

Our data model abstraction was designed to store the content of a typical class, and to enable communication between the different software modules. It was modeled after the organization of a class: by Course, Lecture, and Exercise. Figure 3.1 outlines the details of our model. A course was divided into various lectures, and has multiple students enrolled. An instance of student is created at every lecture, with a timestamp, thus enabling an instructor to track attendance if desired. Each lecture consists of a slide presentation and a set of exercises to be worked by the students[1]. At the time of writing, an exercise answer can be any of the types listed below. Only Scheme Expression is specific to the introductory computer science class [Abelson et al, 1996]. The possible answer types and examples of each are:

- Number: 123
- String: error

[1] The exercise content is stored in both the slide presentation – for use by the ink interpreter that runs on student machines- and in the central repository – for use by the aggregator.

- Set: (1 2) : order does not matter
- Sequence: number, number, number → number
- True-False: #t, #f
- Multiple Choice: A or 1
- Scheme Expression: (lambda (a b) (+ a b))

Our system is capable of interpreting handwritten answers, but it cannot yet process handwritten sketches. Each Exercise object contains a question object and may contain instructor-specified answers. The answer can denote the right answer(s) for the exercise, or incorrect ones illustrating errors that correspond to misunderstood concepts. Instructor answers need not be specified; the aggregator can cluster student answers based on similarity measures rather than matching to an instructor answer. Students' submitted answers are instantiated as student answer class objects which, like instructor answer, inherit from the Answer class. The Answer object stores ink, the semantic representation of the stored ink, and the id of the associated exercise. We store the ink object in order to either check at a later point the accuracy of the recognition, and/or to collect samples for training our recognizer. Multiple student answers then are aggregated and put into answer bin objects. Note that the system allows for multiple student answers per student. The aggregator always uses the most recent answer (ordered by time stamp).

Since the model is accessed by various modules, sometimes simultaneously, we made all classes immutable. In this way, programs are always accessing copies of objects and replacing objects by dereferencing old copies.

In summary, our data model enables us to:

- track students for each class
- store lectures
- store exercises
- store student answers
- store aggregation results

This information has proven sufficient in our studies thus far [Koile and Singer, 2006b]. In addition, an instructor can perform data mining on the database for educational purposes such as measuring the impact of her teaching, or measuring the impact of educational technology, or even looking up what was done in previous years.

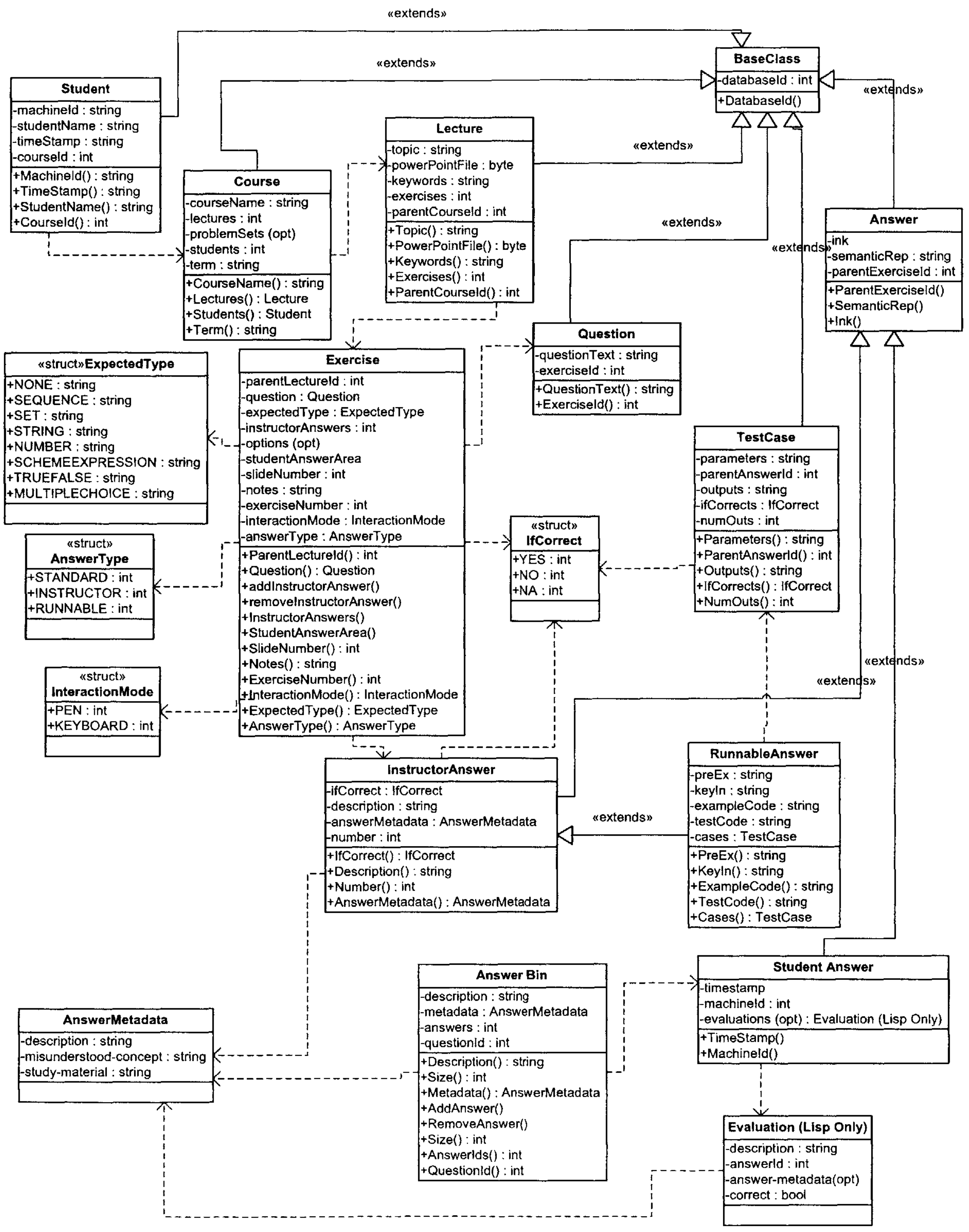


Figure 3.1 UML Diagram of Abstraction Model

3.2 Database

Database Architecture

This section focuses on the mechanics used to implement the abstraction model in a relational database. Our database export module consists of two subcomponents:

- *Database Driver*: a library of simple database methods such as connecting to the database, executing queries or non-queries, rolling back failed transactions[1]. This library encapsulates details of the database connections, specific SQL queries, time-outs and retrying from any client module, and can be reused in other applications.
- *CLPDatabase*: a library that saves and loads classes from the data model to the database. This module implements our data model and enables loading and saving student answers and class information in the database. Multiple relational tables may be used to implement a single object type. When saving a student answer to the database, for example, both the `StudentAnswers` and `Answers` tables need to be modified. Similarly, when loading a student answer, information comes from these two tables as well.

This implementation architecture enables us to switch to another database or to an XML file without having to change the data model. Clients of the database module thus need not know anything about how the database is implemented. Figure 3.2 outlines the UML diagram of the two components, the Database Driver and the CLPDatabase.

[1] A query is a "SELECT" statement and other statements querying data. A non-query is an "UPDATE" or "CREATE TABLE" command and other commands changing the nature of the data. A transaction is a group of queries and non-queries executed in batch.

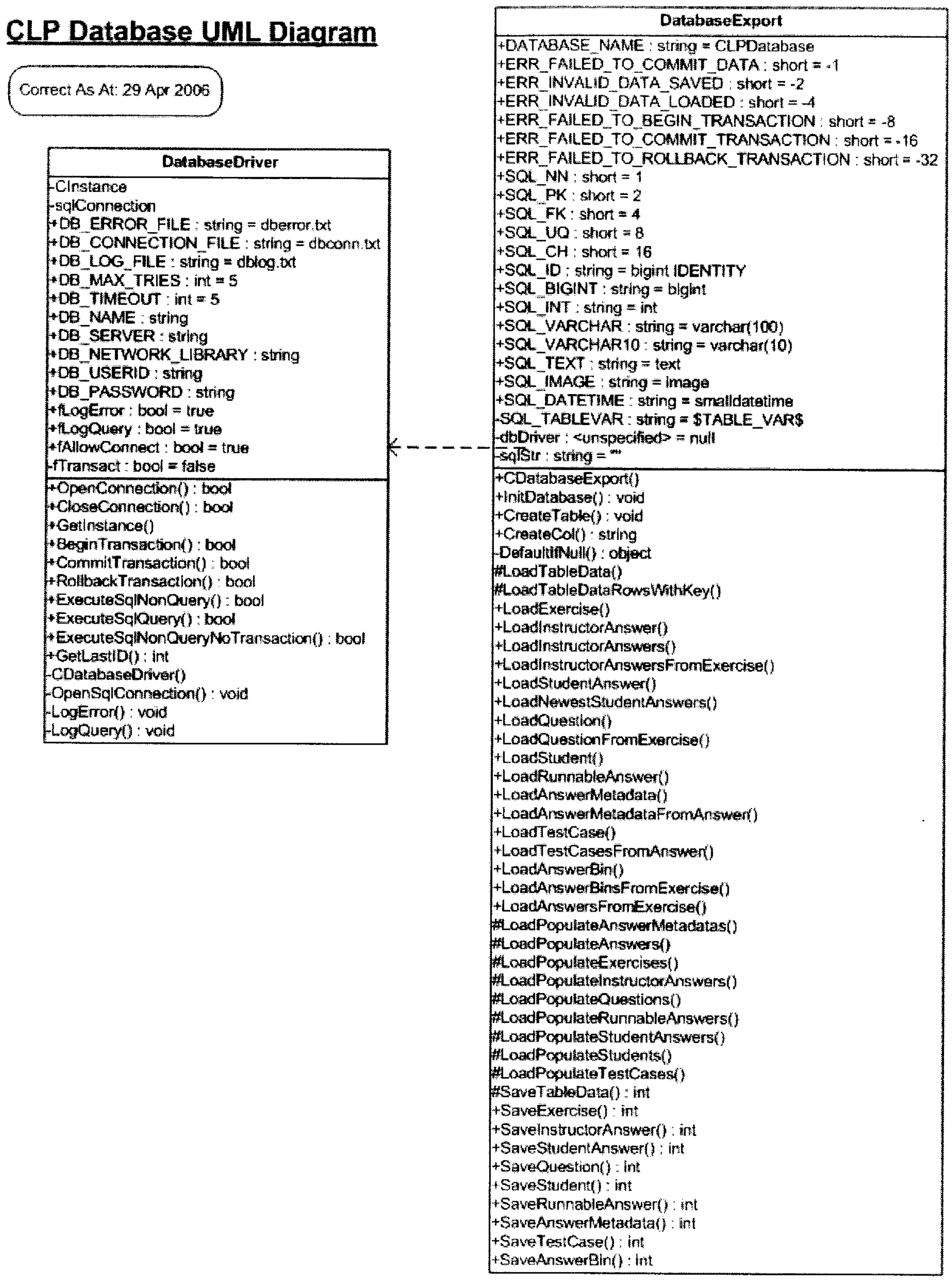

CLP Database UML Diagram
Correct As At: 29 Apr 2006
DatabaseDriver
-CInstance
-sqlConnection
+DB_ERROR_FILE : string = dberror.txt
+DB_CONNECTION_FILE : string = dbconn.txt
+DB_LOG_FILE : string = dblog.txt
+DB_MAX_TRIES : int = 5
+DB_TIMEOUT : int = 5
+DB_NAME : string
+DB_SERVER : string
+DB_NETWORK_LIBRARY : string
+DB_USERID : string
+DB_PASSWORD : string
+fLogError : bool = true
+fLogQuery : bool = true
+fAllowConnect : bool = true
-fTransact : bool = false
+OpenConnection() : bool
+CloseConnection() : bool
+GetInstance()
+BeginTransaction() : bool
+CommitTransaction() : bool
+RollbackTransaction() : bool
+ExecuteSqlNonQuery() : bool
+ExecuteSqlQuery() : bool
+ExecuteSqlNonQueryNoTransaction() : bool
+GetLastID() : int
-CDatabaseDriver()
-OpenSqlConnection() : void
-LogError() : void
-LogQuery() : void
DatabaseExport
+DATABASE_NAME : string = CLPDatabase
+ERR_FAILED_TO_COMMIT_DATA : short = -1
+ERR_INVALID_DATA_SAVED : short = -2
+ERR_INVALID_DATA_LOADED : short = -4
+ERR_FAILED_TO_BEGIN_TRANSACTION : short = -8
+ERR_FAILED_TO_COMMIT_TRANSACTION : short = -16
+ERR_FAILED_TO_ROLLBACK_TRANSACTION : short = -32
+SQL_NN : short = 1
+SQL_PK : short = 2
+SQL_FK : short = 4
+SQL_UQ : short = 8
+SQL_CH : short = 16
+SQL_ID : string = bigint IDENTITY
+SQL_BIGINT : string = bigint
+SQL_INT : string = int
+SQL_VARCHAR : string = varchar(100)
+SQL_VARCHAR10 : string = varchar(10)
+SQL_TEXT : string = text
+SQL_IMAGE : string = image
+SQL_DATETIME : string = smalldatetime
-SQL_TABLEVAR : string = $TABLE_VAR$
-dbDriver : <unspecified> = null
-sqlStr : string = ""
+CDatabaseExport()
+InitDatabase() : void
+CreateTable() : void
+CreateCol() : string
-DefaultIfNull() : object
#LoadTableData()
#LoadTableDataRowsWithKey()
+LoadExercise()
+LoadInstructorAnswer()
+LoadInstructorAnswers()
+LoadInstructorAnswersFromExercise()
+LoadStudentAnswer()
+LoadNewestStudentAnswers()
+LoadQuestion()
+LoadQuestionFromExercise()
+LoadStudent()
+LoadRunnableAnswer()
+LoadAnswerMetadata()
+LoadAnswerMetadataFromAnswer()
+LoadTestCase()
+LoadTestCasesFromAnswer()
+LoadAnswerBin()
+LoadAnswerBinsFromExercise()
+LoadAnswersFromExercise()
#LoadPopulateAnswerMetadatas()
#LoadPopulateAnswers()
#LoadPopulateExercises()
#LoadPopulateInstructorAnswers()
#LoadPopulateQuestions()
#LoadPopulateRunnableAnswers()
#LoadPopulateStudentAnswers()
#LoadPopulateStudents()
#LoadPopulateTestCases()
#SaveTableData() : int
+SaveExercise() : int
+SaveInstructorAnswer() : int
+SaveStudentAnswer() : int
+SaveQuestion() : int
+SaveStudent() : int
+SaveRunnableAnswer() : int
+SaveAnswerMetadata() : int
+SaveTestCase() : int
+SaveAnswerBin() : int

Figure 3.2 UML Diagram of Database Module

Choice of Platform

An object-oriented database would have suited our needs best. We found, however, that a relational database provided by Microsoft integrated more easily with the rest of our system[1]. We, therefore, emulated some features found in an object-oriented database. A course, for example, has a list of students. Lists, however, are not easily implemented in a relational database. So instead, each object has its own table in the system. For each list element, the system creates an entry in the corresponding table for that class. Each list element references the object to which it belongs. Each student entry, for example, references the course with which it is associated.

Object Identification

Each object has a database ID that is unique to that object. Since objects are created by different modules and stored in the same repository, it is only logical that the database assigns a unique identifier to each object. The identifier need only be unique within that a particular table, however. An exercise and a question, for example, can both have the same id number because the combination of object type and id number is unique. Producing an id unique across the entire system is more computationally intensive than a locally unique id and is not necessary.

Inheritance

We emulated inheritance between two objects by assigning the same id across the two tables associated with the respective objects. For example, both `StudentAnswer` and `InstructorAnswer` inherit from Answer. They share the same id across both tables.

---

[1] Our system is implemented as a .NET application. (see http://msdn.microsoft.com/netframework/ for information on .NET).

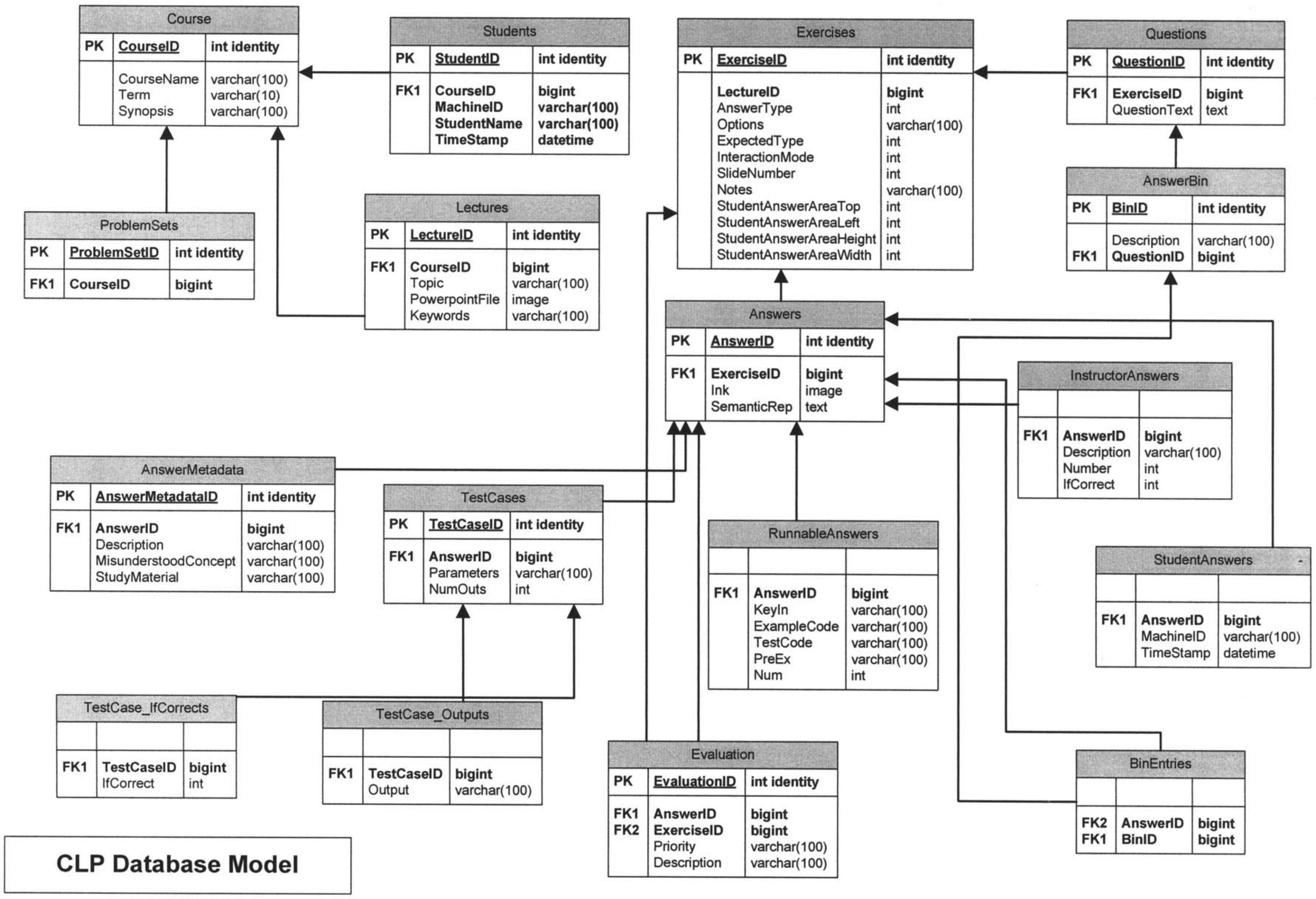


Figure 3.3 CLP Database Model

Communication

Communication with the CLP database happens over a more reliable transmission protocol than Classroom Presenter. Classroom Presenter is built on top of the ConferenceXP research platform [Pahud 2006], whose goal is to enable researchers to develop collaborative applications for learning and research. ConferenceXP supports the development of real-time collaboration and videoconferencing applications by delivering high-quality, low-latency audio and video over broadband connections, as well as by providing a flexible common framework for designing and implementing collaborative applications. ConferenceXP uses RTP as a transport layer. The *Real-time Transport Protocol (or RTP)* defines a standardized packet format for delivering audio and video over the Internet. It is a multicast protocol, and is frequently used in streaming media systems as well as videoconferencing. The shortcomings of RTP that most concern us are the following:

- it does not ensure timely delivery.
- it does not give any quality of service guarantees.
- it does not guarantee in-order delivery [Wikipedia 2006a].

RTP is a suitable choice for Classroom Presenter because transmitting slide from one instructor machine to many student machines needs to be fast and easy. In addition, any unsent slide can be resent easily.

When students submit their answers to exercises, however, they are sending to a single, unique location, i.e., the database. A unicast protocol ensuring packet delivery is better suited to this situation.

The *Transmission Control Protocol (TCP)*, one of the core unicast protocols of the Internet protocol suite, has proven reliable. Using TCP, applications on networked hosts can create connections to one another, over which they can exchange packets of data. The protocol guarantees reliable and in-order delivery of sender to receiver data. TCP also distinguishes data for multiple, concurrent applications (e.g., Web server and email server) running on the same host. [Wikipedia 2006b] TCP is a suitable choice for communicating with the CLP database.

In addition to the protocol, the transmission medium affects communication with the database: Classroom presenter runs over wireless network which is sometimes lossy or can be interrupted. Our module thus took into account the possibility of lossy data by:

- Making each database operation atomic. Each atomic operation executes entirely or not at all. In case a networking issue arises during an operation, the changes to the database can be undone. This roll-back feature was implemented by surrounding operations with transactional features of the relational database package. A transaction in SQL is the database implementation of an atomic operation. The keywords in MS SQL Server are "BEGIN TRANSACTION" and "COMMIT TRANSACTION." Unless operations run to completion, the changes to the database are not permanent and can be rolled back. SQL Server provides the "ROLLBACK TRANSACTION" method, which restores the database to the state before the transaction ran.
- Running the database API in a separate thread. This separation allows for a timeout function and an interrupt function in case the database times out or a sudden bottleneck in communication occurs.
- Using persistent data. The `DatabaseExport` class performs Save instead of Update methods in order to maintain a persistent state of all data previously saved into the database. We can save new objects rather than update old ones because we are not limited by the number of unique ids or space needed. This scheme guarantees that we have a consistent data set at any time, and also allows us to keep a history of all previously stored objects in the database.

4 Ink Interpretation

4.1 Purpose

The purpose of the Ink Interpretation module is to extract semantic information from digital ink, providing the aggregator with enough semantic information to group student answers into equivalence classes. From an implementation point of view, the ink interpretation module tags digital ink bits with metadata about the semantic information of the bits.

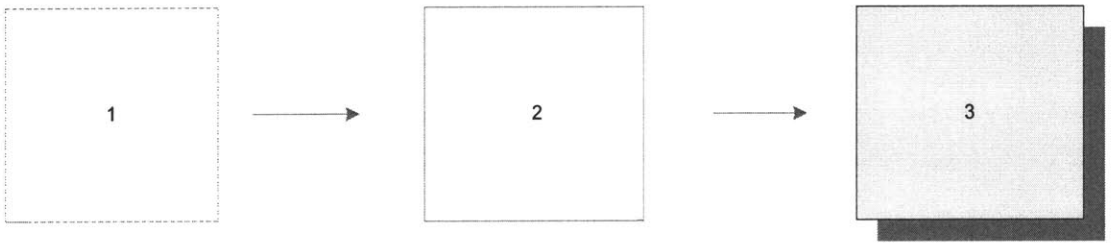


Figure 4.1 Ink Interpretation Example of an Inked Box

An example of ink interpretation is shown in Figure 4.1. A user draws a box using digital ink on a tablet-PC. The tablet-PC libraries capture a sequence of point coordinates (1). A sketch recognizer such as SketchRead [Alvarado and Davis, 2004], recognizes the sequence of dots as four intersecting lines (2). With knowledge of a domain, for example a box diagram domain, the interpreter deduces that the user drew a box (3). If we chose a mathematical domain, the interpreter would deduce that the user drew a square.

This thesis focuses primarily on the design and implementation of an appropriate framework for text and diagram interpretation. The implemented framework supports interpretation of strings, numbers, and symbols (namely arrows). In a second phase, outside the scope of this thesis, the interpreter will be expanded to include support for diagram interpretation. There currently exist state of the art recognizers that provide accurate recognition for text (Microsoft Tablet-PC recognizer), or diagrams [Alvarado and Davis, 2004]. Researchers recently have been integrating diagram and text recognition, such as in the recognition of chemical structures, but no off-the-shelf recognizer provides functionality for recognizing both in a single sequence of ink. The goal of this thesis is to build on current state of the art recognizers to support the combination of diagram and text recognition.

In this thesis, I use the words recognition and interpretation, which are close in meaning but are subtly different. Recognition denotes the process of identification and classification of a certain object. I use the word interpretation as a superset of recognition to mean that there is context supporting the recognition. For example, recognition is a matter of shape matching, whereas interpretation is the process of passing a shape through a recognizer, and making a choice given context.

4.2 Background in Handwriting Recognition

The difficulties encountered in recognizing handwriting come mainly from variations in writing styles. These difficulties translate into a non-perfect degree of recognition accuracy. While a recognizer with 99% accuracy is considered a feat in academia, a commercial application with this accuracy would yield bad results when handling a large volume of inputs (in millions). The United States Postal Service, for example, uses a handwriting recognizer to recognize addresses printed on envelopes. We can assume that the software recognizes all machine written addresses and has a 99% accuracy rate on handwritten addresses. The USPS website [USPS 2005] claims that USPS delivered over 206 billion pieces of mail in 2005. There are multiple categories of mail but if we assume that handwritten addresses appear on 1% of the total volume, then the recognizer would miss the recognition of 1% x 206 billion x (100%-99%)= 20.6 million envelopes. This inaccuracy is potentially one of the reasons why people sometimes do not get their mail. In reality, USPS has workarounds because it benefits from constraints on the possible number of possibilities for zip codes, states etc... USPS also has a person overseeing the results of the recognizer, and this person can make corrections when the recognizer signals low confidence in results. Srihari details the system in place at USPS in [Srihari and Keubert, 1997].

In this thesis, achieving high recognition rates is important because students would like the machine to "get their answer right" or they will not trust the system; they may become distracted if concerned about recognition instead of working problems in class.

There is a distinction that has traditionally been made between on-line and off-line recognition. In on-line systems, the temporal information about the writing speed, acceleration and order of line segments making up a word is available to the recognizer. Tablet-PCs are examples of on-line systems. Off-line systems only use the handwriting information stored in an image. The USPS handwritten address recognizer is an example of an off-line system. There are methods to attempt the extraction of this temporal information from the handwriting in off-line systems, but they are generally not used in commercial systems. This extra information makes for higher recognition rates, leading on-line systems to perform better. Systems using tablet-PCs can be considered on-line because they can capture spatial temporal features, which can be used in recognition.

To some degree, recognition systems attempt to replicate human's abilities to recognize handwriting. In this light, it was observed that humans recognize handwritten words based on the knowledge of possible character combinations. As a result, humans are able to recognize words with blurred or missing characters. [Jacobs et. al 1994] Systems performing recognition try to mimic humans. Thus, in recognizing natural language, a holistic (where the entire word is recognized at a time) approach may be more effective than a character based approach (where each character is recognized individually as part of the word). It is important to note that when a person does not recognize a word, she deciphers the different letters and looks words up in a dictionary. In the case of USPS, a character recognition system might work better than a word recognition system since the words used are not always legal (i.e., part of a dictionary).

A typical architecture for a recognizer:

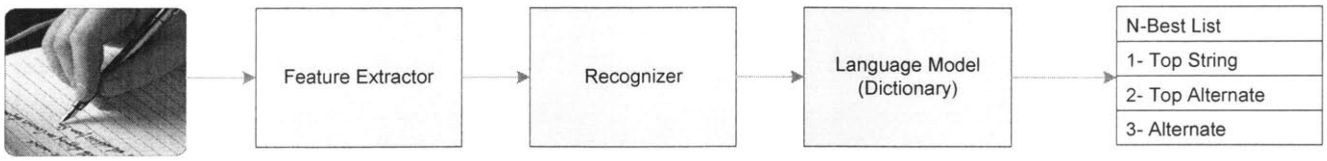


Figure 4.2 Handwriting Recognizer Architecture

As shown in Figure 4.2, each recognizer has several components. The feature extractor attempts to reduce noise found in the input and extracts features relevant to the recognizer. Each recognizer is different in implementation in the sense that the internals of a recognizer, as well as the features emphasized, vary. Examples of these features include slant ("slant" is defined as the average slope of a word with respect to the vertical direction), outer contours (extracted using Gabor Filters) etc. Using these features, the recognizer extracts a list of N-Best possibilities. The recognition process is divided into two sub-processes: segmentation and recognition. Most recognizers operate in a divide and conquer approach: they divide the strokes or the image into pieces and then perform recognition on these pieces. Typical segmentation approaches are segmentation in words or characters. Character segmentation is a challenging task because it is difficult to determine the beginning and ending of a character. Following segmentation, recognition is then performed on the strokes using several AI and pattern recognition algorithms. These pattern recognition algorithms take advantage of statistical information stored in training data in conjunction with shape matching algorithms used in computer vision. The statistical information helps algorithms deal with noise and variations in handwriting, whereas the structural approach with shape matching algorithm helps with machine-printed handwriting. Some of the AI techniques used are:

- Support Vector Machines
- Hidden Markov Models (HMM) [Hu et al, 1996] [Yasuda et al, 2000]
- Neural Networks
- Genetic Algorithms
- Convolutional Time Delay Neural Networks (TDNN)

The latest handwriting recognition methods are very similar on a high level to speech recognition [Makhoul et al 1994] and seem to be focused on modified Hidden Markov Models (HMM) customized to handwriting.
Using HMM, there are generally two basic approaches to perform recognition: word-based recognition and character-based recognition. In word-based recognition, each word has an HMM, which makes the recognizer fairly hard to train because each HMM needs multiple training examples. This approach is limited if the lexicon is big because it has problems recognizing out-of-vocabulary words, i.e., words not present in the training data. Systems with word-level models are therefore appropriate for tasks with small lexicons. For unconstrained handwriting, recognizers typically perform character recognition where each character has an HMM. The HMM, however, can produce an illegal set of characters. The features used in each HMM, i.e., what defines state and transition probabilities, vary by implementation. The HMM produces an N-Best List which is then compared to a list of possibilities and checked against a dictionary or language model. From 26 letters, an HMM potentially can generate all words in a dictionary. Word recognition is performed by matching a possible sequence of character models with the help of a given dictionary.

The recognizer feeds a list of possibilities to the language model module. The complexity of this module depends on implementation: the language model could be as simple as a lexicon of words and a string matching algorithm between candidate words to rank the lexicon. [Plamondon and Srihari, 2000] Another possibility is a dictionary improved by incorporating statistical information for each word. Context-free grammars, and N-Gram class models can also be used as language models, as they are in speech recognition.

One area of research relevant to this thesis is how much weight to give to the language model module and to the recognizer module. If a user writes "confdence", for example, the recognizer might accurately output "confdence". The language model, however, could reject the possibility since "confdence" is not legal and take the alternate which is "confidence". While the language model could potentially recognize the word as it was written, it may use its dictionary to correct the result.

Finally, due to how handwriting recognizers are implemented, they have difficulty recognizing very slanted handwriting. Writing legibly at a 90 degree angle, for example, would throw off most recognizers.

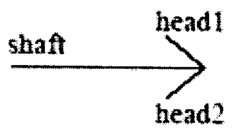
For a more extensive background on handwriting recognition, see [Liu and Cai 2003] and [Plamondon and Srihari, 2000].

4.3 Background in Sketch Recognition

Sketch and Diagram recognition is inherently different than handwriting recognition due to the nature of the task. Single stroke recognition, for example, where recognizers perform incremental character by character recognition such as recognition in handheld computers, is not applicable in sketch recognition. Users tend to draw sketches in multiple steps. It would be terribly awkward to draw a pulley in a single stroke.

Similarly to handwriting recognition, an important sub-problem in recognition is the segmentation of strokes: where are the beginning and the end of a symbol? Sketch recognition also has to deal with the difficulty of context because unlike language, there exists no language models for sketching. A representation of the knowledge contained in a sketch therefore needs to be put in place.

The MIT Design Rationale Group (DRG) has extensively studied sketch recognition. The DRG group took advantage of the fact that complex sketches are composed of simpler sketches and defined a hierarchical sketch recognition language called LADDER. [Hammond and Davis, 2005] They defined primitive objects such as lines and curves usually drawn in a single stroke, making strokes easier to segment. Compound objects are made of primitive objects and the properties of these compound objects vary from domain to domain. A rectangle, for example, could be interpreted as a box in a diagram domain and as a block in a construction domain.



```
(define shape Arrow
(components
        (Line shaft)
        (Line head1)
        (Line head2))
(constraints
        (coincident head1.p1 shaft.p1)
        (coincident head2.p1 shaft.p1)
        (acuteDir head1 shaft)
        (acuteDir shaft head2)
        (equalLength head1 head2)
        (perpendicular head1 head2)))
```

Figure 4.3 Definition of an arrow in LADDER

Figure 4.3 shows the definition of an arrow as a list of object definitions with some constraints in the LADDER recognition language. Recognition is done in incremental, unobtrusive fashion. At no point is the user asked to redraw something because the recognition engine misinterpreted the sketches. If the system tries to interpret the user's strokes before it has enough information, it thus is more likely to err in interpreting pieces of the user's drawing. The system is aware of when it has enough information to interpret a piece of the drawing. Furthermore, the system incrementally adds data to its internal representation making it more complete. Symbol recognition in SketchREAD is done in a 4 stage process, using a blackboard architecture, as illustrated in Figure 4.4:

- **Bottom-Up Step:** Primitive objects and simple compound objects are recognized. [Sezgin et al 2001]
- **Top-Down Step:** Domain-dependent objects are recognized, as in identifying 4 intersecting lines such as in Figure 4.1 as a box in a box and diagram domain. If a compound object has been hypothesized, the search for missing strokes from that compound takes place.
- **Ranking Process:** Many different interpretations may have been proposed for the same strokes. Based on previously interpreted parts of the sketch, the system identifies temporal and spatial context for the newly recognized patterns and uses this context to assign likelihoods to the templates that were generated in steps 1 and 2. [Alvarado et al, 2002] A Bayesian network is dynamically constructed to represent each hypothesis. The hypotheses are then compared using the likelihood of the Bayesian network. The Bayesian network (illustrated in Figure 4.5)

integrates the influence of stroke data and domain specific context in recognition, enabling the recognition engine to recognize noisy input. In addition, input is incrementally added to the Bayesian Network as the sketch is being drawn. The system is designed to be dynamic and flexible. [Alvarado and Davis, 2005]

- **Pruning Process:** The system prunes unlikely interpretations.

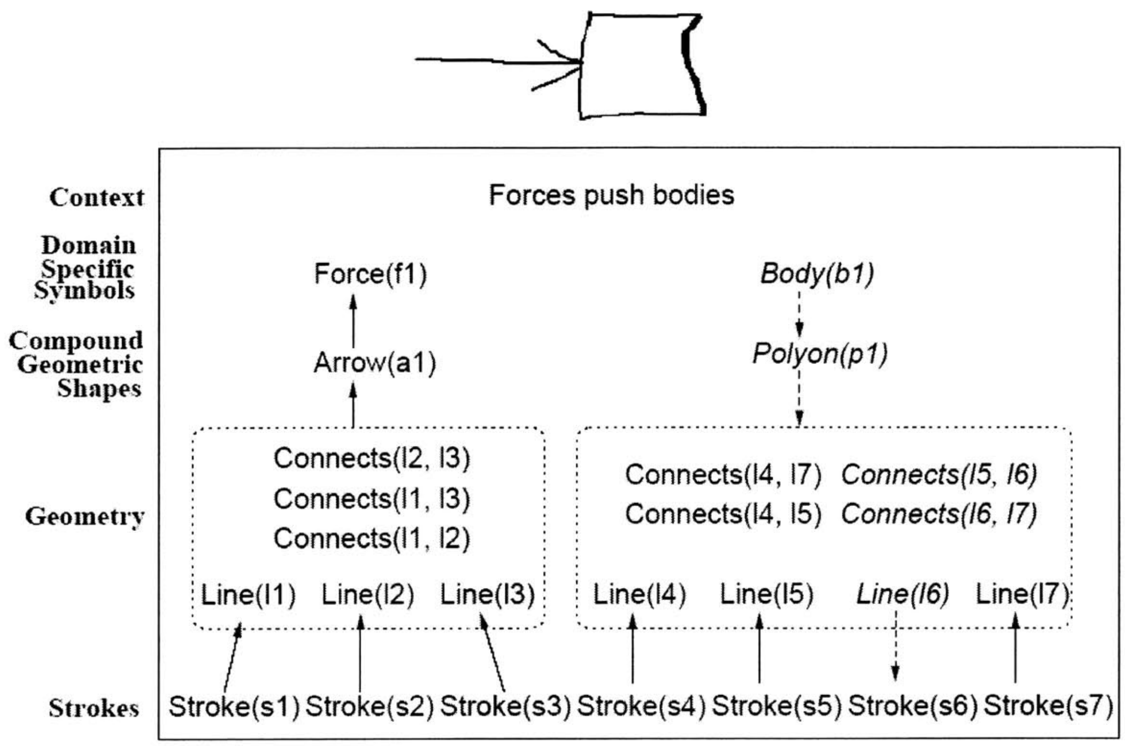


Figure 4.4 Recognition Process in the drawing of a force pushing on a rectangular body.

As shown in Figure 4.4, the blackboard data structure is divided into various levels of information. Italics indicate that an object was hypothesized by the system. In most cases, bottom up information causes the recognition of higher level objects, but in the case of the mechanical body, top down information reinforces the system's hypotheses.

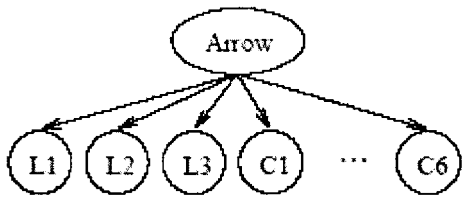


Figure 4.5 Bayesian Network fragment constructed from the description of an arrow.

Shown in Figure 4.5, the Bayesian network created from the description of an arrow. The Bayesian network integrates the influence of stroke data and domain specific context in recognition. In this example, Ls denote the lines and Cs denote the connections. The likelihood of the Bayesian network is used as a score in the recognition of different objects.

It is important to note that because the recognition algorithm is stroke-based, spurious lines and over-tracing hindered the system's performance in both accuracy and running time. A preprocessing step to merge strokes into single lines greatly improved the system's performance. This approach yielded great results with 2D sketches across multiple domains.

For a more comprehensive summary on sketch recognition, see [Alvarado, 2004].

4.4 Architecture

Version one of CLP is designed to interpret and aggregate handwritten text and arrows using existing state of the art recognizers. Interpreting both text and arrows presents a challenge because recognizers, e.g., the Microsoft English Handwriting Recognizer and the Sketch Recognizer from the Design Rationale Group at MIT [Alvarado and Davis, 2004], are able to recognize text or a sketch, respectively, but not both in the same sequence of ink.

In the context of mathematics symbols and mechanics, Dr. LaViola from Brown University was able to accurately recognize mathematic symbols. In addition, he extracted semantic context from the mathematical equations written by users and used this information to animate physics experiments. [LaViola et. al, 2004] Our goals are similar, so we replicated a similar architecture. The CLP ink interpreter uses a two tiered architecture similar to that of LaViola's MathPad program:

- An *Interpreter* that performs recognition and semantic information extraction from digital ink.
- A *Renderer* that renders and displays semantic representation of digital ink. This module displays to the user what it "thinks" the user input was. Rendering is useful for ensuring that the recognized information matches the user's input. (Building a recognizer from scratch such as MathPad is beyond the scope of this thesis.)

Unlike MathPad, however, our interpretation happens synchronously, i.e., after the user inputs ink. While asynchronous interpretation provides less time to complete, it requires the use of an event-based system within Classroom Presenter[1]. Doing so would break the separation between CLP and Classroom Presenter and thus would require more maintenance when time for software upgrades. The performance of synchronous interpretation is negligible for a sentence or two of handwritten text, so we thought it more advantageous to keep ink interpretation separate from Classroom Presenter. The architecture, however, is modular enough to enable a switch to an asynchronous architecture in the future if desired.

In addition, the user does not see the recognition results because we do not want the user to worry about the accuracy of the interpretation. The aggregator [Smith 2006] takes into account interpretation errors by acknowledging the recognition confidence provided by the interpreter. Furthermore, the system is designed to give the instructor an overall picture of understanding in the classroom, and is not designed to give an exact account of how many students got a correct answer.

---

[1] CLP is built on top of Classroom Presenter.

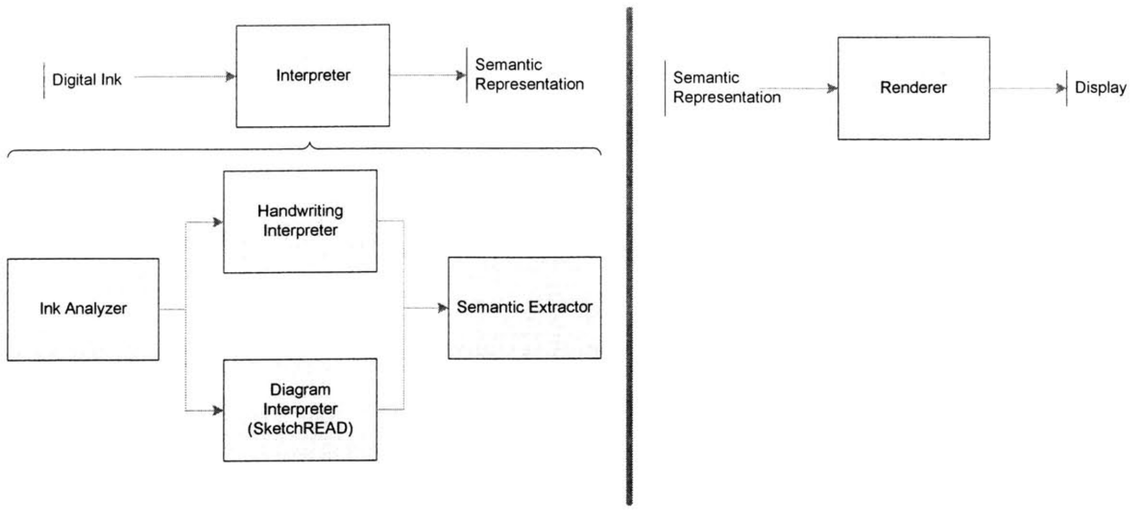


Figure 4.6 Ink Interpretation Architecture

Shown in Figure 4.6 is the ink interpreter architecture, which leverages the power of the two recognizers mentioned above. The Ink Analyzer segments text and sketches and passes corresponding strokes to the appropriate recognizer. Semantic information then is extracted from partial results.

Shown in Figure 4.7 is a diagram we would like the system to interpret eventually: an environment diagram from the MIT introductory computer science course (6.001). The environment diagram has text and drawings intertwined. The challenge will be to differentiate between the drawings and text.

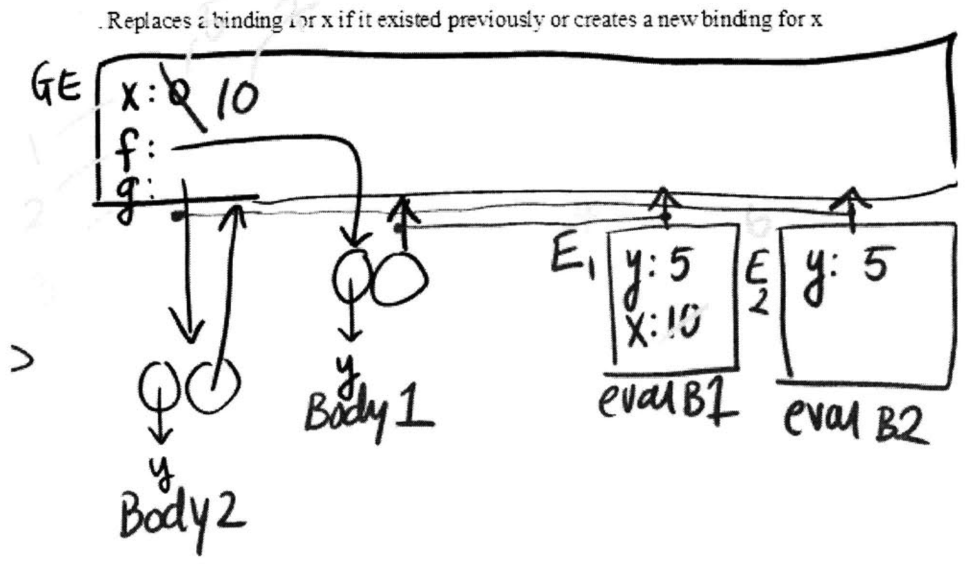

Replaces a binding for x if it existed previously or creates a new binding for x
GE
x: 10
f:
g:
E1
y: 5
x: 10
E2
y: 5
y
Body 1
y
Body2
eval B1
eval B2

Figure 4.7 Environnent Diagram

As our first text and sketch interpretation task, we will start with a simpler box and pointer diagram shown in Figure 4.8.

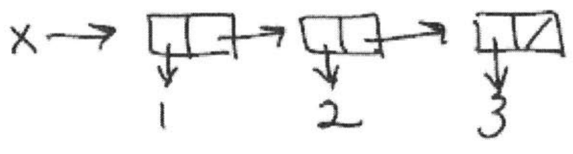


Figure 4.8 Box and Pointer Diagram

This task falls in line with current research being done on chemical structure recognition as well as some research done in vision. In his Phd thesis, Sajit Rao extracted the structure of a directed graph and rendered it, as shown in Figure 4.9. [Rao 1998]

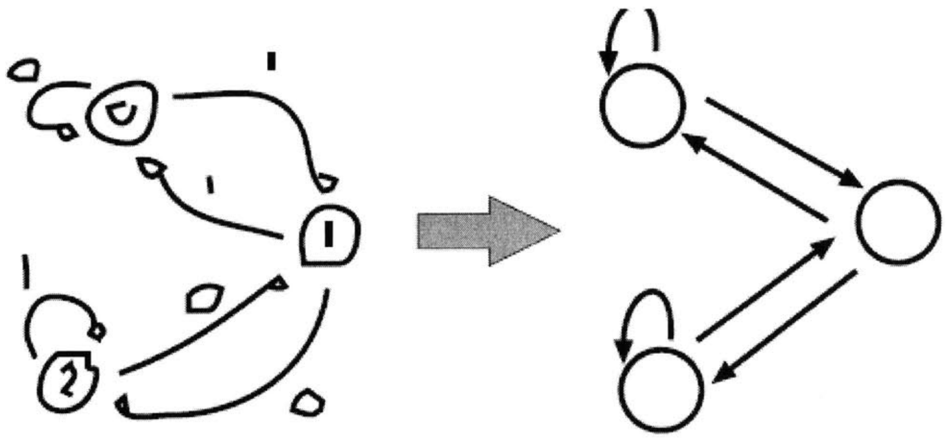


Figure 4.9 Directed Graph Recognition and Rendering [Rao 1998]

Another challenge that the interpreter currently faces is to convert the ink format into some sort of intermediate language such that a diagram or text can be reconstructed or aggregated. In this thesis, I refer to the intermediate language with the term *semantic representation.*

The first step taken in implementing our ink interpreter was to implement the handwriting component, which interprets text and arrows.

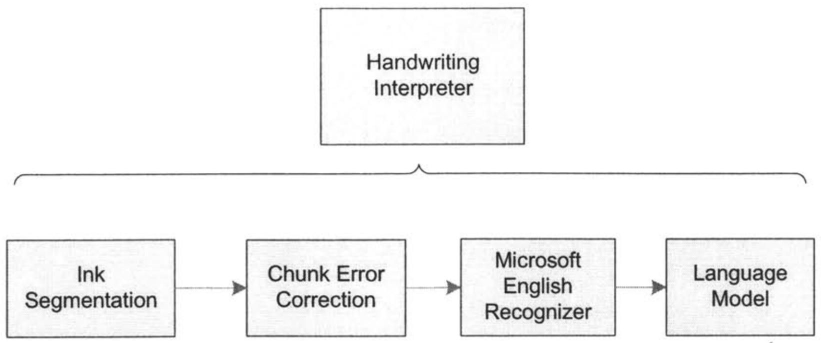


Figure 4.10 Handwriting Interpreter Architecture

Shown in Figure 4.10 is the architecture of the handwriting recognizer:

- The *Ink Segmentation* module segments ink into individual chunks. Chunks are elementary units that are supposed to be individual words or arrows.
- The *Chunk Error Correction* module attempts to fix common errors common of the Ink segmentation process such as splitting a word into two words, or combining two words into one.
- The strokes of each chunk then are passed to the *Microsoft English Recognizer* which outputs several hypotheses ranked by a confidence score.
- The hypotheses then are sent to the *Language Model* module, which uses a domain-specific dictionary and knowledge of expected exercise answer type to choose the best hypothesis.

4.5 Implementation

Scenario

The following example illustrates the interpreter in the context of a class exercise. The instructor asks students: "What is the type of the following Scheme expression: `(lambda (a b) (+ a b))`." The answer to the question is: 'number, number → number'. Other answers such as `'num, num → num'`, `'#, # → #'` or any of their derivatives are all considered correct. A student writes `'number, number → number'` on his/her tablet. The ink segmentation module identifies four chunks as shown in Figure 4.11:

- `'number,'` tagged as a word
- `'number'` tagged as a word
- → tagged as a drawing

- `'number'` tagged as a word

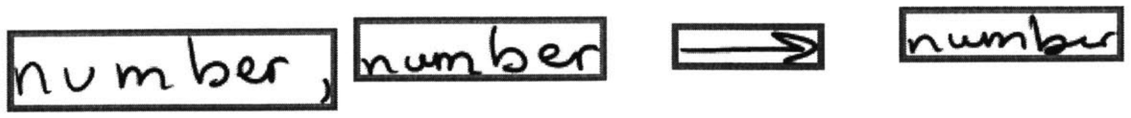


Figure 4.11 Ink Segmentation Illustration

The Chunk Error Correction module identifies no errors and passes the chunks to the recognizer which outputs a list of results for each chunk, ordered by confidence. The output is fed to the language module, which outputs the following semantic representation:

```
<Answer Type="SEQUENCE">
      <Chunk Type="STRING" Confidence="Strong">number,</Chunk>
      <Chunk Type="STRING" Confidence="Strong">number</Chunk>
      <Chunk Type="ARROW" Confidence="Intermediate"/>
      <Chunk Type="STRING" Confidence="Strong">number</Chunk>
</Answer>
```

The renderer takes the semantic representation and outputs the string:

'number, number → number.'

The remainder of this section focuses on the implementation details of the above example.

Software Architecture

Figure 4.12 is the UML diagram for our implementation. The conceptual design shown in Figures 4.6 and 4.10 was implemented as four modules:

- *Recognition:* The recognition package was responsible for Ink Segmentation, Chunk Error Correction, and Recognition
- *Semantic Representation:* This package defined the semantic representation and XML formatting necessary. It was used to manipulate the semantic representation.
- *LM:* This package was responsible for the language model
- *Renderer:* This package was responsible for rendering the different semantic representation

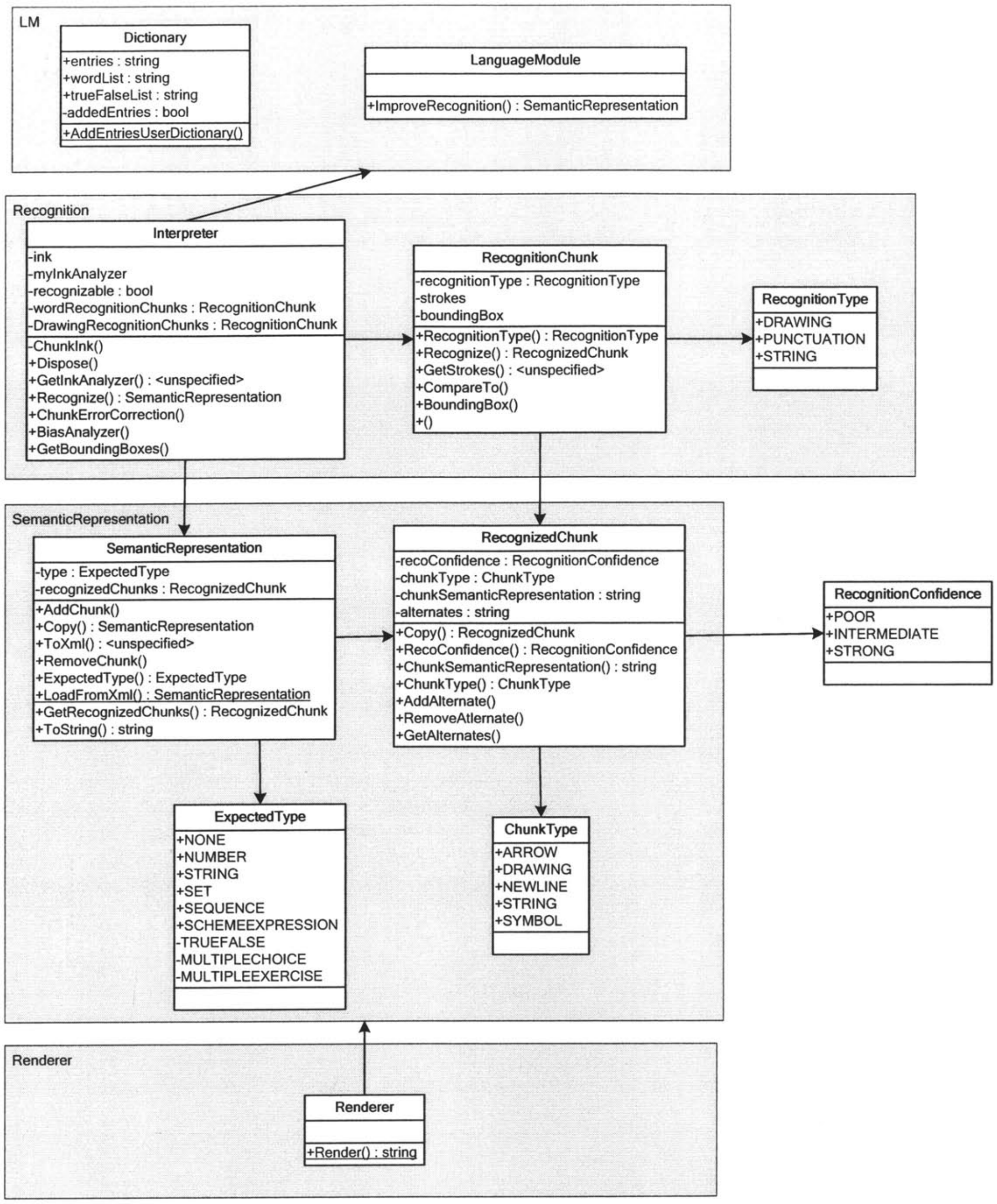


Figure 4.12 UML Diagram of Handwriting Recognizer

Calling the handwriting recognizer

The handwriting recognizer is called via the Interpreter class:

```
Interpreter interpreter = new Interpreter(Microsoft.Ink.Ink ink);
SemanticRepresentation semRep =
interpreter.Recognize(ExpectedType.SCHEMEEXPRESSION);
```

The interpreter is initialized with the ink object containing the strokes. It then is called using the *Recognize* method. The expected type of the answer is used to help bias results and therefore improve accuracy. ExpectedType can take multiple values:

- TRUEFALSE: true-false answers such as '#t' or 'true'
- MULTIPLECHOICE: multiple choice questions where one letter or number is the answer such as 'A' or '1'
- MULTIPLEEXERCISE: this is for multiple exercises per slide (i.e., multiple ink objects), which has not yet been implemented
- STRING: the answer is expected to be one or two words such as 'error', usually put in one chunk.
- NUMBER: the answer is expected to a number such as '1000'
- SET – the answer is expected to be an unordered set of numbers of strings such as '1, 3 , 2'
- SEQUENCE: the answer is expected to be an ordered set of number or strings (including arrows) such as '(1 2)' or 'boolean → string'
- SCHEMEEXPRESSION: A scheme expression is code in the Scheme Language such as '(lambda (a b) (+ a b))'.

More examples are available in Appendix A.

Ink Segmentation

The Ink Segmentation module attempts to segment words and drawings from handwritten ink. This module was implemented using the Ink Analyzer module from Microsoft. Chunking user input provided us with a divide and conquer approach, which enabled us to do recognition error correction in the Language Model. This module, however, had the following shortcomings:

1. Combining two words (or a word and a drawing) into one chunk. Shown in Figure 4.13, the third chunk consists of '# →' where '#' and the arrow should each be in their respective chunks.

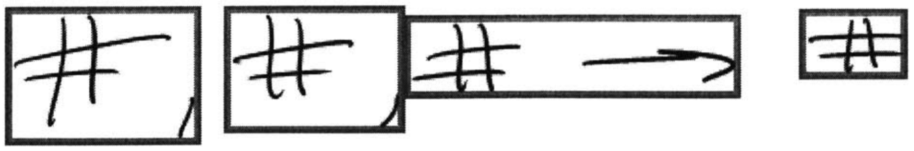


Figure 4.13 Combining a word and a drawing in the same chunk

2. The word chunks and drawing chunks were ordered by type; words were ordered in a separate list than drawings. The Ink Analyzer assumed that drawings were not part of text. Order is defined as to which word or drawing comes first in an English sentence.
3. Dividing one word into two chunks as is illustrated in Figure 4.14.

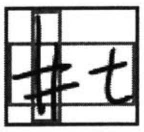


Figure 4.14 Dividing one word into two chunks

4. Classifying drawings as words and vice-versa

There are no details on how the internals of the Ink Analyzer from Microsoft operate but I believe that it attempts to combine ink strokes that form legal words in the dictionary.

Chunk Error Correction

I only dealt with some of the issues presented because the solution to all four of the above issues involves writing a new segmentation module. To deal with chunk ordering (issue 2), I wrote a sorting function by implementing the `IComparable` interface in C# and using the already available Sort method within `System.Collections`. I make the assumption that people write in quasi-straight lines, thus enabling the system to compare two chunks at a time. If order were defined in a more complicated way, then the proposed approach would fail.

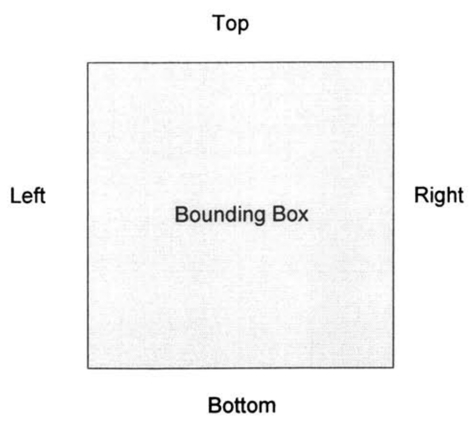


Figure 4.15 Box Visualization

The sorting function compares the location of strokes' bounding boxes ignoring context. It returns 0 if the bounding boxes of the ink strokes superimpose each other, -1 if the first bounding box comes before the second, 1 otherwise.

```
Rectangle thisBoundingBox = this.BoundingBox;
Rectangle otherBoundingBox = otherChunk.BoundingBox;

if (thisBoundingBox.Equals(otherBoundingBox))
// this compares if the boxes superimposes one onto the other
{
   return 0;
}
//check that they are on the same line:
//- check that upper left corner of first box is higher than lower
left //corner of second box
//- check that lower left corner of first box is lower than upper
left //corner of second box
//check that first box comes first:
//- check that upper left corner of first box comes before than
upper left corner of second box
//remember that y coordinates increase in the downward direction
if ((thisBoundingBox.Top <= otherBoundingBox.Bottom) &&
    (thisBoundingBox.Bottom >= otherBoundingBox.Top) &&
    (thisBoundingBox.Left <= otherBoundingBox.Left))
{
   return -1;
}

// this the case where the first box is either the first word
//or above the second box
else if (thisBoundingBox.Bottom <= otherBoundingBox.Top)
{
   return -1;
}
return 1; //if all else fails, then the boundingBox is in front of
the other
```

Figure 4.16 Implementation of the CompareTo method of the IComparable Interface

Figure 4.16 outlines the details of implementing the comparator function, which bundled with the quick sort implementation in C#, enabled me to sort through the different chunks. I only dealt partially with issue 3 by first sorting all the chunks and then combining all chunks with intersecting bounding boxes. The reason the solution is partial is because if a person does not write in cursive but in a rather detached form, it is still possible for the system to identify 'c ar' as two chunks, 'c' and 'ar' instead of one chunk 'car'. For example, Figure 4.17 illustrates a case where the resorting algorithm fails.

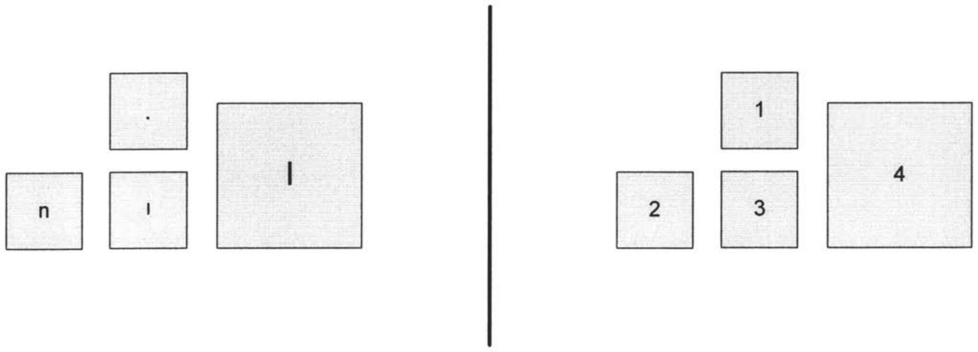


Figure 4.17 Illustration of Chunk Ordering Problem

Figure 4.17 illustrates wrong chunking of the word 'nil' where each letter is placed in a different chunk and the dot on top of the letter 'i' is also placed in a separate chunk. The numbers in the boxes correspond to the ordering of chunks as performed by my algorithm, which poses a small problem for recognition. When recognition is performed, the interpreted result is: '. n * l'. A hypothesis as to why this happens is that the two mechanisms I can think of to avoid this problem fail in this specific example:

1. The ink segmentation module checks the chunks against a dictionary. Each letter is a valid entry of the dictionary, which justifies the chunking.
2. The example is written in non-cursive handwriting where the characters are not attached together. I assume that the program dynamically computes spacing between letters, adjusted to each person's handwriting. When a person writes a sentence, and words are accurately recognized, the program dynamically computes the space between characters in order to find the separation between words. In this example containing one word, the program could not rely on results of remaining words to deduce a robust indicator of spacing between letters, and thus assumed that the user wrote individual chunks.

Dealing with issue 4, where the ink segmentation module classified words as drawings and vice-versa, relied on a bias of the expected answer type. Unless the type is a sequence, there is no reason to expect the handwriting to include any drawing. In addition, if the answer is anticipated to be short, such as numbers or true-false answers, the interpreter put all the strokes of the answer in the same chunk, thus side-stepping the possible problems of issue 3, when words are divided into two chunks.

Recognition

Once the chunks are identified, they are stored in a list of `RecognitionChunk` objects. This list of chunks is ordered according to the method described in Figure 4.16 and each chunk only contains the strokes of that chunk. The recognizer thus is isolated from the context of the sentence, leaving context handling to the language model. In addition, the recognizer is biased in two ways:

- By adding a list of words to the user dictionary that are not typically available with a standard recognizer such as: 'set-cdr!' or 'caar' which are only relevant in the current domain
- By biasing based on the expected answer type. Knowing about a true-false answer, I identified six possible answers: '#t', '#f', 'true', 'false', 't', 'f' and coerced the recognizer in this direction. This list is not exhaustive and more user testing augments the dictionary. It was just brought to my attention, for example, that some students may indicate true and false using bits, 1 being true, and 0 being false. The recognizer is also biased using the `Factoid` capability of the Microsoft Recognizer which had multiple features, such as being able to bias towards numbers, which is useful when expecting a number.

The recognizer passes recognition results ordered by recognition confidence to the language model. Recognition confidence is an internal measure of the confidence the recognizer has in its results; in the Microsoft Recognizer API, only three values are provided: Poor, Intermediate, and Strong. A quantitative value based on probabilities would have been more useful in this case.

Language Model

The language model module used is basic and consists of a lexicon of words. The architecture allows for a future increase in complexity. The language module is only called only if the recognition results are poor. I turned off the Microsoft Recognizer Language Module because it was not well-suited for Scheme[1] code: the language module had been developed for natural language and performed badly with out-of-dictionary words such as the ones in Scheme. When recognition is poor, the language model module compares the various hypotheses of each chunk against entries in our dictionary. Because the recognition hypotheses are ordered by recognition confidence, the program picks the highest ranked hypothesis in the list of hypotheses. In the case in which an arrow is expected (i.e., the answer is of type SEQUENCE), the program waits for all the recognition hypotheses to be compared against the dictionary. If none of them matches, the module checks that the potential string is less than 2 characters and hypothesizes that it's an arrow. This method is not solid arrow recognition, but the architecture lends itself to allowing small drawings to be part of the text. This method eventually will be replaced with an object recognizer, but for the time being, this hack does reasonably well.

Semantic Representation

The interpreter outputs the following semantic representation for the example shown in Figure 4.11:

```
<Answer Type="SEQUENCE">
   <Chunk Type="STRING" Confidence="Strong">number,</Chunk>
   <Chunk Type="STRING" Confidence="Strong">number</Chunk>
   <Chunk Type="ARROW" Confidence="Intermediate"/>
   <Chunk Type="STRING" Confidence="Strong">number</Chunk>
</Answer>
```

[1] The introductory computer science course uses the Scheme programming language.

The semantic representation contains the recognition result of each chunk and the confidence associated with the result. The representation potentially could be augmented to include the list of alternatives if an outside program such as the aggregator deemed it necessary. The confidence is included so the aggregator can determine the amount of flexibility needed in dealing with interpreter errors.

The following graph can be constructed from the XML representation for the semantic representation:

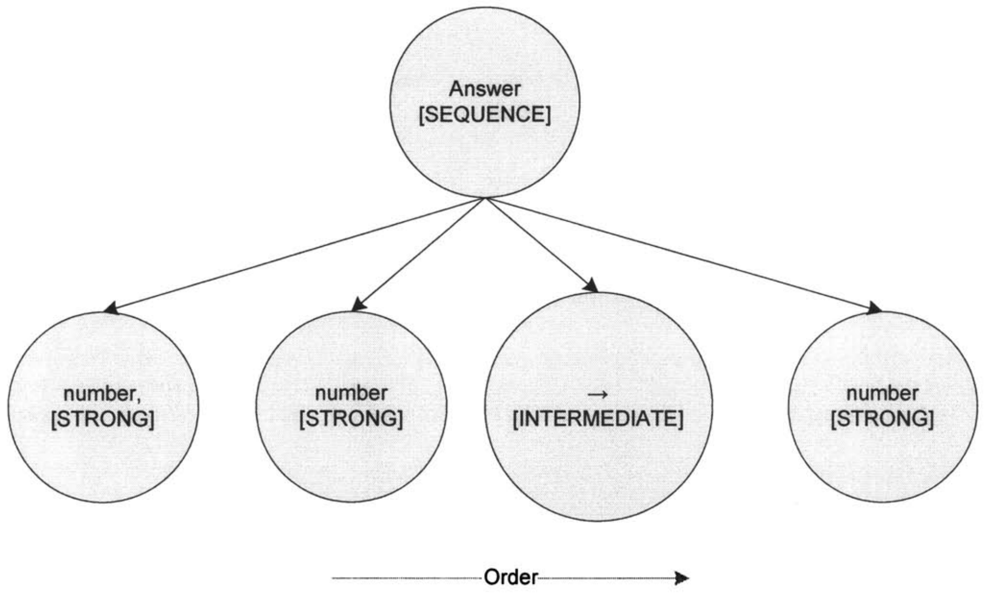


Figure 4.18 Graph for Semantic Representation

The graph for a string answer is uninteresting. In a box and pointer diagram, however, it can provide a powerful tool for aggregation. Let's take the simple diagram shown in Figure 4.19:

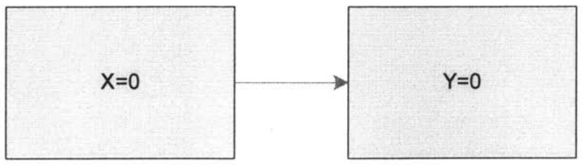


Figure 4.19 Simple Box and Pointer Diagram

Here is the hypothetical semantic representation for the diagram:

```
<Answer Type="BOXANDPOINTER">
   <Box id=1>
         <Text Confidence="Strong">X=0</Text>
         <Position x=0 y=0/>
```

```
        </Box>
        <Box id=2>
              <Text Confidence="Strong">Y=0</Text>
              <Position x=100 y=0/>
        </Box>
        <Pointer>
              <Tail>1</tail>
              <Head>2</head>
        </Pointer>
    </Answer>
```

The reconstructed graph would correspond to the one in the original diagram. This graph representation would simplify aggregation because classification of graphs is a well-studied problem. Furthermore, the representation would enable a renderer to reconstruct the interpreted input.

Renderer

The renderer displays the semantic representation in a human readable form. For the time being, it converts the XML representation into a simple string, but it eventually will be able to reconstruct an image from a text description.

Alternatives

I considered initially using the built-in Microsoft Recognizer and Analyzer. Early results, however, were very bad, prompting the alternative architecture with separate components for the ink segmentation and the language module.

4.6 Results

Recognition accuracy traditionally has been measured with a Word Error Rate. Due to the nature of our test cases, however, it was more appropriate to test the distance between the input and recognized strings in order to test for partial improvements. If the input string was "caar," for example, and the subsequent recognition results were "cr" and "car", it was important for our accuracy measure to take into account the partial progress made, something a word error rate measurement would not do. The *Levenshtein distance* [Atallah 1998], also called *edit distance,* accurately measures the distance between two strings. The distance between two strings is given by the minimum number of operations needed to transform one string into the other, where an operation is an insertion, deletion, or substitution of a single character. In the above example, the edit distance between "caar" and "cr" is 2, while the distance between "caar" and "car" is 1.

The interpreter was tested in two experiments:

- A controlled experiment in which users were asked to ink predetermined answers, and the interpreted results were compared to the actual answers. This experiment was performed multiple times to test the different versions of the interpreter.
- In-class experiments in which I manually compared student submitted answers and their interpretations. Not knowing what the prior answers were only allowed for manual comparison. This experiment was used to understand user behavior and how the interpreter did in a real-world situation.

In the controlled experiment, users were asked to ink twenty-one representative answers (Appendix A); I collected 167 inked answers from several users. These sample answers were stored in a database. The inked answers were dynamically interpreted, then stripped of spaces, and converted to lower case. They then were compared to the input. Samples were collected early in the process in order to test each version of the interpreter against all these answers. This process provided a consistent way to check for improvements in recognition accuracy. The design of the interpreter sometimes involved tradeoffs, and measuring increasing accuracy on the same data set was a reliable way to ensure progress. The way I measured improvement was to check whether the total edit distance between interpreted answers and inputs decreased. The total edit distance between interpreted answers and inputs also could be regarded as the total number of character errors between interpreted answer and input, a character error being a character deletion, insertion or substitution. I present the results of recognition of the current version of the interpreter. The experiment used the examples shown in Appendix A., which illustrate four of the seven answer types present in the database. The following types are subsets of other types and necessary for operation:

- MULTIPLE EXERCISE: this type identifies that the exercise contains multiple ink objects; it is a collection of strings and has not been implemented yet.

- SET: this type is a subtype of sequence because it is an unordered list, which the aggregator distinguishes from ordered sets (i.e., sequences). For the purposes of the interpreter, a set is the same as having a sequence.
- NUMBER: the handwriting recognizer has been built with this type in mind, therefore, using it presents no problems. Number and character recognition are the traditional problems handwriting recognition research has been solving. Current handwriting recognizers have 95% accuracy (i.e., 5% Word Error Rate) when recognizing numbers or characters. [MacKenzie and Chang, 1999]
- MULTIPLE CHOICE : this type is simply single character recognition.

Table 4.1 presents the results grouped by answer type and sorted by length in ascending order.

Type	**Input Text**	**Input Length**	**Mistakes**	**Total Char**	**Error Rate**
Scheme Expression	(1 2)	5	2	32	6%
Scheme Expression	(* 1 2)	7	8	40	20%
Scheme Expression	(caar seq)	10	15	72	21%
Scheme Expression	(define x 3)	12	5	80	6%
Scheme Expression	(eq? id1 id2)	13	15	88	17%
Scheme Expression	(lambda (a b) (+a b))	21	17	119	14%
Scheme Expression	(car (quote (quote a)))	23	18	160	11%
Scheme Expression	(set-cdr! (last-pair x) x)	26	17	184	9%
Scheme Expression	(cons (cdar seq) (cddr se))	27	15	184	8%
Scheme Expression	(cons (cons x (+ 1 (+ 1 (seq-length seq)))	42	58	272	21%
Scheme Expression	(define x (let ((two '(2))) (list (cons 1 two) (list 1) two)))	65	82	416	20%
Sequence	#, #, # -> #	12	17	56	30%
Sequence	boolean -> string	17	4	112	4%
Sequence	nbr, nbr, nbr -> nbr	20	15	120	13%
String	Nil	3	0	24	0%
String	Error	5	0	40	0%
String	double-tree	11	3	88	3%
True-False	#t	2	0	16	0%
True-False	#f	2	2	16	13%
True-False	True	4	0	32	0%
True-False	False	5	0	40	0%

Table 4.1 Interpretation Results ordered by Type and Input Length

Error Rate was measured by dividing the number of mistakes over total number of characters. Figure 4.20 shows a graph of Table 4.1 results.

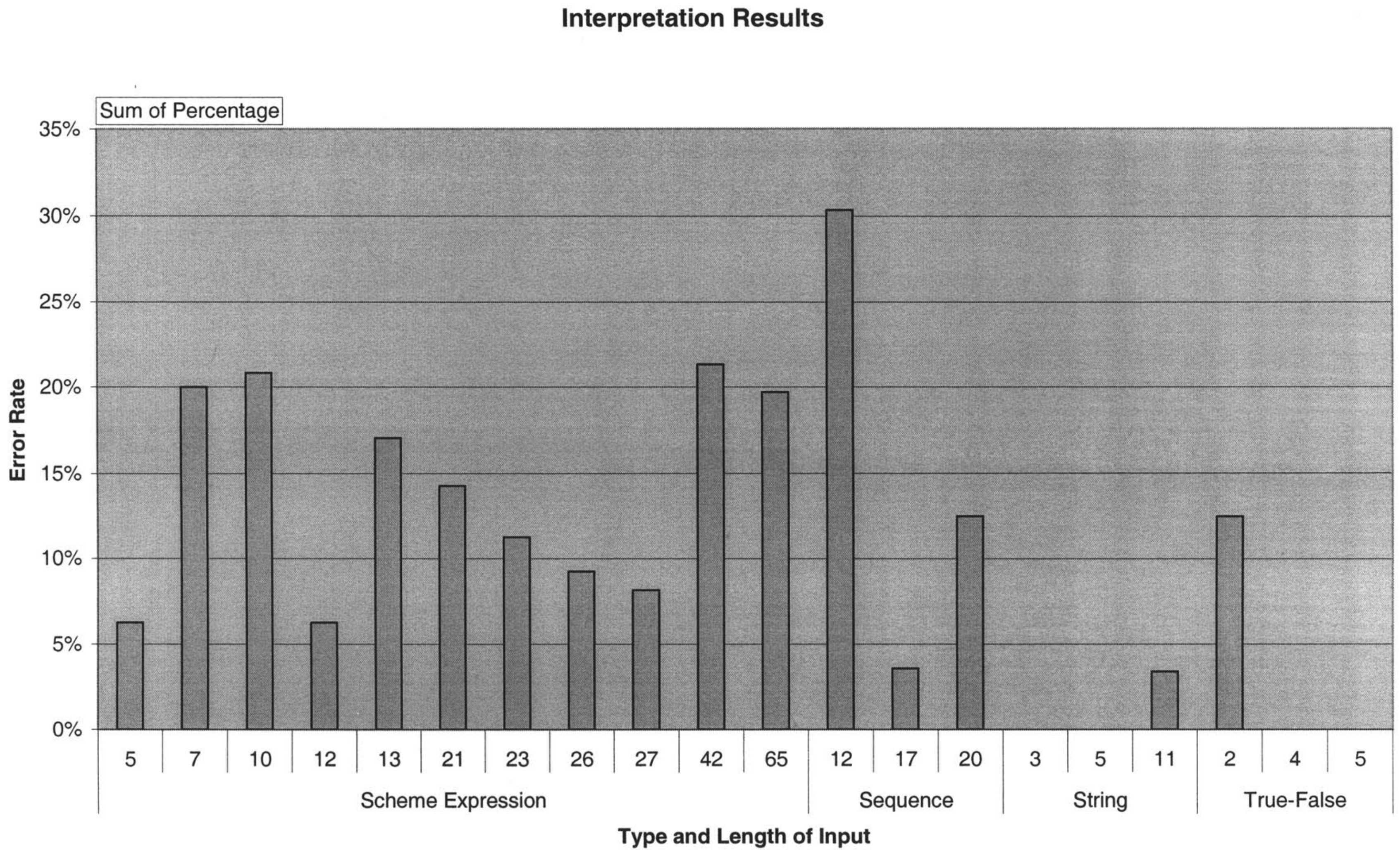


Figure 4.20 Graph of Interpretation Results ordered by Type and Input Length

Finally, Table 4.2 summarizes the accuracy of the interpreter per type.

Answer Type	Error Rate
Scheme Expression	15.30%
Sequence	12.50%
String	1.97%
True-False	1.92%
Total >>	**13.37%**

Table 4.2 General Accuracy of the Interpreter

CLP has been deployed three times at the time of writing. Results of recognition were used to augment the dictionary. In addition, the interpreter did not do very well with messy handwriting. Knowing that even humans have trouble with messy handwriting, the result was not surprising. CLP will be deployed next academic year 2006-2007 again in introductory computer science classes.

4.7 Discussion

In general, the system performs well, and the deployments were a success because the aggregator was able to group interpreted ink into groups and display the results, despite interpretation errors. 13% error rate (or 87% accuracy rate) is not yet excellent but is acceptable for a first iteration. I expect to reach 5% error rate, the industry standard, if all the changes proposed in the Future Work section are implemented.

When taking a closer looks at the results by type, the accuracy for string and true-false questions is 98%. This result, combined with experiments performed on number and character recognition, clearly demonstrate that our system can rival wireless polling systems such as PRS, in which a student submits an answer to a multiple-choice or true-false question using a transmitter. Furthermore, CLP's ink interpreter allows student submission and aggregation of more than multiple-choice or true-false questions.

Difficulties arise when interpreting Scheme expressions and sequences. If we take a closer look at Figure 4.18, accuracy is not closely correlated with the length of the input. I hypothesize that it has to do with the nature of the input. The interpreter has more trouble, for example, with the shorter input `'#, #, # → #'` than with `'boolean → string'`. In the rest of this section, I discuss the most commonly observed errors.

Incorrect Chunking

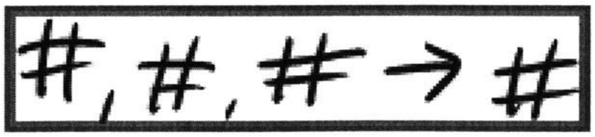


Figure 4.21 Illustration of Incorrect Chunking

In addition to the previously shown example in Figure 4.11, the ink segmentation incorrectly chunks the input, thus, causing the arrow not to be identified.

Ambiguity in the writing

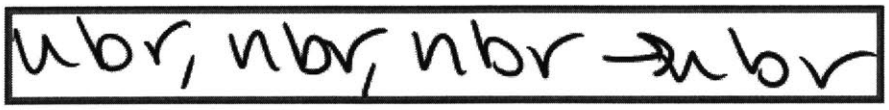


Figure 4.22 Illustration of Ambiguity in Writing

The rendered interpreter result for this exercise was: 'ubr,nbr,nbrsubr.' To credit the interpreter, the 'n' in 'nbr' looks like a 'u.' It is only because we know the context of the question that we know that it was an n that was used not a u. Furthermore, connecting the arrow with the last word misleads the analyzer into thinking that it is part of the same word.

Incorrect Recognition

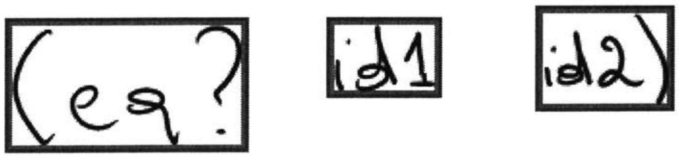


Figure 4.23 Illustration of Incorrect Recognition

The rendered interpreter result for this exercise is: 'lee? id At id2)'. It correctly chunked the input but poorly recognized it.

Confusing letters and punctuation

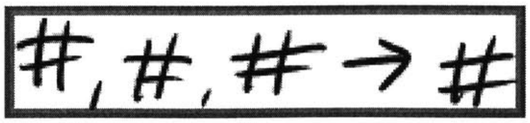


Figure 4.24 Illustration of Confusing punctuation with letters

The interpreted result for this exercise was: '#i#i#7#'. Besides incorrectly chunking the result, the recognizer confused the commas with the letter 'i'.

Confusing numbers and parentheses

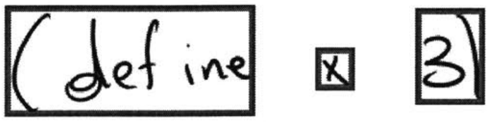


Figure 4.25 Illustration of Confusing parentheses and numbers

The interpreted result for this exercise was: '(define x 31'. In this example, the interpreter mistakes the last parenthesis for the number 1.

Space Elimination

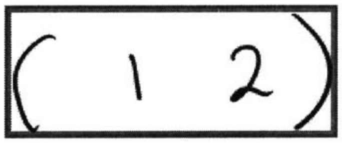


Figure 4.26 Illustration of Space Elimination

The interpreted result for this exercise was: '(12)'. This elimination creates a problem in sets because the machine thinks that the user input a list containing the number 12 rather than a list containing the two numbers 1 and 2.

The improvements proposed in the Future Work section attempt to solve the problems outlined in this section.

5 Future Work & Contributions

5.1 System Improvements

From a functional standpoint, CLP could benefit from two improvements:

- Displaying the results of the interpretation to students and allowing students to correct the interpretation results. The current system does not display results of interpretation, providing no way for students to check if the system "understood" their submission. We originally did not include a correction interface for fear that students would spend too much time making sure their answers were interpreted perfectly. Students have, however, expressed a desire for such a feature, and our classroom deployments have demonstrated that using Classroom-Presenter-like systems is efficient, so we need not worry about time taken to check interpretation results.
- Providing feedback on student answers. Online tutors such as the xTutor are used in the introductory computer science class, Structure and Implementation of Computer Programs. The tutor allows distribution of lecture notes, and problem sets, with formats ranging from simple multiple choice and fill-ins, to programming assignments with automatic checking of results. The desired functionality would be similar to checking of submissions in the online tutor in that students would be able to check their answers and get hints for solving the problems. [xTutor 2006]

These two functional improvements would offer direct feedback to students, enabling them to build more trust in the system.

5.2 Interpretation Improvement

Interpretation can be improved in the following ways:

- Training the recognizer with examples from the introductory computer science class, biasing the recognizer to the Scheme Programming language. A posteriori biasing is accomplished in the current version of the system, but the system is not trained on this data.

- Training the recognizer on the machine to the user's handwriting. This training is only possible, however, if the user maintains the same machine throughout the course.
- Writing a simple object recognizer that recognizes arrows and boxes. This interpreter would yield better results than the current one in that it would look for specific arrow-like shapes rather than hypothesizing the presence of an arrow based on expected answer type. A light version of SketchREAD [Alvarado 2004] would work well, recognizing only what we need (namely arrows and boxes).
- Improving the performance of the Ink Segmentation module. This improvement can be done by tweaking the ink segmentation module to emphasize empty spaces over words. Currently, the ink segmentation module is more likely to put two words in the same chunk rather than to keep individual letters in a word separate.
- Updating the language module to take advantage of syntax constraints. The introductory computer science class is taught using the Scheme Programming language, which gives us a syntactic structure to follow. Often the recognizer confuses the number 1 and parentheses. For example, the student writes `(+ 1 2)` but the interpreter returns `1 + 1 2 1`. This expression is illegal in the Scheme programming language. Two constraints could improve recognition in this example:
    - An operator is preceded by a parenthesis.
    - Each opened parenthesis is matched by a closed parenthesis.

If the interpreter is aware of such constraints, it will choose an alternate recognition result matching the constraints. Note that an improved language model would not cure all problems. The example `1 2 1,` for example, is unsolvable by the language constraints because the student could have written `1 2 1` or `(2)`. The interpreter uses other workarounds such as the context of the question to solve problems. In the above problem, the interpreter could use the expected type to determine the most likely answer. The first solution, `1 2 1,` is the right one if a sequence is expected, and the second solution, `(2),` if a Scheme Expression is expected.

5.3 Contributions

In this thesis, I have made the following contributions towards building a learning partner for the classroom:

- I have implemented the first version of an ink interpreter which is capable of text and arrow interpretation. The interpreter architecture is separated into 4 processes, each of which lends itself to upgrades:
    - Ink Segmentation
    - Chunk Error Correction
    - Recognition
    - Language Model
- I have laid out a framework for sketch and text interpretation in the Classroom Learning Partner system. The framework is inspired from state of the art research in handwriting and sketch interpretation and will serve as the basis for version 2 of the Classroom Learning Partner (to be developed from June 2006 to June 2007).
- I have demonstrated that the interpreter accuracy can rival PRS on multiple choice and true-false questions. Our system, however, has the advantage of not being limited to multiple choice and true-false questions, enabling instructors to ask open-ended questions.
- I have designed and helped implement the infrastructure necessary for creating in-class exercises as part of Classroom Presenter, interpreting student answers, and aggregating them. I have also helped deploy this infrastructure successfully two terms in a class of 20 students. The architecture is modular and has a clear separation from Classroom Presenter. The infrastructure enables communication between the different software modules created by members of the CLP group. It also acts as persistent data storage important for later in-class operation and after-class data mining.

References

[Abelson et. al, 1996] Abelson, H., Sussman, G., Sussman, J., *Structure and Interpretation of Computer Programs,* MIT Press and McGraw-Hill, 2nd edition, 1996.

[Alvarado et al 2002] Alvarado, C., Oltmans, M. and Davis, R., *A Framework for Multi-Domain Sketch Recognition*, in Proc. of AAAI 2002.

[Alvarado, 2004] Alvarado, C., *Multi-Domain Sketch Understanding*. PhD thesis, Massachusetts Institute of Technology, 2004.

[Alvarado and Davis, 2004] Alvarado, C. and Davis, R., *SketchREAD: A MultiDomain Sketch Recognition Engine,* in Proc. of UIST 2004.

[Alvarado and Davis, 2005] Alvarado, C.and Davis, R., *Dynamically Constructed Bayes Nets for Multi-Domain Sketch Understanding,* in Proc. of IJCAI 2005.

[Anderson et al, 2003] Anderson, Richard, Anderson, Ruth, Vandegrift, T., Wolfman, S., and Yasuhara, K., *Promoting Interaction In Large Classes With Computer-Mediated Feedback*, University of Washington, University of Virginia, University of British Columbia, 2003.

[Anderson et al, 2004] Anderson, R., Anderson, R., Simon, B. VanDeGrift, T., Wolfman, S., Yasuhara, K., *Experiences With a Tablet-PC-Based Lecture Presentation System*, University of Washington, University of Virginia, University of British Columbia, 2004.

[Anderson et al, 2005] Anderson, Richard, Anderson, Ruth, Hoyer, C., Prince, C., Su, J., Videon, F., and Wolfman, S., *A Study of Diagrammatic Ink in Lecture,* University of Washington, University of Virginia, University of British Columbia, 2005.

[Anquetil and Lorette, 1995] Anquetil, E., and Lorette, G., *On-Line Cursive Handwrittten Character Recognition Using Hidden Markov Models,* Traitement du Signal, vol. 12, no. 6, pp. 575-583, 1995.

[Atallah 1998] Atallah, M. J. (Editor), *Algorithms and Theory of Computation Handbook,* "Levenshtein Distance (13-5)", CRC Press, 1998.

[Bransford et al, 2000] Bransford, JD, Brown, AL and Cocking, RR., *How People Learn: Brain, Mind, Experience, and School*, National Academy Press, Washington, D.C., 2003.

[Bressler 2004] Bressler, L., *Lessons Learned: Findings from Ten Formative Assessments of Educational Initiatives at MIT (2000-2003),* Report of the MIT Teaching and Learning Laboratory, 2004.

[Chen 2006] Chen, J., *Instructor Authoring Tool Towards Promoting Dynamic Lecture-Style Classrooms*, MEng Thesis, MIT, February 2006.

[Chen et al, 2000] Chen, Z., Lee, K., and Li, M., *Discriminative Training on Language Model*, In Proceedings of the Sixth International Conference on Spoken Language Processing (ICSLP), Beijing, China, 2000.

[Draper 2004] Draper, SW., *From active learning to interactive teaching: Individual activity and interpersonal interaction*, in Teaching and Learning Symposium: Teaching Innovations, The Hong Kong University of Science and Technology, 2004.

[Duncan 2005] Duncan, D., *Clickers in the Classroom,* Addison Wesley, San Francisco, 2005

[Hu et al, 1996] Hu, J., Brown, M., and Turin W., *HMM Based On-Line Handwriting Recognition,* IEEE Transactions On Pattern Analysis and Machine Intelligence, Vol. 18, No. 10, October 1996.

[Koile and Shrobe 2005] Koile, K. and Shrobe, H.E., *The Classroom Learning Partner: Promoting Meaningful Instructor-Student Interactions in Large Classes*, MIT CSAIL TR., 2005.

[Koile and Singer, 2005] Koile, K. and Singer, D., *Educational Assessment for the Classroom Learning Partner*, Massachusetts Institute of Technology, 2005.

[Koile and Singer, 2006a] Koile, K. and Singer, D., *Development of a Tablet-PC-based System to Increase Instructor-Student Classroom Interactions and Student Learning*, To appear in The Impact of Pen-based Technology on Education; Vignettes, Evaluations, and Future Directions. Berque, D., Gray, J., and Reed, R. (editors). Purdue University Press, 2006.

[Koile and Singer, 2006b] Koile, K. and Singer, D., *Improving Learning in CS1 with Tablet-PC-based In-Class Assessment*, Submitted to ICER 2006 (Second International Computing Education Research Workshop), September 9-10, 2006, University of Kent, Canterbury, UK.

[LaViola and Zeleznik, 2005] LaViola, J. and Zeleznik, R., *MathPad: A System for the Creation and Exploration of Mathematical Sketches,* Brown University, 2005.

[Hake 1998] Hake, RR, *Interactive-Engagement versus Traditional Methods: A Six-Thousand Student Survey of Mechanics Test Data for Introductory Physics Courses*, American Journal of Physics, 66(1):64-74, 1998.

[Hammond and Davis, 2005] Hammond, T. and Davis, R., *LADDER, a sketching language for user interface developers*, Elsevier, Computers and Graphics 29 (2005) 518-532, 2005.

[Jacobs and Grainger, 1994] Jacobs, A. M. and Grainger, J., *"Models of visual word recognition- sampling the state of the art,"* Journal of Experimental Psychology, Vol. 20, No. 6, pp. 1311-1334, 1994

[Liu and Cai, 2003] Liu, Z. and Cai, J., *Handwriting Recognition, Soft Computing and Probabilistic Approaches*, Springer, Preface, Chapters 1 , 2 ,3, 2003.

[MacKenzie and Chang, 1999] MacKenzie, I. S., & Chang, L. , *A performance comparison of two handwriting recognizers*, Interacting with Computers, 11, 283-297, 1999.

[Makhoul et al, 1994] Makhoul, J., Starner, T., Schartz, R., and Lou, G., *On-Line Cursive Handwriting Recognition Using Speech Recognition Models,* Proc. IEEE Int'l Conf. Acoustics, Speech and Signal Processing, pp. v125-v128, Adelaide, Australia, 1994

[Pahud 2006] Pahud, M., *ConferenceXP Research Platform: Toward an Extensible Collaborative Environment*, Microsoft Research, 2006, http://www.conferencexp.com/community/default.aspx

[Plamondon and Srihari, 2000] Plamondon, R, Srihari, S., *On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey,* IEEE Trans. Pattern Anal. Machine Intell., vol. 22, pp. 63–85, Jan. 2000.

[Rao 1998] Rao, S., *Visual Routines and Attention,* PhD Thesis, Ch.2, Massachusetts Institute of Technology, 1998.

[Sezgin et al, 2001] Sezgin, T. M., Stahovich, T., and Davis, R., *Sketch based interfaces: Early processing for sketch understanding*, In Proc. of PUI, 2001.

[Smith 2006] Smith, A., *Aggregation of Student Answers in a Real-Time Classroom Setting,* MEng Thesis in preparation, MIT, May 2006.

[Srihari and Keubert, 1997] Srihari, S., and Keubert, E.J., *Integration of Handwritten Address Interpretation Technology into the United States Postal Service Remote Computer Reader System*, Proc. Fourth Int'l Conf. Document Analysis and Recognition, vol. 2, pp. 892-896, Ulm, Germany, Aug. 1997.

[USPS 2005] *USPS, Postal Facts 2005,*
http://www.usps.com/communications/organization/postalfacts2005.htm

[Wikipedia 2006a] *Real-time Transport Protocol (RTP),* Wikipedia, 2006, http://en.wikipedia.org/wiki/Real-time_Transport_Protocol

[Wikipedia 2006b] *Transmission Control Protocol (TCP),* Wikipedia, 2006, http://en.wikipedia.org/wiki/Transmission_Control_Protocol

[xTutor 2006] *XTutor, a toolkit for creating online courses*
http://icampus.mit.edu/xTutor/default.aspx

[Yasuda et al, 2000] Yasuda, H., Takahashi, K., and Matsumoto, T., *A Discrete HMM For Online Handwriting,* International Journal of Pattern Recognition and Artificial Intelligence, Vol. 14, No. 5, pp.675-688, 2000.

Appendix A- List of Representative Examples

These examples were extracted from lectures used in the pilot study [Koile and Singer 2005a] and were the type of question an instructor would want to use.

Number	**String**	**Type**
1	#, #, # → #	Sequence
2	nbr, nbr, nbr → nbr	Sequence
3	boolean → string	Sequence
4	(1 2)	Scheme Expression
5	(* 1 2)	Scheme Expression
6	#t	True-False
7	true	True-False
8	#f	True-False
9	false	True-False
10	(caar seq)	Scheme Expression
11	(cons (cdar seq) (cddr seq))	Scheme Expression
12	(cons (cons x (+ 1 (+ 1 (seq-length seq)))	Scheme Expression
13	(car (quote (quote a)))	Scheme Expression
14	(eq? id1 id2)	Scheme Expression
15	error	String
16	(set-cdr! (last-pair x) x)	Scheme Expression
17	double-tree	String
18	nil	String
19	(define x 3)	Scheme Expression
20	(lambda (a b) (+a b))	Scheme Expression
21	(define x (let ((two '(2))) (list (cons 1 two) (list 1) two)))	Scheme Expression