Team Dijkstra

Crist - Howell - Ghodratnama

t13 - Final Project Design

Email Server/Client System

Software Engineering II

Spring 2013

"Elegance is not a dispensable luxury but a quality that decides between success and failure."
- Edsger Dijkstra

Table of Contents

- Table of Contents i
- Revision Notes ii
- Introduction 1
- Proof of Concept 2
- Data Structure Design 3
  - Email Message Structure 3
  - Address Registration Structure 4
  - Address Book Structure 4
  - Mailing Lists Structure 4
- Component Requirements 5
  - Server 5
  - Client 6
- Module Design 7
  - Server 7
    - ACL2 7
    - Module 11
    - GUI 13
  - Client 13
    - ACL2 13
    - Module 16
    - GUI 18
- Input/Output 19
  - Email I/O 19
  - Email Address Book I/O 19
  - Mailing List I/O 20
- PROBE Estimate and PSP Report 22
- Additional Features 39
  - Overview 39
  - PROBE Estimate 41
- Appendix 43
  - A: ACL2 Source Code 43
  - B: ACL2 Theorems and Test Code 70
  - C: Java Source Code 79
  - D: PSP Source File 121
  - E: PSP Source for Additional Features 144
  - E: Engineering Standards and Procedures 147

The following is a list of the changes that have been made to the document. Each time a round of changes occur, they will be updated and reflected here. Each revision will constitute a new list of notes to be included with this document.

Revision Notes - I

- **Page 1:** The graphic was changed to include the XML Reader/Writer in the program overview.
- **Page 8:** The text in the first paragraph was changed to direct the reader to the Input/Output sections of the document to see the XML requirements for the project.
- **Page 8:** The graphic was changed to add the "/" sign to denote the difference between the open and close tags.
- **Page 10:** The text was changed in the first paragraph to fix grammatical errors
- **Page 10:** The a paragraph was added to the end of the section to denote that TCP/IP is possible though the UNIX shell and can reduce the overhead of the outside programming language on the project.
- **Page 10:** It was noted after the first paragraph in the Client Monitor section that the client will NOT implement TCP/IP, this will be handled though the server
- **Page 12:** The XML format was updated to correct mistakes
- **Page 12:** The Document Type Definition for the XML format was added
- **Page 13:** The XML format was updated to correct mistakes
- **Page 13:** The Document Type Definition for the XML format was added
- **Page 14:** The Document Type Definition for the XML format was added

Revision Notes - II

- **Page 5:** Added the subsection Email Functionality to split the required functions for the Server Module into smaller components based on current implementation of these modules.
- **Page 6:** Added the functions getEmailXML, getContactStructure, and getEmailStructure based on current implementation
- **Page 6:** Added subsections Address Functionality and Mailing List functionality based on implementation and current program structure.
- **Page 9:** The function ignoreWhitespace was added to the XML reader section based on actual implementation of the program.
- **Page 15:** The Section Product Testing was added and included planned properties to include with the project.
- **Page 16:** The subsection Client Modules was added to the product testing section. This is planned requirements and test for the client module once it gets implemented.
- **Page 18-27:** The PSP Report for the initial summary has been updated to include currently implemented object along with the teams combined time and defect logs.
- **Page A1- A20:** The PSP file has been updated with all our current object, time and defect logs used to generate the updated PSP report in the document.

Revision Notes - III

- **Page 1:** Changed the introduction and graphic to our current state of the application.
- **Page 2:** Updated the proof of concept to be inline with the final design and specifications.
- **Page 5:** Added the section Component Requirements in order to provide specific high-level requirements for each module.
- **Page 7:** Added the section Module Design in order to provide full explanation of how a module is to be implemented, tested, and executed. This will allow a developer to implement the requirements spelled out in the Component Requirements section.
- **Page 19:** Updated the Email XML to current implementation standards.
- **Page 22:** Updated PSP report to include all reports submitted by our team to date.
- **Page 43:** Added the full, complete source code files to the appendix.

Introduction

The proposed application is that of an Email Client/Server transaction system that utilizes ACL2 in order to theoretically prove the correctness of the data transformations that occur on both the client and the server applications. While not all components of the system can be theoretically proven, we can provide the means to implement predicate based testing on the data acquired and formulate data structures in which we can inductively test, as opposed to the discrete methods currently implemented in the software industry. However, ACL2 is not without its shortcomings and as a result, we have had to implement some of the systems in a secondary language. For this application, we chose to use Java for interoperability. This also promotes a multilevel design that is easy to build upon and upgradable without detriment to the application. The basic layout of the program hierarchy is as follows:

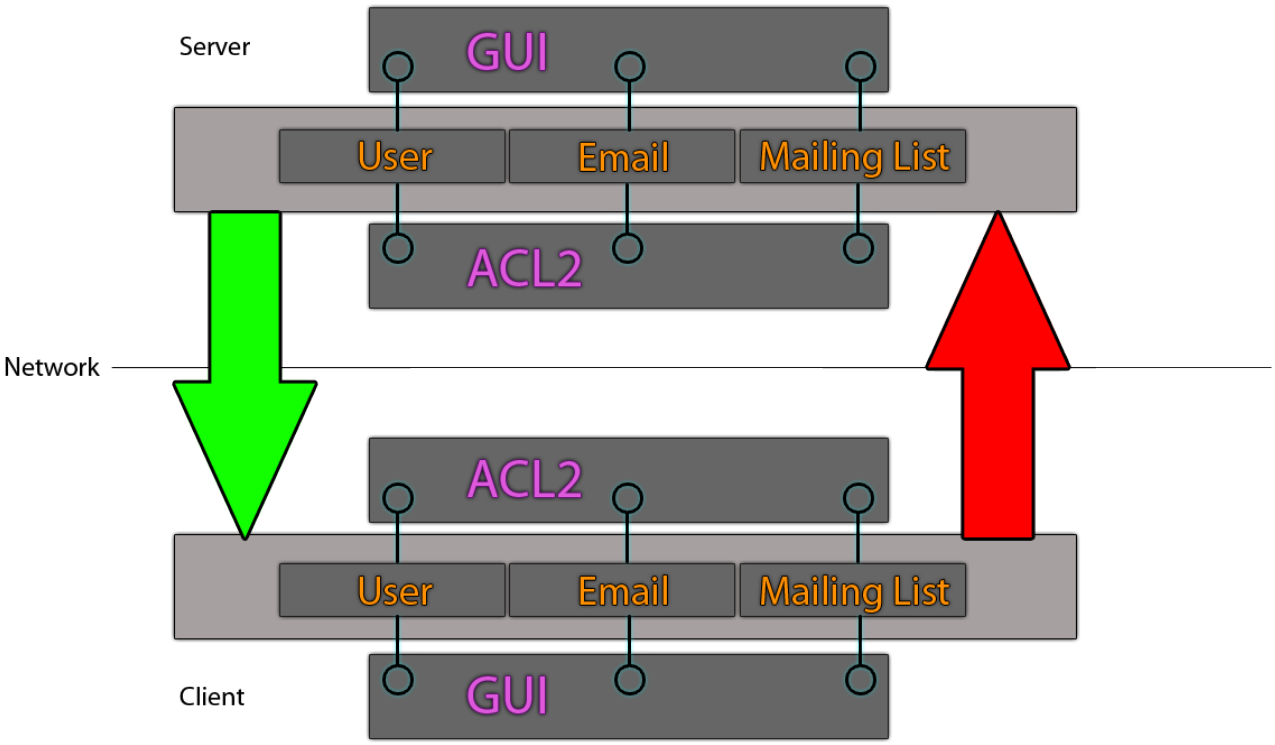


Figure 1: Program Layout

There are two separate applications to this program, which must be running in a synchronized fashion. One is the client, which is the program that invokes the actions upon the program, and the other is the server, which is an automaton after the user starts it on a machine. This allows the client to send information to the server and expect a response based on the type of transaction it invokes. Each program consists of a modules, which in itself, is a standalone application. These modules may be called individually without the GUI intervention. They are, however, dependent upon ACL2 for data transformation. The structure of both applications is the same, with a GUI layer for client interaction, an ACL2 layer for data processing and a module layer for integrating the two and providing a means for invocation.

Proof of Concept

In order for this project to be feasible, we had to research and test our environments to see if the idea of an email client and server system would be possible. This section will outline our research and tests of the ACL2 and operating system environments.

ACL2 Environment Research

Our basis for this project was to develop an application that would implement a network connected application that would have a host program and client programs. Networking is not inherently implemented in ACL2. In order to do this, development of ACL2 modules in Common Lisp would be required. We deemed that this was going to be too much of an undertaking and out of the scope of the Software Engineering - II requirements for the project.

In our investigation, we found that the best way to have ACL2 execute from an outside environment is to call the ACL2 system directly from a Java program. Java provides two things that are critical to this application. First, it provides a means of executing ACL2 in a way that eliminates the need for ACL2 executables. Second, it allows us to use local networking protocols to send our message information over a data connection. A Sample of the the Java code that executes this environment follows.

```
String script = "(in-package \"ACL2\")(include-book \"modules/email/action/rw-
email\"" +" :uncertified-okp t) (readEmail \"incoming/email/"+f.getName()+"\"
\""+unique+"\" state)";

try{
//Run on ACL2
// Initialize ACL2 and dump its output to the log
System.out.println("Executing ACL2 runtime for Email Generation...");
ProcessBuilder processBuilder = new ProcessBuilder("acl2");
File log = new File("logs/acl2_log.txt");
processBuilder.redirectErrorStream(true);
processBuilder.redirectOutput(log);

Process process;
process = processBuilder.start();

PrintWriter procIn = new PrintWriter(process.getOutputStream());

// Write the ACL2 to the process, close ACL2
procIn.println(script);
procIn.println("(good-bye)");
procIn.flush();
procIn.close();
} catch(IOException e) {
e.printStackTrace();
}
```

Once this process is complete, we can use the built in network modules in Java and have the system send data across a port that is registered with the server.

Data Structure Design

This project will handle numerous types of data in order to transmit and receive the required information needed for a complete email message. In order to make this process more streamlined and allow effective communications between the different client and server platforms. We have developed the following internal data structures for email messages, address books, contacts, and mailing lists.

Email Message

An email message in the scope of this project will contain the following elements. A "To" field which will contain the recipient(s) of a message, a "From" field to denote the sender of a message, a "Subject" field for the message's subject line, and the message field which will hold the content of a message. The ACL2 format of this structure is outlined below:

$$'('(to_1, to_2, ..., to_n) from \text{“}subject\text{”} \text{“}message\text{”})$$

Where:
$(to_1, to_2, ..., to_n)$: is the recipient email addresses represented as contact structures.

$(from)$: is the senders email address in a contact structure.

"*subject*": is the message's subject represented as a string value.

"*message*": is the message's content represented as a string value.

Contact Structure

Instead of storing email addresses in the format "name@domain". We have decided to use a tokenized version of this format which will allow easy access to the contacts information without having to re-parse the address each time we need to find out details of the address. The format of the email address in ACL2 will be:

$$'(domain\ name)$$

Where:
domain: is the address domain content. This information is the information after the "@" symbol in the address.

name: is the users email name on the domain. This information is written before the "@" symbol.

Address Registration Structure

In order for a client to send email to another client, it must first register with the server. Server registration will also add a abstracted view of security to the program as well. The registration will take place each time the client is initiated. When initiated, the client will send the server its domain name, user name, and password. This will allow the server to check the registration log and address book to see if the client has registered in the past. If it hasn't, it will add the registration and allow the client to send email messages.

The data structure to be used for this process in ACL2 is:

$$'(\text{"}domain\text{"}\ \text{"}name\text{"}\ \text{"}password)$$

Where:
domain: is the address domain content. This information is the information after the "@" symbol in the address.

name: is the user's email name on the domain. This information is written before the "@" symbol.

password: is the user's password used to verify the user with the server and address book and registration log.

Address Book Structure

The address book will allow a user to search through to view and send email messages to stored contacts. Also this will be used by the server to see if a email address has been registered and is available to receive email messages. The address book will be constructed as:

$$'((\text{“}AddressString\text{”})_1, (\text{“}AddressString\text{”})_2, ..., (\text{“}AddressString\text{”})_n)$$

Where:
"*AddressString*": is the string value of a registered email client.

Mailing Lists Structure

A mailing list be used by the clients of the server. This will allow messages to be sent to multiple clients using one address. This will add convenience to the client when sending email messages. The structure of a mailing list will be:

$$'(\text{“}MailingListName\text{”}\ '(contact_1, contact_2, ..., contact_n))$$

Where:
"*MailingListName*": is the string value of the mailing list's name.

$contact_n$: is a contact in the mailing list. This will be stored as a contact structure as defined above.

Component Requirements

The program is split into two separate programs. The server program will handled the communications between the clients. The client program will handle all user end features and input. Each program is built in a modular fashion and the detailed structure of each module will be discussed in the next section.

Server

The server will need the following modules for implementation. The requirements for each module are:

Email

This is one of the most important modules as it will handle the message interaction between the client and server. The server will need to implement the following in order to successfully handle email messages.

- Open an email message from a file in XML format and save the message in a client's inbox in XML format.
- Split an incoming message into it individual components.
- Read email addresses and generate store directories for saving XML files.
- Parse an incoming email into XML tokens
- Monitor port 20005 for incoming messages. Then save incoming messages into the incoming email directory

User

The User module will need to accomplish two task. The first task will be to register new users into the system. Once a user has been registered, the client will have an inbox available on the server. The next task will be to verify a user and send the user its email messages. The specific task for this module are:

- Create and maintain an email address book of active users.
- Take a user request and generate an email address entry into the address book.
- Assign a password to a user.
- Check to see if a user is already entered into the address book.
- Verify a users password
- Verify a user request as belonging to the user
- Upon verification send all email messages on server to the user.
- Monitor the network on port 20003 for user registration request
- Monitor the network on port 20002 for verification request

Mailing List

The mailing list will be a feature that will allow a user to send a message to multiple recipients by sending an email to an alias address that is registered on the server. Users must have permission to send messages to a mailing list and must be able to request access to either send and receive messages from a mailing list. Specifically the mailing list should:

- Allow a user to request a mailing list be created.
- Allow a user to request access to the mailing list
- Allow a user to request to be removed from the list.
- Recognize the email alias as a mailing list address.
- Upon list recognition, copy the email and send to users on the list.

Client

The client program will be the point of interaction for the users of the system. The client will need these modules in order to interact with the server.

Email

The client email module will need to be able to read and write messages to or from the server. The functionality needed for the email module is:

- Allow a user to input an email message to send to another client.
- Generate the correct XML format from the inputted email message.
- Allow a user to send a single message to multiple recipients.
- Split a multiple recipient email message into separate XML documents for each message.
- Process an incoming email in XML format and convert the email to HTML for the client's inbox.
- Connect to the server over port 20005 and send an email XML file to the server for processing
- Connect to the server over port 20002 and retrieve email messages waiting for the client on the server.

User

The user module on the client will be used to register the client on the server. The User module will need to:

- accept as input the username, domain, and password of a client.
- Generate a request XML file with the given information for the request.
- Connect to the server on port 20003 to send the request file.

Mailing List

The mailing list will allow a client to send messages to multiple clients with one alias. The mailing list will need to:

- Accept as input, the name and recipients for the mailing list.
- Generate a new mailing list request XML file to send to the server.
- Generate a request file to be added to a mailing list.
- Generate a request file to be removed from a mailing list.
- Generate a request to obtain access to a mailing list.

Module Design

There are two separate applications to this program. Both must be running in some synchronized fashion. The first being the client, which is the program that will invoke actions on the user end of the application and make request to the server. Then the server, which is an automaton, will process request and send appropriate information back to the clients. Each program will consist of modules specific to the operation of the program. This section will discuss the exact structure for modules on both the client and server since each application has subtle differences in its development.

Server Design

This section will cover the design standards of the server program and general module structure. The server is divided into three parts: An ACL2 module, the actions the module performs, and a GUI interface for easy startup and shutdown processes.

ACL2

The structure of the ACL2 that is implemented on the program consists of 3 separate parts:
1. The Logic
2. The Tests
3. The Actions

The Logic

The logic consists of the transformations that will occur upon the proposed data structures in the application. An example of this would be the address-book.lisp file.

```
; (addAddress addressBook address)
; Appends an address onto the end of the address book structure.  If the
; user is already in the address book, then the return is the original
; address book.
; addressBook - the address book in which to add the address.
; address     - the address to add to the address book.

(defun addAddress (addressBook address)
  (if (equal (isInAddressBook addressBook address) nil)
      (append addressBook (list address))
      addressBook))
```

In this code snippet, we examine the addressBook, which is a user defined structure a list of addresses, address containing a domain, name and password. It uses a user defined function in order to determine whether the address is in the addressBook structure and if that is false, then we can append the address to the list of addresses. If not, then it is already in the addressBook, thus we can return the original addressBook.

There is not technically a type enforcement on this data, as it can actually hold numerical values. We can safely assume that we are not concerned with the type value until actionable information is passed from the invocation portion of the script since we can explicitly cast this information, but safely assume our testing can include any such value this logic can be

tested again including integers, null values, strings, and even lists of strings. It suffices to say, at the logic level, we are not considerably concerned with the data the user has entered, rather whether the transformations are correct and true. This absolves the developer from considering this issue and allows them to focus explicitly on the logic.

The logic should be included in the modules root folder and a name given (such as users in the case of the example: *modules/users/address-book.lisp*). We will go into actions a bit later, but it is safe to say that anything that invokes this file should be a part of this module.

The Test

Testing is done via Racket utilizing the Dracula environment. There are three alternative methods of testing two of which are discrete, and one through induction. Proof through induction is the most solid means of verifying the transformations from the logic are true and correct. The other two include property based randomized testing (PBRT), and check-expects.

Proof through induction must be implemented through the ACL2 engine. You can use Racket in order to prove the formulated hypothesis, thus accepting it as a theorem into the logical world.

```
; Theorem:
; If the address is not in the address book, then adding the address
; to the address book will return an address book with the length of
; the original address book + 1.

(defthm address-is-not-in-book-add-length-thm
  (implies (and (listp addressBook)
                (not (isInAddressBook addressBook address)))
           (equal (+ (length addressBook) 1)
                  (length (addAddress addressBook address))))
  :rule-classes (:rewrite :forward-chaining))
```

In this example, we are verifying that if we add an address to the addressBook structure that is currently not a part of the addressBook, we can conclude that the size of the addressBook is increased by one. Using the theorem prover yields us our result:

```
ACL2 Observation in ( DEFTHM ADDRESS-IS-NOT-IN-BOOK-ADD-LENGTH-THM...):  The :TRIGGER-TERMS for the :FORWARD-CHAINING
rule
ADDRESS-IS-NOT-IN-BOOK-ADD-LENGTH-THM will consist of the list containing (LISTP ADDRESSBOOK).
ACL2 Warning [Non-rec] in ( DEFTHM ADDRESS-IS-NOT-IN-BOOK-ADD-LENGTH-THM...):  A :REWRITE rule generated from
ADDRESS-IS-NOT-IN-BOOK-ADD-LENGTH-THM will be triggered only by terms containing the non-recursive
 function symbol LENGTH.  Unless this function is disabled, this rule is unlikely ever to be used.
ACL2 Warning [Free] in ( DEFTHM ADDRESS-IS-NOT-IN-BOOK-ADD-LENGTH-THM...):  A :REWRITE rule generated
 from ADDRESS-IS-NOT-IN-BOOK-ADD-LENGTH-THM contains the free variable ADDRESS.  This variable will be
  chosen by searching for an instance of (NOT (ISINADDRESSBOOK ADDRESSBOOK ADDRESS)) in the context of
  the term being rewritten.  This is generally a severe restriction on the applicability of a :REWRITE
   rule.  See :DOC free-variables.
ACL2 Warning [Subsume] in ( DEFTHM ADDRESS-IS-NOT-IN-BOOK-ADD-LENGTH-THM...):  The previously added
 rule COMMUTATIVITY-OF-+ subsumes a newly proposed :REWRITE rule generated from
  ADDRESS-IS-NOT-IN-BOOK-ADD-LENGTH-THM, in the sense that the old rule rewrites a more general
   target.  Because the new rule will be tried first, it may nonetheless find application.
ACL2 Warning [Free] in ( DEFTHM ADDRESS-IS-NOT-IN-BOOK-ADD-LENGTH-THM...):
When the :FORWARD-CHAINING rule generated from
```

```
ADDRESS-IS-NOT-IN-BOOK-ADD-LENGTH-THM is triggered by (LISTP ADDRESSBOOK) it contains the free
 variable ADDRESS.  This variable will be chosen by searching for an instance of (NOT (ISINADDRESSBOOK
  ADDRESSBOOK ADDRESS)) among the hypotheses of the conjecture being rewritten.  This is generally a
   severe restriction on the applicability of the forward chaining rule.
ACL2 Warning [Non-rec] in ( DEFTHM ADDRESS-IS-NOT-IN-BOOK-ADD-LENGTH-THM...):  The term (LISTP
 ADDRESSBOOK) contains the non-recursive function symbol LISTP.  Unless this function is disabled,
  (LISTP ADDRESSBOOK) is unlikely ever to occur as a trigger for ADDRESS-IS-NOT-IN-BOOK-ADD-LENGTH-THM.
<< Starting proof tree logging >>
Goal'
([ A key checkpoint:
Goal'
(IMPLIES (AND (CONSP ADDRESSBOOK)
              (NOT (ISINADDRESSBOOK ADDRESSBOOK ADDRESS)))
         (EQUAL (+ 1 (LEN ADDRESSBOOK))
                (LEN (APPEND ADDRESSBOOK (LIST ADDRESS)))))
*1 (Goal') is pushed for proof by induction.
])
Perhaps we can prove *1 by induction.  Three induction schemes are suggested by this conjecture.
 Subsumption reduces that number to two. These merge into one derived induction scheme.  We will
  induct according to a scheme suggested by (LEN ADDRESSBOOK), but modified to accommodate
   (ISINADDRESSBOOK ADDRESSBOOK ADDRESS).
These suggestions were produced using the :induction rules BINARY-APPEND, ISINADDRESSBOOK and LEN.  If
 we let (:P ADDRESS ADDRESSBOOK) denote *1 above then the induction scheme we'll use is
(AND (IMPLIES (NOT (CONSP ADDRESSBOOK))
              (:P ADDRESS ADDRESSBOOK))
     (IMPLIES (AND (CONSP ADDRESSBOOK)
                   (:P ADDRESS (CDR ADDRESSBOOK)))
              (:P ADDRESS ADDRESSBOOK))).
This induction is justified by the same argument used to admit LEN.  When applied to the goal at hand
 the above induction scheme produces three nontautological subgoals.
Subgoal *1/3
Subgoal *1/2
Subgoal *1/1
Subgoal *1/1.2
Subgoal *1/1.1
*1 is COMPLETED!
Thus key checkpoint Goal' is COMPLETED!
Q.E.D.
Summary
Form:  ( DEFTHM ADDRESS-IS-NOT-IN-BOOK-ADD-LENGTH-THM ...)
Rules: ((:DEFINITION ADDADDRESS)
        (:DEFINITION BINARY-APPEND)
        (:DEFINITION ISINADDRESSBOOK)
        (:DEFINITION LEN)
        (:DEFINITION LENGTH)
        (:DEFINITION LISTP)
        (:DEFINITION NOT)
        (:EXECUTABLE-COUNTERPART BINARY-+)
        (:EXECUTABLE-COUNTERPART CONSP)
        (:EXECUTABLE-COUNTERPART EQUAL)
        (:EXECUTABLE-COUNTERPART LEN)
        (:EXECUTABLE-COUNTERPART LENGTH)
        (:FAKE-RUNE-FOR-TYPE-SET NIL)
        (:INDUCTION BINARY-APPEND)
        (:INDUCTION ISINADDRESSBOOK)
        (:INDUCTION LEN)
        (:REWRITE CDR-CONS)
        (:REWRITE COMMUTATIVITY-OF-+)
        (:TYPE-PRESCRIPTION BINARY-APPEND)
        (:TYPE-PRESCRIPTION ISINADDRESSBOOK)
        (:TYPE-PRESCRIPTION TRUE-LISTP-APPEND))
Warnings:  Subsume, Free and Non-rec
Time:  0.00 seconds (prove: 0.00, print: 0.00, proof tree: 0.00, other: 0.00)
Prover steps counted:  1017
 ADDRESS-IS-NOT-IN-BOOK-ADD-LENGTH-THM
```

As we can see, the theorem has been accepted and the inductions require are listed above.

It also goes into a full textual description of the proving steps and lists the number of steps required to prove the function. Thus we can assume that the logic for adding an address to the addressBook is theoretically sound.

PBRT describes a hypothesis of the transformation of a function to be tested. Random values are assigned, and injected into this logic and tested for its trueness. The range of testing can be user specified, but for our sake, we have implemented a default value of 50 cycles. The formulation of the proof structure is similar to that of the theorem test, but we are required to define out input values.

```
(defproperty address-is-not-in-book-add-length-tst
  (domain      :where (stringp domain)
               :value (random-string)
   name        :where (stringp name)
               :value (random-string)
   pass        :where (stringp pass)
               :value (random-string)
   domain-not  :where (stringp domain-not)
               :value (random-string)
   name-not    :where (stringp name-not)
               :value (random-string)
   pass-not    :where (stringp pass-not)
               :value (random-string)
   address     :where (listp address)
               :value (list domain name pass)
   address-not :where (listp address-not)
               :value (list domain-not name-not pass-not)
   addressBook :where (listp addressBook)
               :value (list address))
  (implies (not (isInAddressBook addressBook address-not))
           (equal (+ (length addressBook) 1)
                  (length (addAddress addressBook address-not))))
  :rule-classes (:rewrite :forward-chaining))
```

We are using exactly the same hypothesis that we proved in ACL2. The only different is the discrete values that are populated. Though these values are considered to be random, they are limited, this is not a correct bearing for the correctness of the hypothesis, but rather standard foundation for augmenting out hypothesis if we are unable to get it to verify correctly in ACL2. The results of this test are:

We can view each individual trial, thus if there is an error, we can see the values as opposed to trying to change our hypothesis based on theoretical results usually it is more intuitive for developers to see a value that is causing it to fail.

Sometimes developers may find this process frustrating as theory is generally more difficult to prove over that of expected results with known inputs. This is where the check-expect typically helps. With it, we can enter a value that we predetermine and verify it against the expected output (producing a true/false result).

```
(check-expect (length (addAddress '((1 2 3)) '(4 5 6))) 2)
```

This statement states that if we add the address *(4 5 6)* to the addressBook that contains *(1 2 3)* only, we will acquire the addressBook with the additional address, verified by the length of the total address book in this case, addressBook is a list of lists or numbers (three values). Running this test will only result in an echo of "test passed" or "test failed", with no reasoning.

Typically these logic files are included in the modular folder with the logic file (usually suffixed by _tests). No naming conventions are enforced at this level other than knowing that this file is not to be invoked by the ACL2 subsystem when the module is running. It is explicitly used for testing purpose.

The Actions

Logic is for naught if we are not doing anything with it. This is where our actions come into play and the integration into or client/server environment. The actions are the standalone program portion of the module that that will perform the IO as well as any network connectivity. Portions of this are written in ACL2 and others are written in Java. There were some considerations previously of writing the whole module in ACL2, but network connectivity proved to be a challenge. This was later changed to shell scripting in the BASH shell, but synchronization through connectivity, yet again, proved to be an issue. We could send information across a network, but waiting for a response proved to be an issue. In the end, we decided that Java was the best alternative and many students would integrate into this environment seamlessly. This is also where the client and server start to deviate in code production.

The Module

There are typically two parts to the action of each module, but a user can define as many as they need to complete the task. You can implement GUI elements in this application or you may even perform some third party processing to get the information ready for ACL2 processing. For the sake of demonstration, we will be implementing this in a simplistic manner through the user registration process.

Since the action we defined is register, a subfolder in the *user* folder is created and will contain the files responsible for registration. We can define the files as needed, but we are going to invoke against the RegisterUser class. Since the execution of the server occurs from the root, we must include this Java class in the package: "*modules.user.register*" which makes the fully qualified name for invocation *modules.user.register.RegisterUser*. More will be com-

pounded upon this when the GUI wrapper is introduced. We also need a minimum of one lisp file that will generate IO for the ACL2 environment and put the logic to work on data structures. This is also the point to which the user should be concerned with the data types being introduced into the ACL2 environment, since it will be responsible for writing persistent data.

Example of lisp required:

```
; (registerUser regXML abXML state)
; Processes the information that is passed via XML string to add the user
; to the global server address book.
; regXML - The XML that is contained in the registration file - sent
;          dynamically via shell script.
; abXML  - The XML that is contained in the address book file - sent
;          dynamically via shell script.
; state  - The state of the streams in ACL2.

(defun registerUser (regXML abXML state)
       (let* ((tokens (tokenizeXML regXML))
                      (domain (getDomain tokens))
                      (name (getName tokens))
                      (pass (getPassword tokens))
                      (addressBook (getAddressBook (tokenizeXML abXML))))
              (mv-let (error state)
                     (string-list->file
                        "store/address-book/temp_address-book.xml"
                        (getAddressBookXML (addAddress addressBook (list domain name
pass)))
                            state)
                     (if error
                            (mv error state)
                            (mv "Wrote temp_address-book.xml successfully" state)))))
```

This example provides an entry point that will attempt to add the user to the addressBook and store it for persistence. There are a couple of things to note we are writing to *temp_address-book.xml* which generally does not denote permanent storage, but because of one of the shortcomings of ACL2, we cannot overwrite a file that is currently open for read. This will be expanded upon when we discuss the GUI integration.

The XML is tokenized from the incoming request, parsed into the address and addressBook data structures. This information is fed directly from the Java file into this function and pushed into ACL2. The response is written and additional logic in the Java file ensures that the addressBook data structure has changed by verifying the sizes of the *temp_address-book.xml* and *address-book.xml* files. We can then move on to the java file construction. In this file, the networking (if needed) and typical IO functions that cannot be handled by ACL2 generally occur. Since this is a standalone application, it must have a main method associated in the file. From there we can call the *ProcessBuilder*, and invoke ACL2 by building "wrappers" for the ACL2 input and output. You are able to feed ACL2 commands and functions directly into the environment, after which you must call (good-bye) in order to exit the process. General cleanup of memory must occur and the process is repeated for every incoming request.

After the Java file is constructed, you must compile it and register the binary in the server module registration manager (for server side modules). Client side requires integration into

the application at this point, which is direct manipulation of the source code and recompilation.

The GUI

Modules are invoked from a *ProcessBuilder* similar to the one that invokes ACL2. This requires that we register each module so the user can identify it. Since this is allowed on the server side, there is no overhead for integration into the Server UI, rather just standalone application specifics that are introduced when the module has been ran.

To register the module on the server, execute the Server class through a console (java Server). A GUI should show up with a menu at the top of the screen. Under the "modules" menu, you will find "management". Clicking this will open a list of modules currently registered with the server and by clicking the "add" button, you can add the module. It is important to note that you cannot register modules that are listening on the same network port. They will create a confliction and fail to start. The registration will simply reject it completely. The fields are fairly straight forward with the name of the module, location and port. Location can be selected via the locator button, which will generate the expected package name for invocation.

Client Design

The client is developed into modules with three major components: the logic, test, and actions. Each module runs independently from each other, thus making each module a stand alone application itself. The modules are tied together into the GUI in order to make reading the results and executing the application easier. However, the GUI is not needed in order to make the application run. The layout of each layer is described below.

ACL2

The ACL2 components are structured into 3 parts: 1. The Logic 2. The Test 3. The Actions

Logic

The logic consists of transformations that need to be handled by the proposed data structures in the application. An example of this would be the client-email.lisp file.

```
;(email to from sub msg)
;Generates the email XML for email messages based on user string input.
(defun email (to from sub msg)
   (if (consp (cdr (splitToField to)))
     (multRecip (splitToField to) from sub msg)
      (getEmailXML (generateEmailFromStrings (car (splitToField to)) from sub
msg))
   )
)
```

In this code, we take in the contents of an email message that will need to be generated into an XML format. The function takes in four parameters with each being a corresponding component to an email message. Once the function has completed, we are left with the XML contents for an email message that will then need to be stored for output, which will be handled in the actions component. An example of the raw XML output would be:

```
<?xml version='1.0'?><!DOCTYPE user SYSTEM '../../../dtd/messages.dtd'>
<email>
    <to>howell@localHost</to>
    <from>crist@localhost</from>
    <subject>SUB001</subject>
    <content>MSG001</content>
</email>
```

As you can see, each of the four fields is generated into an email message format. This XML format is strictly defined in the document type definition located in the directory specified. This DTD must be included on BOTH the client and server in order for the XML definitions to pass.

Logic files will need to be stored in the root directory of the module and given a name such that it represents the logic it is performing (For the above example, the email module on the client is stored in the following directory: modules/email/client-email.lisp). Any operation that needs to include this main logic file should be part of this module.

Test

Testing is done through the Dr. Racket and Dracula interface. There are three methods of testing which include Theorems, Properties, and Checks. Theorems are proofs through induction of your methods. Properties and Checks utilized specified and random data to test your code for expected outputs.

Proofs through inductions that are accomplished through the theorems are the soundest way of testing your logic code. It is safe to say that if your theorem passes your inductive hypothesis, your function produces correct output according to your hypothesis. You can use Dr. Racket for theorem proving. An example theorem would be:

```
;(genEmailProducesMessage)
;Theorm to test that the email structure
;returned from generateEmailFromStrings is the correct
;email structure ((list) str str str)
(defthm genEmailProducesMessage
  (implies (AND (stringp msg)
                (AND (stringp sub)
                (AND (stringp to)
                (stringp from))))
                (AND (listp
                 (generateEmailFromStrings to from sub msg))
                (listp
                 (car (generateEmailFromStrings to from sub msg))))
           ))
```

This theorem proves that an email message structure is returned from the generateEmailFromStrings function. It test to see if the format of the output follows this format. `((list) str str str)` If it does, then the correct structure of an email message has been processed and returns a true result. Since this theorem passes ALC2 logic, it is safe to say that our logic produces the desired output.

Property based testing and Check Expects are needed to test boundary cases of your logic

functions. These forms of testing are useful if you are experiencing trouble with specified conditions and if random data is needed for your project. Since the email client relies heavily on invariant data, these forms of testing are not idea for this application.

It is also worth noting that through the Racket interface, a property-based test can be passed to the ACL2 logic mechanism. The same logical induction is performed on the properties and its results are the same as if the test were a theorem. So you can use a property as if it were a theorem in this case.

Actions

The actions are where the logic portions of the module are executed. The actions file should contain all the external dependencies of the module. Also the actions should also perform all the IO operations for your logic. This is done to help the module become its own stand alone application. In the client, the actions layer is developed into two programs. A Java program handles the invocation and network connections, and the ACL2 program which handles the ALC2 logic and IO for the module.

We need to expand on the Java program. Each module will need a Java program to be included with the actions of a module. This will take care the network connections, ACL2 invocation, and interfacing with the GUI. The Java program will need to be developed as a library, which deviates from the server. This will allow the module to stand on its own, as well as be expanded on in the future. An example of a header for the Java program follows:

```
package modules.email.action;

import java.io.*;
import java.util.*;
import java.net.*;

public class GetEmail {

      public final static String OUTPATH = "store/email/outbox/";
      public final static String INPATH = "incoming/email";

       public static void getEmail (String name, String domain, String
password){
```

As you can see, this Java program is registered in a package. This package will need to match your file path the location of your Java class. This will allow other applications in the client program to import your Java class and use its functions.

Also this program will contain statically defined functions, which the calling program will be able to use. These functions will use Javas ProcessBuilder to build out a process to ACL2 and run the ACL2 environment. An example of the ACL2 call for this action is on the following page:

```
String script = "(in-package \"ACL2\")(include-book \"modules/email/action/rw-
email\"" +" :uncertified-okp t) (readEmail \"incoming/email/"+f.getName()+"\"
\""+unique+"\" state)";

try{
//Run on ACL2
// Initialize ACL2 and dump its output to the log
System.out.println("Executing ACL2 runtime for Email Generation...");
ProcessBuilder processBuilder = new ProcessBuilder("acl2");
File log = new File("logs/acl2_log.txt");
processBuilder.redirectErrorStream(true);
processBuilder.redirectOutput(log);

Process process;
process = processBuilder.start();

PrintWriter procIn = new PrintWriter(process.getOutputStream());

// Write the ACL2 to the process, close ACL2
procIn.println(script);
procIn.println("(good-bye)");
procIn.flush();
procIn.close();
} catch(IOException e) {
e.printStackTrace();
}
```

This code will create a new process; call the ACL2 environment, and the build a string that is an ACL2 call to your ACL2 action file. The output of the ACL2 environment is then stored in a log folder in your clients programs root directory. This is a way of seeing where the client program failed during debugging.

The Module

There are usually two parts to the to the action of a module. But it is possible to expand a module to more than two actions if it is needed to get a task completed. In order to explain the structure of a module, we will show the example of sending and Email message to the server.

The action to be performed is email; a subfolder in the modules folder is created. This root folder, email, will contain the logic and test for the client-email platform. Within the email directory, another subfolder called actions is created. This folder will contain both the ACL2 actions and the Java actions for the email module.

Since the execution of this module comes from the main root of the client application, we need to register the Java classes associated with sending emails with the following path: "modules.email.action" which makes the fully qualified name for invocation modules.email.action.sendEmail. We also need at least one lisp file in this directory which will contain the ACL2 IO functions needed to generate an email message to be sent.

An example of one of these lisp files that will need to be included in the actions directory:

```
(in-package "ACL2")

(include-book "../../../include/xml-scanner" :uncertified-okp t)
```

```
(include-book "../../../include/io-utilities" :uncertified-okp t)
(include-book "../client-email" :uncertified-okp t)
(set-state-ok t)

(defun writeEmailToFile (xmlStr to ts state)
  (mv-let (error state)
          (string-list->file (concatenate 'string
                                          "store/email/outbox/"
                                          to ts ".xml")
                             xmlStr
                             state)
     (if error
         (mv error state)
         (mv "XML File Written Successfully" state))))


(defun writeMessage (to from sub msg state)
(let* ((msgs (email to from sub msg)))
       (writeEmailToFile msgs to from state))
)
```

This example lisp file will attempt to generate the XML file for the email message the user has inputted into the function *writeMessage* and store it into the store/email/outbox directory on the client.

The input though, is first sent to the *email* function in the client logic to generate the XML, if the user had addressed the email to multiple people, then the logic will handle the delegation of generating a list of multiple XML outputs. Once the logic has completed the function then saves the list of XML files in the outbox directory.

Once ALC2 has finished and saved the XML file in the Outbox, the Java process continues. It will open the file in the outbox and check the file to see if one or more XML messages are contained in the file. If more than one message is detected by using a regular expression, Java will split the file into multiple files such that only one message is included with each file and holding with the DTD definition.

Once the files are split, Java will send the messages to the server using a Java socket command. This is outlined in the Java program and is simple to implement. Most of the communications will occur on the following ports: 20002 for user verification, 20003 for user registration and 20005 for sending emails. Since we are sending an email, we will use port 20005 on the server to send the message.

This gives a basic overview of how the actions are performed in the client program. The other modules follow a similar pattern as sendEmail except each action has its own port as noted earlier. Once you complete a Java file, make sure you compile the *.class* file in the same directory as the Java source code file. This will allow the action to be reachable from

the root of the application and be used in other applications such as a GUI interface.

The GUI

Modules are invoked by a static method inside the modules Java library. This requires you to import the library in the Java program you are using. To import your Java library, you should use the import command using the path to your source code. This path should match the package name of your Java class. An example import would be:

```
import modules.email.action.*;
```

This will allow you to call the static action functions in the library and use the ACL2 modules from a Java program. For the *sendEmail* example, you would call SendEmail.sendEmail("To", "From", "Subject", "Message"). This will call the library and invoke the send email process in ACL2.

By producing the Java programs in this fashion, we allow modularity of the program to remain independent of the GUI and other invocations. This has also allowed us to develop a stand-alone client GUI interface for the client. Once this is compiled on your machine. You will need to access the client directory from your command terminal. Once here you can issue the command:

```
java ACL2Email
```

This will launch the Email client GUI interface. The current modules already imported into the interface are:

- Send Email
- Verify User and Get Email
- User Registration

Once more modules are developed; they can be easily implemented into this interface. With this structure, you are also able to make changes to the action files without having to worry about changing the GUI. The only action the GUI should make is to the static function in the actions library file.

Input/Output

Email Message I/O

The input and output for this project will be handled using XML data formatting. Each client connected to the server will be able to send email messages to the server. Messages sent to each client from the server should also use the same XML format for the message. Keeping this format for both outgoing and incoming messages will allow the client and server to use the same set of functions for reading and generating data.

A sample of an XML email message output is listed below:

```
<?xml version="1.0"?>
<!DOCTYPE messages SYSTEM "email.dtd">
   <email>
        <to>wesley.howell@localhost</to>
        <from>matthew.crist@localhost</from>
        <subject>Message1 subject</subject>
        <content>Test For a Message Would be Here</message>
   </email>
```

The document type definition for this XML format is below:

```
<!ELEMENT email   (to, from, subject, content)>
<!ELEMENT from    (#PCDATA)>
<!ELEMENT to      (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT content (#PCDATA)>
```

Email Address Book I/O

The email address scheme for this project will be handled using the standard email format with the following modification. A sample email address for this project is:

```
name@domain
```

The name portion of the email is the standard user name of the client on the domain. However, on the domain name, we have decided to drop the `.com, .net, .org, etc.` extensions since they will not be needed for this program's initial functionality.

Email addresses are to be stored in an address book file which will have the following format.

```
<?xml version="1.0"?>
<!DOCTYPE addresses SYSTEM "address.dtd">
<addresses>
   <address>
        <domain>localhost</domain>
        <name>Matthew.Crist</name>
```

```
   </address>
   <address>
        <domain>localhost</domain>
        <name>Wesley.Howell</name>
   </address>
   <address>
        <domain>localhost</domain>
        <name>Adam.Ghodratnama</name>
   </address>
</addresses>
```

The document type definitions for the addresses XML are as follow:

```
<!ELEMENT address* (name, domain)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT domain (#PCDATA)>
```

Mailing List I/O

Mailing list will also be stored to a file on the local machine/ server machine. The mailing list will allow one client to send a message to multiple clients. Also, a mailing list will need to be identified by its owner along with the email contacts to be stored on the list.

The structure of a mailing list file will be in the following XML format:

```
<?xml version="1.0"?>
<!DOCTYPE mailing_list SYSTEM "mailing_list.dtd">
<mailing_list>
   <list_name>My mailing list</list_name>
   <addresses>
      <address>
         <domain>localhost</domain>
         <name>matthew.crist</name>
      </address>
      <address>
         <domain>localhost</domain>
         <name>wesley.howell</name>
      </address>
      <address>
         <domain>localhost</domain>
         <name>adam.ghodratnama</name>
      </address>
   </addresses>
   <owners>
      <owner>
         <name>matthew.crist</name>
         <password>my_password</password>
      </owner>
   </owners>
</mailing_list>
```

The document type definitions for the Mailing List are as follow:

```
<!ELEMENT addresses (address+)>
<!ELEMENT owners (owner+)>
<!ELEMENT list_name (#PCDATA)>
<!ELEMENT address (name, domain)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT domain (#PCDATA)>
<!ELEMENT owner (name, password)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT password (#PCDATA)>
```

PROBE Estimates

During the initial design phase of this project, we created estimates based on our initial designs. The new objects section of the PSP++ report contain the information of these modules and the projected, detailed estimate for each object.

The estimations provided were calculated using our team's Lines of Code table from the compilation of all the projects from the fall semester.

The table is as follows:

Function Type	Tiny 7%	Small 24%	Medium 38%	Large 24%	Huge 7%
Non IO Functions	3	4	6	9	18
IO Functions	4	5	9	14	16
Properties	3	4	6	8	14
Check-Expects	2	3	4	8	14

Using this table, we were able to assign an estimated size of each function defined in this project. Each function is listed below along with the estimated size and Lines of Code for each particular function.

Totaling all these functions, we are able to predict that this project will be around **1080** lines of new code.
We also need to note that the aim for this project is 2,500 - 3,500 lines of code. We can accept the 1080 line estimate because of the margin of error with the PROBE method. Since most of the projects that our team has completed mostly contained smaller scale functions, we can assume that the PROBE estimate can be off by a **factor of 3**.

The PSP file that contains the objects used for the PROBE estimate are located in **Appendix A**. The summary of the new objects to be created are included with the following PSP++ report:

Email with ACL2

Personal Software Process Summary

Project Essentials

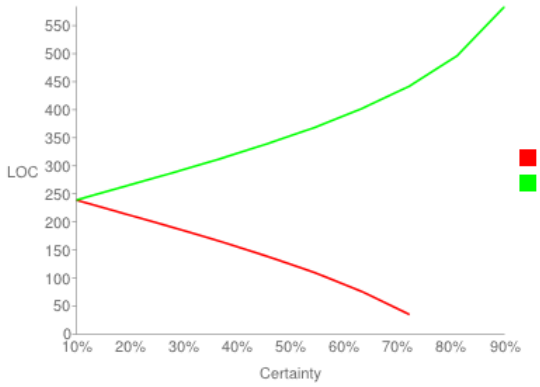
Name: Team Dijkstra
Instructor: Dr. Rex Page
Date: April 16, 2013
Language: ACL2, C++, Shell Scripting, XML

Lines of Code

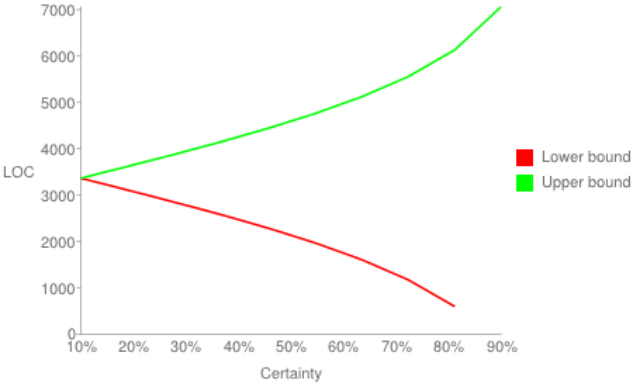
Type	Prediction by user	Actual
Added	1080	401
Base	0	66
Modified	0	15
Removed	0	0

PSP Projection

LoC Certainty



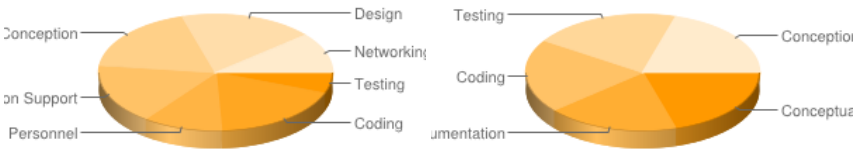
Time Certainty



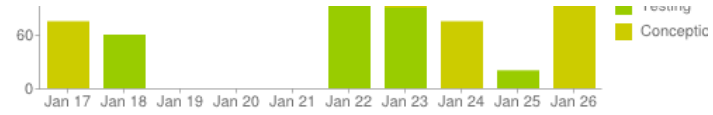
Project Data

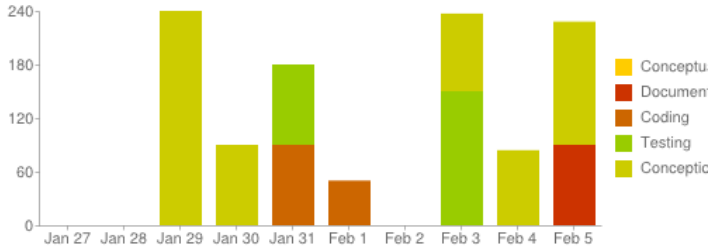
Time Per Defect Type

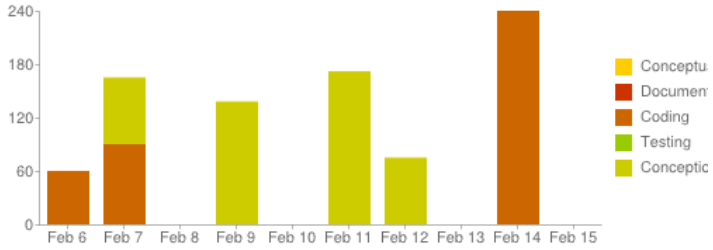
Time Per Phase

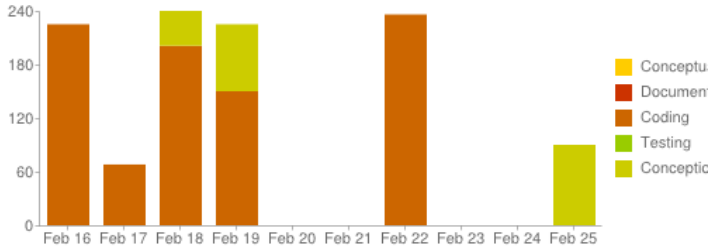


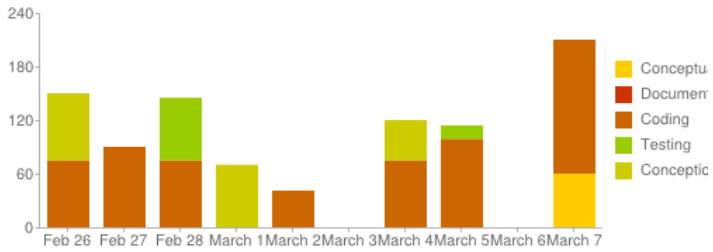
Time by Day

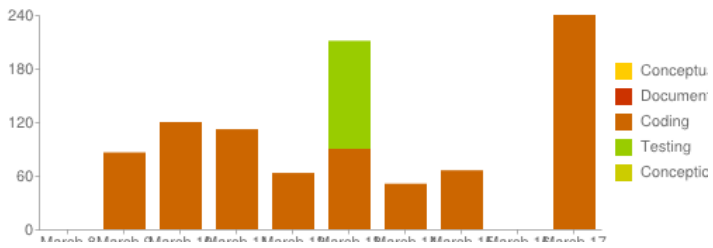

60
0
Jan 17 Jan 18 Jan 19 Jan 20 Jan 21 Jan 22 Jan 23 Jan 24 Jan 25 Jan 26
Conceptic

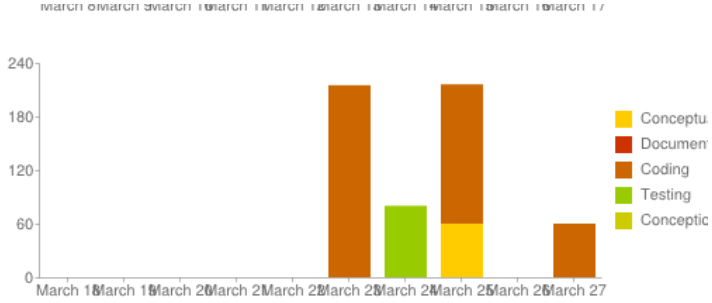

240
180
120
60
0
Jan 27 Jan 28 Jan 29 Jan 30 Jan 31 Feb 1 Feb 2 Feb 3 Feb 4 Feb 5
Conceptu
Documen
Coding
Testing
Conceptic

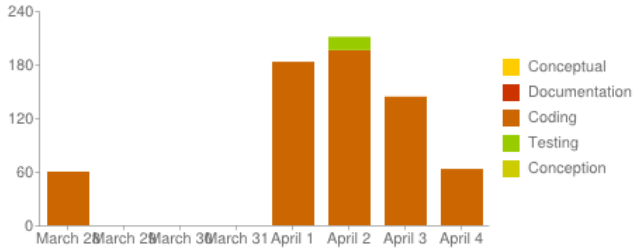

240
180
120
60
0
Feb 6 Feb 7 Feb 8 Feb 9 Feb 10 Feb 11 Feb 12 Feb 13 Feb 14 Feb 15
Conceptu
Documen
Coding
Testing
Conceptic


240
180
120
60
0
Feb 16 Feb 17 Feb 18 Feb 19 Feb 20 Feb 21 Feb 22 Feb 23 Feb 24 Feb 25
Conceptu
Documen
Coding
Testing
Conceptic


240
180
120
60
0
Feb 26 Feb 27 Feb 28 March 1 March 2 March 3 March 4 March 5 March 6 March 7
Conceptu
Documen
Coding
Testing
Conceptic

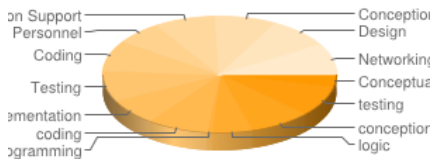

240
180
120
60
0
Conceptu
Documen
Coding
Testing
Conceptic



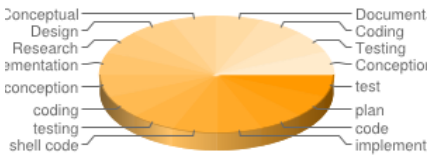


Cumulative Data

Time Per Defect Type



Time Per Phase

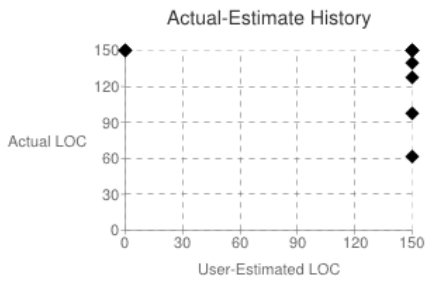


Actual vs Estimated LoC

Actual vs Estimated LoC

Project	Estimate	Actual
Email Client Server System	0	416
Email Client Server System	0	171
Email Client-Server System	1080	274
tImpl	307	288
tCodeRvw	258	41
iEx5	198	189

4/14/13 PSP Report



iEx4	111	65
iEx3	173	93
iEx0	120	85

Time Log

Date	Type	Int. Time	Description
Jan 17, 2013, 10:30 AM - 11:45 AM	Conception	0	Team Dijkstra's regular meeting time. We brainstormed project ideas and decided on a networked chat client and server system.
Jan 18, 2013, 1:00 AM - 2:00 AM	Testing	0	Tested network connectvity for ACL2 using various methods. Found that through file streaming, we could transfer files through connected network drives. We are not able to connect to a specific TCP/IP address or anything without a directory resolution on the network. Example: \\MatthewCrist-Laptop would work, \\127.0.0.1 would work, but http://127.0.0.1 or \\127.0.0.1:80 would not yeild results. I have determined that either we will need supplemental language in order to make the transfer or map network drives.
Jan 22, 2013, 10:30 AM - 11:45 AM	Conception	0	Team Dijkstra's regular meeting time. Finalized the idea of the chat system and began initial designs for the proposal.
Jan 22, 2013, 7:32 PM - 8:44 PM	Conception	0	Worked on updating the team's LoC table to include tImpl from last semester. Reused the spreadsheet from last semester and added the new data. Then included the information into the t2 and t3 documents.
Jan 22, 2013, 8:53 PM - 10:49 PM	Testing	0	Worked on researching networking from within ACL2. There is not any native support for networking. However, we did discover that we could use networking if the Operating System supports it. Since our operating systems can be networked, we can work around this limitation.
Jan 23, 2013, 2:30 AM - 4:00 AM	Testing	0	Attempted to research means by using Common Lisp to implement a layer for TCP/IP transfer to see if this is a viable alternative. The solutions that are available seem to be beyond the scope to which we can implement. It appears the network drive route is our best alternative.
Jan 23, 2013, 8:04 PM - 10:27 PM	Conception	0	Worked on writing the content of the Proposal. Finished the sections for the Overview, High-level design, some of the requirements, and began the PROBE estimate.
Jan 24, 2013, 10:30 AM - 11:45 AM	Conception	0	Team Dijkstra's regular meeting time. We worked on the proposal and worked on several proofs of concepts to determine if the project is feasible.
			Researched event handling in ACL2 from previous project regarding "Blue Ball" scenario. Determined that this could be used to implement the client side

Jan 25, 2013, 10:00 AM - 10:20 AM	Testing	0	gestures for the chat messages to be sent and received. Also noted that the primitive nature of the interface would require that we construct many of the GUI element ourselves. I think we should be able to assign functions to these events to read and write files where necessary.
Jan 26, 2013, 4:04 PM - 6:31 PM	Conception	0	Finished the complete proposal with all the specified requirements and PROBE estimate.
Jan 29, 2013, 10:30 AM - 11:45 AM	Conception	0	Team Dijkstra's regular meeting time. Dr. Page addressed concerns about the Project and that GUI's and File IO cannot exist in the same ACL2 program. We began a major re-design of the project and re-wrote most of the project requirements that are to be included in the proposal.
Jan 29, 2013, 11:45 AM - 2:12 PM	Conception	0	Found out that some of the components of the original proposal were not feasible. Worked with Matthew to re-write sections of the proposal to include the new ideas for the project and write out the requirements for the project. The new project will be an email system instead of a chat system, with more emphasis on message delivery and content.
Jan 29, 2013, 12:00 PM - 2:30 PM	Conception	0	After discussion with Rex Page regarding some of the features of ACL2, he described to us that ACL2 cannot use a GUI and write/read from files at the same time. This has thrown us in a tizzy about writing a chat client. After reassessment of the situation, we determined that we could salvage much of the information that we derived in previous sessions by using supplemental language, such as C to write the server and client interfaces that would invoke ACL2 executables on demand and have these applications handle the events for both client and server processes. We could use folder monitoring on server side and invoke those modules that are created adding to the modularity of the application itself. Isaac, Wes and I rewrote the design document so that it would be available for Thursday (January 31, 2013) for the presentation. We also determined that Adam should be the one to give the overview of the application since he may want the extra speaking opportunity.
Jan 30, 2013, 4:00 AM - 5:30 AM	Conception	0	Created the slides for the presentation that will be held tomorrow (January 31, 2013). Used the design document as a point of reference for bullet points. Also devided the slide information based on role (as opposed to design layer) to "sell" the idea to the end user and convey the purpose of the process we are intending to use.
Jan 31, 2013, 8:00 AM - 9:30 AM	Testing	0	Attempted to get Proofpad to function on my windows PC (Microsoft Windows 8) to no success. Attempted to get it to work on my Linux partition as well (Ubuntu 11.04 LTS) with no such luck either. Both result in NullPointerExceptions being tossed back by the application itself. Unable to invoke ACL2 internally as a result. Decided to do some testing with ACL2 command line and get familiar with the environment Noticed that the teachpacks were not certified for io-utilities.lisp and list-utilities.lisp, so I had to include them in a subfolder and include them in the file directly. Cannot use program mode in ACL2 and have to stay in logic mode else the files will not include correctly into the project.

			Considering DrRacket as the IDE for development as a result.
Jan 31, 2013, 1:00 PM - 2:30 PM	Coding	0	Started to develop the infrastructure for the server monitor program. Determined that the use of processes and a dependency relationship was necessary to ensure that a process was complete before invoking a dependent processes. Derived an XML DTD for the input format to load modules into the server environment for invocation. Assigned names to modules in the format (program).(content).(action) naming convention, a folder to monitor for files, and a process to invoke when a file is detected. Information has been updated on the wiki for reference on module creation under >> The Server Monitor.
Feb 1, 2013, 4:30 AM - 5:20 AM	Coding	0	Did some more work on the Server Monitor. Wrote threading for directory monitoring and process invocation. Have not tied module invocation directly to program yet, as I need to verify that executables can be effectively created from Racket. File size seems to be rather large at the moment for just a hand full of code. May consider an alternative approach.
Feb 3, 2013, 12:42 AM - 2:09 AM	Conception	0	Matthew emailed about an issue with the ACL2 EXE's not working correctly and possibly being unable once the project gets large. Spent this time researching how to move the project to UNIX based servers and working with launching ACL2 from the terminal and using shell scripts to call ACL2 lisp files.
Feb 3, 2013, 10:00 AM - 12:30 PM	Testing	0	Determined that executables in ACL2 is not a practical solution. Tried invocations through redirection on the input method for ACL2 since the executable did not take any arguments. So far I have had success. Module invocation can occur through a shell script which does not require dependency checks (which will allow for linear execution.) Reassessing the use of unsynchronized process invocation. Perhaps synchronizing the main processing thread to halt while a process is running could be the best alternative. Invocation of a shell script also alleviates the overhead that may have been involved and we can implement some of the operating system features to transfer information (such as ftp for file transfer).
Feb 4, 2013, 5:23 PM - 6:47 PM	Conception	0	Worked on typing the progress report with the completed task to date and writing out the plans for the task to be completed. Also noted in in the progress report the need to move the system server to a UNIX based system to run the server components
Feb 5, 2013, 4:00 AM - 5:30 AM	Documentation	0	Updated progress report with findings over the last two weeks for delivery to Dr. Page and the team. Need to address issues with the change in the group make up, since Isaac will be unable to participate in the project until his personal issues are resolved.
Feb 5, 2013, 10:30 PM - 11:45 PM	Conception	0	Team Dijkstra's regular meeting time. We delivered a progress report to Dr. Page. We worked on converting the Windows based designs to a UNIX file system in order for the system to run.
Feb 5, 2013, 7:32 PM - 8:35	Conception	0	Worked on setting up ACL2 modules to run though the UNIX shell and wrote test scripts to

PM			automatically call ACL2 functions from a shell script
Feb 6, 2013, 10:00 AM - 11:00 AM	Coding	0	Reorganized server code into better directory management so ensure a structure that can be deployed more effectively. Updated references to be on local scope, as opposed to global folder references for portability purpose.
Feb 7, 2013, 6:30 AM - 8:00 AM	Coding	0	Wrote scanner portion for the analyzer for XML content that can be parsed with ACL2. Added consume, tag, pcData, nextToken, and tokenizeXML functions. Entry point is tokenizeXML which takes a string of XML and will return a list of tokens and the token type. This should then be added to a stack to verify xml correctness and can then be used to extract the PCDATA information contained between the brackets.
Feb 7, 2013, 10:30 PM - 11:45 PM	Conception	0	Team Dijkstra's regular meeting time. We worked on the XML format and multi-level design and designed the data structure format for the project. We then wrote sections of the progress report and assigned an XML module to each team member.
Feb 9, 2013, 5:14 PM - 7:32 PM	Conception	0	Worked on designing the XML structure for email messages. Designed the XML layout, Document type definition and explanation. These topics were then uploaded to the groups wiki for review, comment, and inclusion in the design document.
Feb 11, 2013, 5:12 PM - 8:04 PM	Conception	0	Worked on the team's design document and formatting of the sections. Took the information from the group's wiki and generated the data structure and IO format sections from this information. The other sections were expanded from the initial proposal with updates to relevant sections where the server information had been changed.
Feb 12, 2013, 10:30 PM - 11:45 PM	Conception	0	Team Dijkstra's regular meeting time. We reviewed the team design document and made many changes to the XML format and found several errors. We marked up the design for submission on Thursday.
Feb 14, 2013, 12:15 AM - 12:30 AM	Conception	0	Derived the format, in XML, for the transportation of the registration of a user into the address book for the server. Also formulated the steps to be involved in order to invoke the appropriate modules.
Feb 14, 2013, 12:30 AM - 1:45 AM	Coding	0	Wrote the shell script that acquired the contents of the source file (ACL2 definitions) and compiled a function invocation to be written to a temporary file, input directed into ACL2 and removal of the temporary file created. Considering just including the book to the source file and writing a new temporary file that invokes the function and includes to appropriate books in order to not manipulate the source files for unexpected results.
Feb 14, 2013, 2:00 AM - 4:00 AM	Coding	0	Created functions for managing address-book. getAddress, parseAddresses, getAddressBook acquire values from the XML that was passed to the script via redirection through the XML that was acquired through the shell scripting.
Feb 14, 2013, 6:15 AM - 8:00			Created the functions for managing output of the address-book data structure. addressXML, addressBookXML and getAddressBookXML will acquire the XML for the structure to be written back to the xml data file. Determined that I will need to write to a temporary file and delete the original

AM	Coding	0	source, and rename the temporary file to the permanent persistent file in order to allow for the file to be updated (since I cannot write back to a read file). Updated shell script to reflect these changes in ./server/modules/user/register/register-user.sh.
Feb 16, 2013, 1:15 AM - 3:00 AM	Coding	0	Added functions that acquires the contents of the tokens passed by parsing the XML information. getDomain, getName, getPassword all use these tokens in order to extract the information from the registration file that will be used to determine what information needs to be added to the address-book.
Feb 16, 2013, 3:00 AM - 5:00 AM	Coding	0	Added functions that will test the address existence (to prevent duplication) and add/remove an address from the address-book. isInAddressBook tests the predicate conditions to determine if an address can be added/removed by addAddress/removeAddress functions.
Feb 17, 2013, 3:43 PM - 4:51 PM	Coding	0	Worked on designing the server email components of the server module. Wrote out data flow diagrams to match the data structure format and XML I/O from the design document, and wrote function prototypes.
Feb 18, 2013, 5:05 PM - 6:12 PM	Conception	0	Typed the progress report for the presentation tomorrow to give Dr. Page an update on our projects task and goals. Also updated the design document to Revision I for submission tomorrow.
Feb 18, 2013, 7:23 PM - 10:44 PM	Coding	0	Worked on implementing the server email components. Took my designs on paper from yesterday and worked them into working functions. I was able to finish an email splitter which splits one email sent to many recipients into multiple messages and XML output for email messages.
Feb 19, 2013, 9:00 AM - 10:30 AM	Coding	0	Updated shell script to generate static script that will invoke the registerUser function also defined in the register-user.lisp file. Temporary lisp file is generated with static XML information and then redirected into the ACL2 program in order to parse the information. File input is delegated to the shell, file output is delegated to ACL2.
Feb 19, 2013, 10:30 AM - 11:45 AM	Conception	0	Team Dijkstra's regular meeting time. We delivered a progress report to Dr. Page on our task so far. Also we updated our SVN repository to the most recent changes to the groups implementation. Isaac rejoined the group and the remaining time was spend catching him up on the groups progress.
Feb 19, 2013, 3:00 PM - 4:00 PM	Coding	0	Corrected references the xml-scanner to ignore whitespace since some of the whitespace tokens that were being parsed were causing issues with the interpretation of the xml tokens. user/registration module deemed complete at current point in time.
Feb 22, 2013, 4:32 PM - 7:11 PM	Coding	0	I worked on the ACL2 implementation of the server-email functions and finished the implementation of the components. The functions did not admit correctly and the details are described in the defect log. The issues were traced to file IO so we are still able to run and test the computational functions.
Feb 22, 2013, 2:59 PM - 4:16 PM	Coding	0	I re-worked the file IO functions to accommodate the requirements of the IO utilities functions. This fixed the majority of the admission issues for these functions. Still the getEmail and runEmail functions

			are the global execution functions still have work needed to get them to admit and run with ACL2.
Feb 25, 2013, 8:00 AM - 9:30 AM	Conception	0	Starting in the development of the network connectivity subsystem that will allow for the transmission of information across a network via IP address. Determined previously that the "nc" command in UNIX was sufficient in order to accomplish this task Proceeded to decompose the component and derived the need for a separate "method invocation" language.
Feb 26, 2013, 8:00 AM - 9:15 AM	Conception	0	Proceeded to develop a rough draft for the diagram that would be formalized into the design document for network connectivity. Upon further investigation, I realized that I need to find a way in order to send a request to the server to open a sending port after the client has acknowledged opening a recieving port for data transmission. This appears to be the most difficult scenario for the server to overcome.
Feb 26, 2013, 10:30 AM - 11:45 AM	Coding	0	Team Dijkstra's regular meeting time. We brainstormed bug issues and found a solution to the Server-Email IO problem. We will resolve multiple email messages from the command shell and call ACL2 for each individual message.
Feb 27, 2013, 12:01 PM - 12:53 PM	Coding	0	Worked on the IO contents and designing the Test and Theorems for the server email module. Had an idea to combine the IO into one function to see if it fixes the IO issue I'm having with the server-email. Started implementation and will finish tonight.
Feb 27, 2013, 9:54 PM - 10:32 PM	Coding	0	I finally have the IO fixed. With one call I can have and input file in XML format parse and get passed to an output file. Now its time to handled multiple XML messages in one file.
Feb 28, 2013, 10:30 AM - 11:45 AM	Coding	0	Team Dijkstra's regular meeting time. We worked on server implementation and fixed issues with the server-email file. We split it into two files where one contained the IO functions and the other contained the logic functions. This was done in order to make proving theorems in ACL2 easier.
Feb 28, 2013, 4:02 PM - 5:12 PM	Testing	0	I wrote the theorem suite for the server-email functions. These tested each function in the file at least once. This guarantees that the written code does what it is needed to do.
March 1, 2013, 12:35 AM - 1:45 AM	Conception	0	Started to develop the basis for the "Server Messaging Language" that would overcome the need for remote method invocation. Syntax of which would be encapsulated in curly brackets and separated by semicolins. Started prototyping functions to scan for this language. Header information would be passed via first line in the XML transmission over the network and would be saved in a timestamp relative to UNIX conception date (date +%s.txt).
March 2, 2013, 4:53 PM - 5:34 PM	Coding	0	I wrote a shell script that dynamically writes a lisp file with the command to call the rwEmail function and output the files to a directory with the clients name in the servers store folder. Each file is names msg_timestamp where the timestamp is the actual timestamp of the message.
			Started to rough draft a client side development plan. Wes and I are considered the "server side" team, while we had hopes of Adam and Isaac being the "client side" team. Being that both Adam and

March 4, 2013, 8:00 AM - 8:45 AM	Conception	0	Isaac were gone, Wes and I have had to develop a plan of action toward pushing for project completion. RMI has been temporarily placed on hold until we can solve these issues. Perhaps and explanation of the RAD process and a bit of motivation would be the route to go. Designed a flow chart for an overview of the basic functionality and what components were complete. Decided that I would give the presentation on the update of the status of the project.
March 4, 2013, 11:49 AM - 1:04 PM	Coding	0	Worked on updating rwEmail. I changed the output file from being solely passed in by the parameters to where only a timestamp is required. The output directory is now dynamically generated to pull the tag from the XML and outputs to output each email message into a directory based on the contents of the tag and the naming convention that was implemented earlier for each individual file.
March 5, 2013, 7:15 PM - 8:30 PM	Coding	0	Started development on the RMI language. Instead of developing this as a module, I have decided to place this in the "includes" folder since it would more than likely be used by other modules. Point of entry is getActions function which would return a data structure of actions that will be updated in the next design document. On a side note, we can use this for user authentication where required instead of a separate request for authentication.
March 5, 2013, 2:30 PM - 2:45 PM	Testing	0	Tested out some shell solutions to multiple file traversal, since Wes was stating he needed to find a resolution for this issue, as discussed in our meeting earlier today. Posted a short "how-to" on the for loop and file acquisition. Wes responded shortly after implementing the code to verify its working condition. It appears to be the solution we were looking for. Considering implemenation in other modules as well as the register-user module.
March 5, 2013, 7:24 PM - 7:48 PM	Coding	0	Worked on updating the shell script to include a for loop to process each file that exist in the incoming directory. Also edited, the route-email ACL2 file to change the directory structure of the email output.
March 7, 2013, 12:00 AM - 2:30 AM	Coding	0	Continued development on the RMI language interpreter. Developed last few functions getActionString, getAction. Brackets encapsulate the actions, and semicolins separate the actions. SERVERACTION denotes a server action to take place. CLIENTACTION denotes that a client action should take place. At the time of this file conception, MOVEFILE and COPYFILE actions are only considered. The portion containing the string between the brackets is the module that will be considered. This will acquire the "monitor" value from the module registration on the server side. In otherwords, the file that contains the "header" information will be MOVED/COPIED (depending on action) to the monitor for the module that is in the brackets, thus invoking the monitor process and kick starting the module itself. Files that contain monitor information will end with a file extension of $unix_timestamp.rmi. When this file is copied, the header information is removed and the output is the following XML that is contained in the file ($unix_timestamp.xml).
March 7, 2013, 7:30 PM - 8:30			Creation of the create-user-request.lisp file.

PM	Conceptual	0	Planning how to generate XML for user access requests
March 9, 2013, 7:59 PM - 9:25 PM	Coding	0	Worked on the client code for the email module. I wrote the functions to parse strings into XML files for outgoing messages. I also handled the need for the client to sent an email to multiple recipients to where a separate XML file will need to be written for each recipient indicated in the to field of the email xml.
March 10, 2013, 5:30 PM - 7:30 PM	Coding	0	Completion of createRequests function in create-user-request.lisp. Creation of the create-user-request.sh script. This script invokes the create-user-request.lisp file
March 11, 2013, 12:04 PM - 12:51 PM	Coding	0	I worked on writing IO code for the client side email module. This included writing code for both incoming and outgoing messages. Since these needed to be handled differently, there are multiple functions for each of these requirements.
March 11, 2013, 8:13 PM - 9:18 PM	Coding	0	I finished the IO code for the client email. It now handles multiple recipients and will output a single XML file that will need to be processed to send the multiple email messages. A shell script was written to handled ALL incoming messages and process them to HTML files for easy reading.
March 12, 2013, 11:46 AM - 12:49 PM	Coding	0	The client email code has been finished. Worked on tweaks to get the output correct and in the correct XML format. Worked on naming of files to ensure unique naming for each generated file. Finished a shell script to split the email output that contains multiple recipients.
March 13, 2013, 9:00 AM - 10:30 AM	Coding	0	Began adding shell commands to extract the domain and name from the registration files to be able to create the directories for the email storage on the server.
March 13, 2013, 5:13 PM - 7:14 PM	Testing	0	Tested the client code. Worked on verification of the connection between the logic module and the IO module. Ensured that the correct functions were called, and returned the correct structures. This is set up for writing the theorems for this module. Also added the getHTMLtext to the client logic to allow an HTML file to formed and written instead of regular plain text for a nicer looking output.
March 14, 2013, 6:43 PM - 7:34 PM	Coding	0	Wrote the shell scripts to automate the client email processing. One shell script was made to handle incoming email messages and directs incoming messages to the inbox folder. The other shell script handled outgoing messages and places them in the outbox folder.
March 15, 2013, 9:46 PM - 10:52 PM	Coding	0	Added to the outgoing shell script the capability to parse multiple emails and create an XML file that contains a single email message with a unique file name. This shell script will then send the single email script to the server using the nc command.
March 17, 2013, 12:15 AM - 12:48 AM	Coding	0	Finished adding the directory creation mechanisms to the shell scripts and tested the user registration process. It seems to be working correctly now.
March 17, 2013, 12:50 AM - 12:55 AM	Testing	0	Continued to test the functionality of the user registration process on the server side. One thing to note, we cannot have spaces in our name and domain. Whitespace has been trimmed.

March 17, 2013, 9:30 PM - 10:00 PM	Coding	0	Changed parameters of createRequests function to accept an argument for the time stamp of creation used by the create-user-request.sh script
March 17, 2013, 1:00 AM - 1:15 AM	Coding	0	Added and modified lines in the register-user and address-book lisp files to allow for a user to register with a password. This will be used to verify the user can perform actions on the server side.
March 17, 2013, 1:15 AM - 3:00 AM	Coding	0	Created the verify action on the user module. Defined utility functions to acquire domain, name, password and the location of the client. The location will need to be extracted from the original XML from the shell script since it does not conform the the verification user information.
March 17, 2013, 3:00 AM - 6:17 AM	Coding	0	Created all the functions that will perform the actions on the mailing list. subscribe, unsubscribe, and all predicates to allow for these actions to complete.
March 17, 2013, 6:37 AM - 6:45 AM	Testing	0	After looking at some of the properties from the address-book_tests file, I realize that I forgot to log these functions into the PSP logs from last cycle (probably because I did them right before class and lost track of time). These have been added to this log.
March 23, 2013, 2:00 AM - 2:45 AM	Coding	0	Added verify-user.sh file to invoke the actions required to start the user verification process on the server side. Having a few issues getting remote-actions.lisp to be accepted into logic (guard checking issues).
March 23, 2013, 12:00 AM - 2:50 AM	Coding	0	Finished the verify-user.sh action file for establishing a connection between the client and server for mail reception.
March 24, 2013, 4:58 PM - 6:18 PM	Testing	0	Worked on the theorem suite for the client email module. The theorems are used to test the data integrity of the logic functions in the client. We were unable to test the IO functions with theorems since they rely on variant data. However, the logic could be tested using theorems and we were able to prove that the client email logic module returns the correctly formatted data based on correct input.
March 25, 2013, 6:12 PM - 6:49 PM	Coding	0	Tweaked the shell scripts for the clients to ensure that the server has ample time to process a single email message and that the client will not overload the server by adding too many emails to the queue.
March 25, 2013, 7:30 PM - 8:30 PM	Conceptual	0	Creation of the create-block-request.lisp file. Planning how to generate XML for requests to block users
March 25, 2013, 9:30 PM - 11:00 PM	Coding	0	Completion of the function responsible for generating xml for creating requests to block users as well as I/O functions
March 25, 2013, 11:30 PM - 11:59 PM	Coding	0	Creation of create-block-request.sh script. This script invokes the create-block-request.lisp file
March 27, 2013, 9:30 PM - 10:30 PM	Coding	0	Creation of create-mailing-list.lisp file. Creation of the XML Generating functions addressesXML and ownerXML
March 28, 2013, 7:00 AM - 8:00 AM	Coding	0	Creation of file output method
April 1, 2013, 5:55 PM - 6:47			Worked on reworking the Java Gui client to work directly with ACL2 instead of relying on shell scripts.

PM	Coding	0	This allows faster and more secure connections between remote host for our networking portions.
April 1, 2013, 8:59 PM - 11:10 PM	Coding	0	Continued to work on the Java integration with ALC2. I was able to complete most of the Send email functions for the client. All that remains on this front is sending the file's contents over the network.
April 2, 2013, 6:32 PM - 9:48 PM	Coding	0	I reworked the client actions into the required Java programs. They now work with the server and can send and receive information. The scripts that handled the ACL2 function calls were also integrated into the GUI interface and are easily accessible to the user if they are running the interface.
April 2, 2013, 9:49 PM - 10:04 PM	Testing	0	I found a bug with the server code that was already implemented. When looping though all files in a directory, it was picking up extra hidden files as well, This caused the server to crash. I added code to fix this issue.
April 3, 2013, 5:47 PM - 8:11 PM	Coding	0	I finished coding the Java integration for the current ACL2 implementation. The client side of the program can now send and receive emails and User registration. This also includes user verification in order to get email messages.
April 4, 2013, 4:31 PM - 5:34 PM	Coding	0	I added the delete function to the client GUI interface. I also worked on fixing a bug on the transmission of messages. The current issue is that the verification module does not allow for correct exceptions to be processed from the server. If a transmission fails, it does not handle the output correctly and sends an incorrect XML file back to the client.

Defect Log

Date	Phase	Fix Time	Description
Jan 18, 2013	Networking	60	IP resolution cannot occur in ACL2 unless a network drive is mapped, after which you can call it by its network path. Networking drives may be our resolution to this issue.
Jan 22, 2013	Design	60	Found out that networking is not feasible from within ACL2. To make it natively supported, writing and extending several Common Lisp features would need to be done. We cannot do this in the scope of this project. So we found that using the Operating System's native filesystem and networking support would be much more friendly to deal with once we get to this stage in the project.
Jan 29, 2013	Design	147	Found out that GUI's and File IO cannot coexist in ACL2. We can have one or the other, but not both. So we had to scrap the GUI portions of the project and replace them with a new idea. The new idea is the current design of the email server and client system. This project is strictly data processing and file IO. This project will be file and text based rather than Visual and Interactive.
Jan 29, 2013	Conception	150	ACL2 cannot work with a GUI and IO at the same time. Had to reevaluate how to salvage what we had regarding design. Instead of real time chat, we would be sending email. Instead of ACL2 interfaces, we would program them in C or C++.

Jan 31, 2013	Application Support	90	Unable to get Proofpad to work correctly on any of my computers. Windows 8 and Ubuntu both show NullPointerExceptions when trying to implement ACL2 and Proofpad will not correctly identify with ACL2. Opted to use Dracula instead.
Feb 3, 2013	Conception	150	Determined that the size of the executables generated by ACL2 would not be a practical application for our program. Opted to use input redirection into ACL2 prompt and shell script invocation. Ubuntu would be the server platform and the two Macs would be used as clients to send and recieve information.
Feb 5, 2013	Personnel	15	With the loss of Isaac from the team, modules had to be prioritized for completion and new due dates had to be set. Determined that I would need to finish the XML Parser as quickly as possible to begin development in order to maintain deadlines.
Feb 5, 2013	Design	124	We discovered that generating ALC2 executables and invoking these files from an outside source is a troublesome experience and that the generated files are hundreds of megabytes in size. Since we will have several modules for this project, we saw this as a negative side effect of executable files. To solve this problem, we moved all our project to the UNIX platform. This has allowed us to use the UNIX shell environment to generate shell scripts that invoke the ACL2 environment while passing in ACL2 source code files. This reduces the size of the files to kilobytes and streamlines the execution process and eliminating the size of outside programming needed for the original idea to work. Thus we will be executing our ACL2 code through a UNIX shell script and the shell scripts will in turn be executed from the outside programming environment.
Feb 12, 2013	Design	75	When we looked at the design review. We saw several errors in the XML format that would not pass if it were to be sent though a web browser. We had to work on setting the XML to a correct format and modify the document type definitions to comply with proper XML syntax.
Feb 15, 2013	Coding	10	Made the decision to allow shell scripts to take care of much of the IO on the read side as possible and file operations would need to be done by the OS in order to keep correct RWE privs on the file for security purposes. CHMOD properties will need to be determined at a different date, since access to store files has not been completely determined. Best guess is that server will be the only one that needs read/write access to these files and no user group will need execution access.
Feb 19, 2013	Coding	20	Issue arose when parsing XML tokens that the whitespace in the document was being identified by
Feb 22, 2013	Coding	12	After finishing coding the ACL2 functions, the functions would not admit under normal ACL2 invocation. However, it did work under Dr Racket. The issue was traced to the IO and List utilities files as they were un-certified files within the regular ACL2 environment. Adding the suppression to the certification requirement, the files worked as usual.
			After fixing the certification issue. Certain functions were still not admitting to ACL2. This was due to illegal arguments as the ACL2 output stated. To fix

Feb 23, 2013	Coding	52	these arguments, the state variable had to be set and implemented differently than I had intended. I added some safe guards to the variables and added constraints to the functions that depended on them.
March 4, 2013	Personnel	45	Has to reconsider a new "plan of action" with regards to completion of the project. Current methods seemed to archaic for the current situation as we were unable to adapt due to the relaince on people. Opted for a RAD development solution where SCRUM and Extreme Programming were the foundations for development. Will need to explicitly sit down and speak with team to describe the course of action. Developed a visual aid to describe where we are and what is completed. There seems to be more questions on this as opposed to the completion of the project.
March 5, 2013	Coding	21	After finishing the IO entry point function on the server email module, We noticed that the XML files were being generated with a comma instead of the @ symbol between names and domains in the email address. Also, the output file for the email was a statically named file. This file needed to be dynamically named with a timestamp.
March 11, 2013	Coding	24	Email parsing had a one off error that did not have the correct lines returned for the XML file which rendered the outputted file useless.
March 13, 2013	Coding	20	Was not able to get email to write to the server. Determine that it was an issue that the directories did not exist, thus the information was not being written properly by the ACL2 script, which was a difficult thing to determine since acquiring error information from the runtime environment would only be possible if we wrote the error to an output log. Determined that I would need to finish the registration process by adding directory creation to the shell script in order to make this function correctly. Until then, we can manually create the directories.
March 14, 2013	Coding	32	Shell script was not correctly splitting the XML files based on the regular expression. Started using awk to parse the file. However, the file did not have a unique file name and was getting overwritten every time the script was executed.
March 23, 2013	Coding	35	Type checking for string-listp on XML conversion in the server actions was incorrect. Was using endp and stringp tests, when combined I could use string-listp, which ended up being the required fix and not having to turn off guard checking.
March 24, 2013	Testing	16	The theorem that tested the email data structure was not passing. This was traced to an error in the proof and not in the code. The error was trying to access an item that was not in the structure, hence the failure of the proof.
March 27, 2013	Design	0	Need to restructure the create-mailing-list.lisp file to handle multiple addresses and multiple owners.
April 1, 2013	Coding	23	Working on integrating the ALC2 with Java. Having trouble getting Java to see the files that ACL2 has generated. Right now, the current solution is to make the Java sleep for a couple of seconds while ACL2 finishes its processing then resume. Then it sees the files that ALC2 generates

April 2, 2013	Testing	15	The script that Matthew had written to open all files in a directory was not working. It was needed that the function open only the xml files, since there are hidden files in a directory, I had to modify the function to account for these changes.

Additional Features

The requirements mentioned so far in this project proposal are far from comprehensive. Numerous possibilities exist for expanding this project and adding additional features. In this section we will discuss extending the core features of the program.

These features outlined here, along with the original features that do not get implemented in the project will be available for the other groups in Software Engineering to implement.

Private Messaging with Attachments

This feature will extend each level of this project (i.e. Server, Client, I/O, and Networking). In order to implement secure messaging, we will need to setup facilities on the server to set up secure messaging. To do this we must add the following functions on the server.

- **isSecureMessage** - This function will parse the message's XML formatting to determine if the message is a secure message.
- **postToSecureClient** - This function will send the secured messages to the proper secure clients and not the general population of unsecured clients.
- **getSecureMessages** - This function will open the secured message buffers and act similarly to getClientMessages.
- **getSecurityCredentials** - This function will obtain security credentials from the client to see if it is authorized to receive secure messages.
- **authorizeSecureClient** - This function will take the client's user credentials and validate.
- **registerSecureClient** - This function will register the client once the credentials are authorized.

The client program will do most of the work in the secured messaging. This allows for minimal server intervention and allows the messages to be encrypted as well. The client must perform the following functions:

- **isSecuredMessage** - This function will determine whether or not the most recently received message is a secured message.
- **setSecuredType** - This function will set the secured output in XML format to the client's buffer.
- **getSecuredOutput** - This function will parse a secured message's text format and return the function as a series of tokens.
- **postSecuredMessage** - This function will post the secured message to the server's secure chat buffer file.
- **inputEncryptionCode** - This function will take in the user's encryption key and use it to encrypt/decrypt messages.
- **encryptMessage** - This function will encrypt the message.
- **decryptMessage** - This function will decrypt a message.

In addition to secure messaging. The capability of secure attachments needs to be implemented. In order for attachments to be handled. The I/O XML format will need to be updated to accommodate an attachment tag. The secure attachment functionality will need these components:

- **openAttachmentFile** - Opens the attachment to send in a message.
- **attachmentToBytes** - Converts the attachment to a byte stream.
- **attachmentToXML** - Generates an XML tag for the attachment.
- **addToBody** - Adds to the email message that an attachment is being sent.
- **encryptAttachment** - Encrypts the attachments bytes for secure transmission.
- **decryptAttachment** - Decrypts the attachments bytes.
- **readAttachmentXML** - Reads XML metadata for the attachment.
- **sendAttachmentToServer** - Sends the attachment to the server for forwarding.
- **getAttachmentsFromServer** - Retrieves attachments from server.
- **setAttachmentKey** - Sets the attachment's encryption key. 0 should be used for no encryption.

The file used to generate the following PROBE estimate is attached in Appendix B. The PROBE estimate for this additional functionality is attached below:

Additional Features

Personal Software Process Summary

Project Essentials

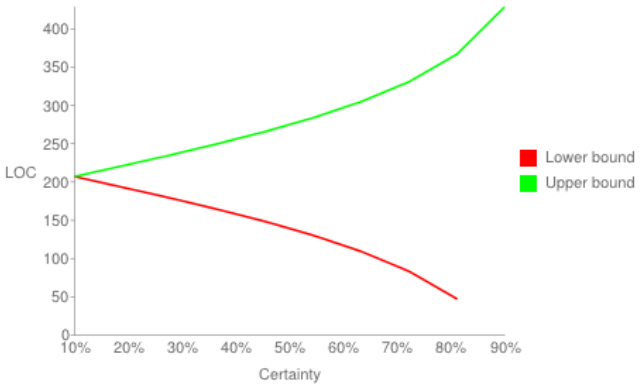
Name: Team Dijkstra
Instructor: Dr. Rex Page
Date: Jan 31, 2013
Language: ACL2

Lines of Code

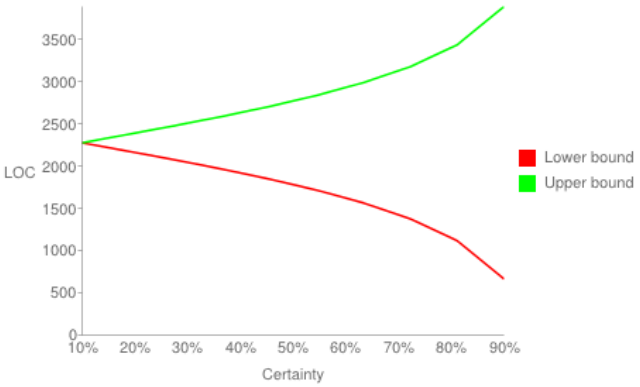
Type	Prediction by user	Actual
Added	303	---
Base	0	---
Modified	0	---
Removed	0	---

PSP Projection

LoC Certainty



Time Certainty



Project Data

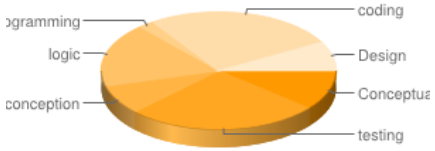
Time Per Defect Type
No data

Time Per Phase
No data

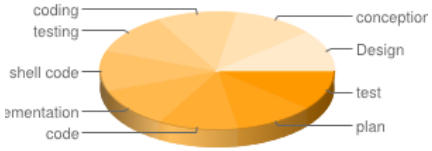
Time by Day
No data

Cumulative Data

Time Per Defect Type



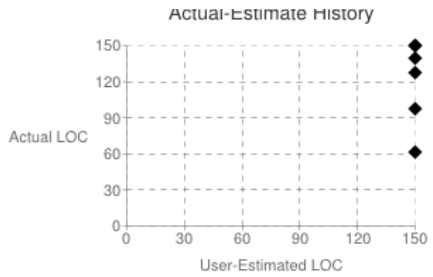
Time Per Phase



Actual vs Estimated LoC

Actual vs Estimated LoC

1/29/13 PSP Report



Project	Estimate	Actual
tImpl	307	288
tCodeRvw	258	41
iEx5	198	189
iEx4	111	65
iEx3	173	93
iEx0	120	85

Time Log

Date	Type	Int. Time	Description

Defect Log

Date	Phase	Fix Time	Description

Appendix

Appendix A: ACL2 Source Code Files

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Client-email.lisp
;
; Created on March 7, 2013 by Wesley R. Howell.
;
; Purpose:
; This file will execute the email generation from a client.
;
; CHANGE LOG:
; 3/11/2013 - Added the getEmailText function
; 3/7/2013 - Created file to handled the client logic for email messages
; ---------------------------------------------------------------------
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(in-package "ACL2")

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Base Lines
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;(break-at-set delimiters xs)
;Function copied from the list utiliites package. This function is copied
;since it is not necessary to load all the functions in this package and
;have ACL2 certify each function.
(defun break-at-set (delimiters xs)
  (if (or (not (consp xs))
          (member-equal (car xs) delimiters))
      (list '() xs)
      (let* ((first-thing (car xs))
             (break-of-rest (break-at-set delimiters (cdr xs)))
             (prefix (car break-of-rest))
             (suffix (cadr break-of-rest)))
          (list (cons first-thing prefix) suffix))))

;(break-at delimiter xs)
;Function copied from the list utiliites package. This function is copied
;since it is not necessary to load all the functions in this package and
;have ACL2 certify each function.
(defun break-at (delimiter xs)
   (break-at-set (list delimiter) xs))

;(getEmailXML email)
;returns the XML formatted code for the email object passed
;param email - the email structure
;return string
(defun getEmailXML (email)
  (if (endp email)
      nil
      (let* ((hd (concatenate 'string
                              "<?xml version='1.0'?>"
                              "<!DOCTYPE user SYSTEM '../../../dtd/messages.dtd'>"
                              "<email>"))
             (to (concatenate 'string
                              "<to>" (car (car (car email))) "@"
                              (car (cdr (car (car email))))
                              "</to>"))
             (from (concatenate 'string
                                "<from>" (car (cadr email)) "@"
                                (car(cdr (cadr email)))"</from>"))
             (sub (concatenate 'string
                               "<subject>" (cadr (cdr email)) "</subject>"))
             (msg (concatenate 'string
                               "<content>" (cadr (cdr (cdr email)))
                               "</content>"))
             (ft (concatenate 'string "</email>"
                              )))
              (list hd to from sub msg ft))))
```

```
;(getContactStructure str)
;gets the contact structure for a given string
;name@domain
;param - str - the string to parse
(defun getContactStructure (str)
  (let* ((chrs (coerce str 'list))
         (name (car (break-at #\@ chrs)))
         (domain (cdr (break-at #\@ chrs))))
        (list  (coerce name 'string)
               (coerce (cdr (car domain))
                       'string))))

;(getEmailStructure xml)
;Gets the email data structure for the tokenized XML file passed
;param - xml - the tokenized list of XML contents
(defun getEmailStructure (xml)
  (let* (;(xml (cdr (getEmailXMLTokens file)))
         (to (car (car (cdr (cdr xml)))))
         (from (car (car (cdr (cdr (cdr (cdr (cdr xml))))))))
         (sub (car (car (cddddr  (cddddr xml)))))
         (msg (car (car (cddddr (cddddr (cdddr xml)))))))
    (list (list (getContactStructure to))
                (getContactStructure from) sub msg)))
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;(splitToString toString)
;Returns the string value of a list of characters
;param - toString - the list of characters
(defun splitToString (toString)
   (coerce toString 'string)
)

;(splitToStruct toList)
;The recursive fuction to separate a string based to field
;separated by commas ',' to a list of contact structures
(defun splitToStruct (toList)
        (if (consp (car (cdr toList)))
            (cons (splitToString (car toList))
                   (splitToStruct
                    (break-at #\, (cdr (car (cdr toList))))))
            (list (splitToString (car toList)))
          )
)

;(splitToField toString)
;Entry point for the 'to' field parsing. This function will break apart
;the to field into a list of characters and then call the splitToStruct
;param - toString - the string to be parsed
(defun splitToField (toString)
  (let* ((tos (break-at #\, (coerce toString 'list))))
    (splitToStruct tos)
  )
)

;(generateEmailFromStrings to from sub msg)
;This will be the main logic point for the user input.
;taken string values entered from the user, this function will
;gereate the XML format for an email message.
(defun generateEmailFromStrings (to from sub msg)
   (let* (
         (tos (getContactStructure to))
         (fr (getContactStructure from))
         (subject sub)
         (message msg))
         (list (list tos) fr subject message))
)
```

```
;(multRecip toList from sub msg)
;This function will generate a list of Email XML objects for
;each recipient specified in the to field of an email message
(defun multRecip (toList from sub msg)
    (if (consp (cdr toList))
        (append (getEmailXML (generateEmailFromStrings (car toList)
                                                from sub msg))
                (multRecip (cdr toList) from sub msg))
        (append (getEmailXML (generateEmailFromStrings (car toList) from sub msg)))
     )
)

;(email to from sub msg)
;Generates the email XML for email messages based on user string input.
(defun email (to from sub msg)
   (if (consp (cdr (splitToField to)))
     (multRecip (splitToField to) from sub msg)
    (getEmailXML (generateEmailFromStrings (car (splitToField to)) from sub msg))
   )
)

;(getEmailText tokxml)
;This function will be the base for the client output. This fuction will
;return the contents of an email message to the client
(defun getEmailText (tokxml)
  (let* ((xml (getEmailStructure tokxml))
             (to (concatenate 'string
                              "To: " (car (car (car xml))) "@"
                              (car (cdr (car (car xml))))
                              ))
             (from (concatenate 'string
                                "From: " (car (cadr xml)) "@"
                                (car(cdr (cadr xml)))))
             (sub (concatenate 'string
                               "Subject: " (cadr (cdr xml))))
             (msg (concatenate 'string
                               "Message: " (cadr (cdr (cdr xml)))
                               ))
        )
              (list to from sub msg))
)

;(getEmailText tokxml)
;This function will be the base for the client output. This fuction will
;return the contents of an email message to the client
(defun getEmailHTML (tokxml)
  (let* ((xml (getEmailStructure tokxml))
         (hd (concatenate 'string "<html><body>"))
             (to (concatenate 'string
                              "<h2>To: " (car (car (car xml))) "@"
                              (car (cdr (car (car xml)))) "</h2>"
                              ))
             (from (concatenate 'string
                                "<h2> From: " (car (cadr xml)) "@"
                                (car(cdr (cadr xml))) "</h2>"))
             (sub (concatenate 'string
                               "<h1>" (cadr (cdr xml)) "</h1><hr><br>"))
             (msg (concatenate 'string
                               "<p>" (cadr (cdr (cdr xml))) "</p>"
                               ))
             (ft (concatenate 'string "</body></html>"))
        )
              (list hd sub to from msg ft))
)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; rw-email.lisp
;
; Created on March 11, 2013 by Wesley R. Howell.
;
; Purpose:
; This file will contain the dependencies to other .lisp files and
; handled the IO portion of the client functions.
;
; CHANGE LOG:
; 4/ 1/2013 - Changed the file path for outputted files to be consistant with
;             new Java interface.
; 3/11/2013 - Created file to handled the client logic for email messages
; ---------------------------------------------------------------------------
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(in-package "ACL2")

(include-book "../../../include/xml-scanner" :uncertified-okp t)
(include-book "../../../include/io-utilities" :uncertified-okp t)
(include-book "../client-email" :uncertified-okp t)
(set-state-ok t)

;(readEmailText file state)
;Reads an email message, then outputs the contenets to the console.
;Working on getting it to write out to an HTML file.
(defun readEmailText (file state)
  (mv-let (input-xml error-open state)
          (file->string file state)
     (if error-open
         (mv error-open state)
         (mv (getEmailText (cdr (cdr (tokenizeXML input-xml)))) state))))

(defun readEmailHTML (file state)
  (mv-let (input-xml error-open state)
          (file->string file state)
     (if error-open
         (mv error-open state)
         (mv (getEmailHTML (cdr (cdr (tokenizeXML input-xml)))) state))))

(defun readEmail (fin fout state)
  (mv-let (input-as-string error-open state)
      (file->string fin state)
     (if error-open
         (mv error-open state)
         (mv-let (error-close state)
                 (string-list->file
                  (concatenate 'string "store/email/inbox/"

                               "msg_"
                               fout
                               ".html")
                     (getEmailHTML
                            (cdr (cdr
                             (tokenizeXML input-as-string)))
                            ;These two cdr's remove the two XML
                            ;headers for the file
                      )
                       state)
           (if error-close
               (mv error-close state)
               (mv "Success File has been written!"
                    state))))))

(defun writeEmailToFile (xmlStr to ts state)
  (mv-let (error state)
          (string-list->file (concatenate 'string
                                          "store/email/outbox/"
```

```
                                        to ts ".xml")
                            xmlStr
                            state)
     (if error
         (mv error state)
         (mv "XML File Written Successfully" state))))


(defun writeMessage (to from sub msg state)
    (let* ((msgs (email to from sub msg)))
       (writeEmailToFile msgs to from state))
)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; mailing-list.lisp
; Created on March 27, 2013 by Adam Ghodratnama
;
; This file will create a request to register a mailing list.  An XML mailing
; list file will be created from the following arguments:
; -Mailing list name (name string)
; -List of adresses '('(domain1 name1)... '(domainN nameN))
; -And an owner (name password)
;
; CHANGE LOG:
; ---------------------------------------------------------------------------
; 2013-03-27    -   Initial file conception.
;                   -       Creation of XML Generating functions:
;                           addressesXML and ownerXML
; 2013-03-28        -       Creation of file output method
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(in-package "ACL2")
(include-book "../../../include/io-utilities")
(include-book "../../../include/xml-scanner")

; (addressesXML address)
; Generates the XML form for an address from the address structure.
; address - the address structure in the form of (domain name).
(defun addressesXML (address)
    (if (endp address)
        nil
            ; Domain tag <domain>#PCDATA</domain>
        (let* ((domain (concatenate 'string
                            "<domain>" (car address) "</domain>"))
                       ; Name tag <name>#PCDATA</name>
                       (name (concatenate 'string
                            "<name>" (cadr address) "</name>"))
                       ; Create a list with the address fields in it
                       (regRequest (list domain name)))
            regRequest)))

;Generates the XML form for an owner for use in a mailing list
;owner - the owner structure in the form of (owner password)
(defun ownerXML (ownerInfo)
    (if (endp ownerInfo)
        nil
            ; Name tag <name>#PCDATA</name>
        (let* ((name (concatenate 'string
                            "<name>" (car ownerInfo) "</name>"))
                       ; Password tag <password>#PCDATA</password>
                       (password (concatenate 'string
                            "<password>" (cadr ownerInfo) "</password"))
                       ; Create a list with the ownerInfo fields in it
                       (regRequest (list name password)))
            regRequest)))

; The purpose of this function is to create an XML file that requests
; access for the creation of a mailing list.
;
; The function will create an XML file containing the name of the mailing
; list, addresses to be sent to, and the owner
; password for a user
;
; Example call: (createRequest '("localhost" "adam" "password") timeStamp state)
;
(defun createRequest (name addresses owner timeStamp state)
  (let* ((domain (car (addressesXML addresses)))
         (name (cadr (addressesXML addresses)))
         (ownerName (car (ownerXML owner)))
         (ownerPass (cadr (ownerXML owner))))
    (mv-let (error state)
```

```
(string-list->file
 (concatenate 'string
      (string-append "request-mailing-list" (rat->str timeStamp 0))
      ".xml")
 (list
      "<?xml version='1.0'?>"
      "<!DOCTYPE mailing-list SYSTEM '../../../dtd/mailing-list.dtd'>"
      (concatenate 'string
                   (string-append "<list-name>" name)
                   "</list-name>")
      "<address>"
      domain
      name
      "</address>"
      "<owners>"
      "<owner>"
      ownerName
      ownerPass
      "</owner>"
      "</owners>")
 state)
(if error
    (mv error state)
    (mv "Wrote mailing-list creation request" state)))))
```

```
;
; Created on March 24, 2013 by Adam Ghodratnama
;
; Purpose:
; This file contains the functions that are essential to creating a request
; from the client side to block a user from sending emails to a client
;
; CHANGE LOG:
; ---------------------------------------------------------------------
; 2013-03-25    -   Initial file conception.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(in-package "ACL2")
(include-book "../../../include/io-utilities")
(include-book "../../../include/xml-scanner")

(set-state-ok t)
(set-ignore-ok t)

; (addressXML address)
; Generates the XML form for an address from the address structure minus.
; the password.
; address - the address structure in the form of (domain name).
(defun addressXML (address)
    (if (endp address)
        nil
            ; Domain tag <domain>#PCDATA</domain>
        (let* ((domain (concatenate 'string
                            "<domain>" (car address) "</domain>"))
                       ; Name tag <name>#PCDATA</name>
                       (name (concatenate 'string
                           "<name>" (cadr address) "</name>"))
                       ; Create a list with the address fields in it
                       (regRequest (list domain name)))
            regRequest)))

;
; The purpose of this function is to create an XML file that requests
; that a user is blocked from emailing the client for a user.
;
; The function will create an XML file containing the two addresses:
; 1. The address of the blocker
; 2. The address of the user to be blocked
;
; When this structure is sent to the server it will allow identification
; of both parties so that the appropriate actions are taken on the
; respective parties.
;
; Example call:
; (createRequest '("localhost" "blocker")
;     '("localhost" "blockee") timeStamp state)
(defun createRequest (blockerAddress blockeeAddress timeStamp state)
  (let* ((blockerDomain (car (addressXML blockerAddress)))
         (blockerName (cadr (addressXML blockerAddress)))
         (blockeeDomain (car (addressXML blockeeAddress)))
         (blockeeName (cadr (addressXML blockeeAddress))))
    (mv-let (error state)
            (string-list->file
             (concatenate 'string
                  (string-append "../../../store/user/requests/block/request-block" (rat->str timeStamp 0))
                  ".xml")
             (list
                  "<?xml version='1.0'?>"
                  ;"<!DOCTYPE user SYSTEM '../../../dtd/register-user.dtd'>" ; NEED DTD FOR BLOCK USER
                  "<blocker>"
                  blockerDomain
                  blockerName
```

```
      "</blocker>"
      "<blockee>"
      blockeeDomain
      blockeeName
      "</blockee>")
 state)
(if error
    (mv error state)
    (mv "Wrote block user request" state)))))
```

```
;
; Created on March 7, 2013 by Adam Ghodratnama
;
; Purpose:
; This file contains the functions that are essential to creating a request
; from the client side to register a user on the server.
;
; CHANGE LOG:
; ---------------------------------------------------------------------
; March 10, 2013 - Completion of createRequeset function
;
; March 17, 2013 - Changed parameters of createRequests function to accept
;                  an argument for the time stamp of creation used by the
;                  shell script
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(in-package "ACL2")
(include-book "../../../include/io-utilities")
(include-book "../../../include/xml-scanner")

(set-state-ok t)
(set-ignore-ok t)

; (addressXML address)
; Generates the XML form for an address from the address structure.
; address - the address structure in the form of (domain name password).
(defun addressXML (address)
    (if (endp address)
        nil
            ; Domain tag <domain>#PCDATA</domain>
        (let* ((domain (concatenate 'string
                            "<domain>" (car address) "</domain>"))
                       ; Name tag <name>#PCDATA</name>
                       (name (concatenate 'string
                           "<name>" (cadr address) "</name>"))
                       ; Password tag <password>#PCDATA</password>
                       (password (concatenate 'string
                            "<password>" (caddr address) "</password>"))
                       ; Create a list with the address fields in it
                       (regRequest (list domain name password)))
            regRequest)))
;
; The purpose of this function is to create an XML file that requests
; access for a user.
;
; The function will create an XML file containing the domain, name and
; password for a user
;
; Example call: (createRequest '("localhost" "adam" "password") timeStamp state)
;
(defun createRequest (address timeStamp state)
  (let* ((domain (car (addressXML address)))
         (name (cadr (addressXML address)))
         (pass (caddr (addressXML address))))
    (mv-let (error state)
            (string-list->file
             (concatenate 'string
                  (string-append "store/user/requests/register/request-register" (rat->str timeStamp 0))
                  ".xml")
             (list
                  "<?xml version='1.0'?>"
                  "<!DOCTYPE user SYSTEM '../../../dtd/register-user.dtd'>"
                  "<user>"
                  domain
                  name
                  pass
                  "</user>")
             state)
```

```
(if error
    (mv error state)
    (mv "Wrote registration request" state)))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; address-book.lisp
;
; Created on February 14, 2013 by Matthew A. Crist.
;
; Purpose:
; This file contains the functions that are essential to acquiring address
; book information and storing entries into the address book.
;
; CHANGE LOG:
; ---------------------------------------------------------------------
; 2013-04-07    - Adjusted isInAddressBook to only check domain and name.
; 2013-03-17    - Added password field to address book for verification.
; 2013-02-19    - Predicate isInAdressBook was making incorrect reference
;                 to variable address when checking endp for recursion.
;                 this caused stack overflow.  Corrected to endp
;                 addressBook.
;
; 2013-02-19    - Added predicate test to addAddress function that would
;                 determine if the address was already in the address
;                 book.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(in-package "ACL2")

; (getAddress tokens)
; Acquires the address structure for an address entry that exists in the
; address book. (domain name) form.
; tokens - the tokenized XML string that will be used to extract contact
;          information.
(defun getAddress (tokens)
  (if (endp tokens)
      nil
      (if (equal "</address>" (caar tokens))
          nil
          (if (equal "<domain>" (caar tokens))
              (cons (caadr tokens) (getAddress (cdddr tokens)))
              (if (equal "<name>" (caar tokens))
                  (cons (caadr tokens) (getAddress (cdddr tokens)))
                  (if (equal "<password>" (caar tokens))
                      (cons (caadr tokens) (getAddress (cdddr tokens)))
                      (getAddress (cdr tokens))))))))

; (parseAddresses tokens)
; Acquires all the addresses that are present in the tokenized XML string
; and returns the list of the addresses to the requestor.
; tokens - the tokenized XML string that will be used to extract address
;          information.
(defun parseAddresses (tokens)
  (if (endp tokens)
      nil
      (if (equal "<address>" (caar tokens))
          (cons (getAddress (cdr tokens)) (parseAddresses (cdr tokens)))
          (parseAddresses (cdr tokens)))))

; (getAddressBook tokens)
; Acquires the address book - point on entry for address book acquisition.
; tokens - the tokenized XML string that will be used to extract the
;          address book.
(defun getAddressBook (tokens)
  (if (endp tokens)
      nil
      (let* ((addresses (parseAddresses tokens)))
        addresses)))

; (addressXML address)
; Generates the XML form for an address from the address structure.
```

```
; address - the address structure in the form of (domain name).
(defun addressXML (address)
  (if (endp address)
      nil
      ; Domain tag <domain>#PCDATA</domain>
      (let* ((domain (concatenate 'string
                     "<domain>" (car address) "</domain>"))
             ; Name tag <name>#PCDATA</name>
             (name   (concatenate 'string
             "<name>" (cadr address) "</name>"))
             ; Address tag <address>[domain][name]</address>
             (password (concatenate 'string
                     "<password>" (caddr address) "</password>"))
             (address (concatenate 'string
                     "<address>" domain name password "</address>")))
        address)))

; (addressBookXML addressBook)
; Generates the non header portion of the XML address book output.
; addressBook - the address book structure to be converted to XML.
(defun addressBookXML (addressBook)
  (if (endp addressBook)
      nil
      (cons (addressXML (car addressBook))
            (addressBookXML (cdr addressBook)))))

; (getAddressBookXML addressbook)
; Converts the address book data structure into XML that can be stored to
; a document.
; addressBook - the address book stuctures that is to be converted to XML.
(defun getAddressBookXML (addressBook)
    (if (endp addressBook)
        nil
        (let* ((xml (append (append (list
    "<?xml version='1.0'?>"
    "<!DOCTYPE addresses SYSTEM '../../dtd/address-book.dtd'>"
    "<addresses>")
        (addressBookXML addressBook))
    '("</addresses>"))))
            xml)))

; (isInAddressBook addressBook address)
; Predicate to determine if an address is in the address book.
; addressBook - the address book to use to determine if an address is
;               contained in this address book.
; address     - the address in which we are to be locating.
(defun isInAddressBook (addressBook address)
  (if (endp addressBook)
      nil
      (if (and (equal (caar addressBook) (car address))
               (equal (cadar addressBook) (cadr address)))
          t
          (isInAddressBook (cdr addressBook) address))))

; (addAddress addressBook address)
; Appends an address onto the end of the address book structure.  If the
; user is already in the address book, then the return is the original
; address book.
; addressBook - the address book in which to add the address.
; address     - the address to add to the address book.
(defun addAddress (addressBook address)
  (if (equal (isInAddressBook addressBook address) nil)
      (append addressBook (list address))
      addressBook))

; (removeAddress addressBook address)
; Removes an address from the address book.
; addressBook - the address book in which the address supposedly exists.
```

```
; address     - the address to remove from the address book.
(defun removeAddress (addressBook address)
  (if (endp addressBook)
      nil
      (if (equal (car addressBook) address)
          (cdr addressBook)
          (cons (car addressBook)
                (removeAddress (cdr addressBook) address)))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; register-user.lisp
;
; Created on March 17, 2013 by Matthew A. Crist.
;
; This file will invoke the actions required to verify that a user has the
; ability to perform actions on this server.  This is used to test the
; facility of the server and does not provide direct verification for use
; of the services of the server.  (SEE isInAddressBook IN
; ADDRESS-BOOK.LISP)
;
; XML Document Format:
; ----------------------------------------------------------------------
; <?xml version="1.0"?>
; <!DOCTYPE verify SYSTEM "dtd/verify-user.dtd">
; <verify>
;    <domain>localhost</domain>
;    <name>matthew.crist</name>
;    <password>simulation</password>
;     <location>128.0.0.2</location>
; </verify>
;
; CHANGE LOG:
; ----------------------------------------------------------------------
; 2013-03-17    -   Initial conception of this file.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(in-package "ACL2")

(include-book "../../../include/io-utilities" :uncertified-okp t)
(include-book "../../../include/remote-actions" :uncertified-okp t)
(include-book "../../../include/xml-scanner" :uncertified-okp t)
(include-book "../address-book" :uncertified-okp t)

; (getDomain tokens)
; Acquires the domain of the verification request from the token list.
; tokens    -   the scanned tokens from the incoming XML verify request.
(defun getDomain (tokens)
    (if (endp tokens)
        nil
        (if (equal "<domain>" (caar tokens))
            (caadr tokens)
            (getDomain (cdr tokens)))))

; (getName tokens)
; Acquires the name of the verification request from the token list.
; tokens    -   the scanned tokens from the incoming XML verify request.
(defun getName (tokens)
    (if (endp tokens)
        nil
        (if (equal "<name>" (caar tokens))
            (caadr tokens)
            (getName (cdr tokens)))))

; (getPassword tokens)
; Acquires the password of the verification request from the token list.
; tokens    -   the scanned tokens from the incoming XML verify request.
(defun getPassword (tokens)
    (if (endp tokens)
        nil
        (if (equal "<password>" (caar tokens))
            (caadr tokens)
            (getPassword (cdr tokens)))))

; (getLocation tokens)
; Acquires the physical location of the user that is being verified for
; network connectivity on the verification process.
```

```
; tokens - the scanned tokens from the incoming XML verify request.
(defun getLocation (tokens)
    (if (endp tokens)
        nil
        (if (equal "<location>" (caar tokens))
            (caadr tokens)
            (getLocation (cdr tokens)))))

; (verifyUser user addressBook)
; Verifies that a user is present in the address-book.  This function is
; essentially the same as the isInAdressBook predicate in the address-book
; file, but is placed in a "wrapper" here for obfuscation.
(defun verifyUser (user addressBook)
    (if (endp addressBook)
        nil
        (if (equal (car addressBook) user)
            t
            (verifyUser user (cdr addressBook)))))

(set-state-ok t)
(set-ignore-ok t)

; (testUser userXML addressBookXML)
; Writes a server action file based on the result of a user being verified
; If the user is accepted, it will return a response to the client that
; verification was successful.  If not, it will write back a failure.
(defun testUser (userXML addressBookXML state)
    (let* ((userTokens (tokenizeXML userXML))
             (domain (getDomain   userTokens))
           (name   (getName     userTokens))
              (pass   (getPassword userTokens))
              (loc    (getLocation userTokens))
              (user   (list domain name pass))
              (abook  (getAddressBook (tokenizeXML addressBookXML))))
        (if (verifyUser user abook)
            (mv-let (error state)
                (string-list->file "incoming/user/verify/server-action.xml"
                      (list "<?xml version='1.0'?>"
                       "<!DOCTYPE message SYSTEM 'dtd/system-message.dtd'>"
                       "<message>"
                                 (concatenate 'string "   <action>" "ACCEPT" "</action>")
                                 (concatenate 'string "   <location>" loc "</location>")
                               "</message>")
                    state)
                (if error
                    (mv error state)
                    (mv "Action written successfully!" state)))
            (mv-let (error state)
                (string-list->file "incoming/user/verify/server-action.xml"
                      (list "<?xml version='1.0'?>"
                       "<!DOCTYPE message SYSTEM 'dtd/system-message.dtd'>"
                       "<message>"
                                 (concatenate 'string "   <action>" "REJECT" "</action>")
                                 (concatenate 'string "   <location>" loc "</location>")
                               "</message>")
                    state)
                (if error
                    (mv error state)
                    (mv "Action written successfully!" state))))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; register-user.lisp
;
; Created on February 14, 2013 by Matthew A. Crist.
;
; Purpose:
; This file contains the functions that are essential to adding a user to
; the address book.
;
; CHANGE LOG:
; ------------------------------------------------------------------------
; 2013-03-17    -   Added uncertified book ignore on load.
; 2013-03-17    -   Added the password to be added to address-book on
;                   registration.
; 2013-02-14    -   Initial conception of this file.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(in-package "ACL2")

(include-book "../../../include/io-utilities" :uncertified-okp t)
(include-book "../../../include/xml-scanner" :uncertified-okp t)
(include-book "../address-book" :uncertified-okp t)

; (getDomain tokens)
; Acquires the domain of the registration request from the token list.
; tokens    -   the scanned tokens from the incoming XML register request.
(defun getDomain (tokens)
    (if (endp tokens)
        nil
        (if (equal "<domain>" (caar tokens))
            (caadr tokens)
            (getDomain (cdr tokens)))))

; (getName tokens)
; Acquires the name of the registration request from the token list.
; tokens    -   the scanned tokens from the incoming XML register request.
(defun getName (tokens)
    (if (endp tokens)
        nil
        (if (equal "<name>" (caar tokens))
            (caadr tokens)
            (getName (cdr tokens)))))

; (getPassword tokens)
; Acquires the password of the registration request from the token list.
; tokens    -   the scanned tokens from the incoming XML register request.
(defun getPassword (tokens)
    (if (endp tokens)
        nil
        (if (equal "<password>" (caar tokens))
            (caadr tokens)
            (getPassword (cdr tokens)))))

(set-state-ok t)
(set-ignore-ok t)

; (registerUser regXML abXML state)
; Processes the information that is passed via XML string to add the user
; to the global server address book.
; regXML - The XML that is contained in the registration file - sent
;          dynamically via shell script.
; abXML  - The XML that is contained in the address book file - sent
;          dynamically via shell script.
; state  - The state of the streams in ACL2.
(defun registerUser (regXML abXML state)
    (let* ((tokens (tokenizeXML regXML))
             (domain (getDomain tokens))
             (name (getName tokens))
```

```
     (pass (getPassword tokens))
     (addressBook (getAddressBook (tokenizeXML abXML))))
(mv-let (error state)
    (string-list->file
       "store/address-book/temp_address-book.xml"
       (getAddressBookXML (addAddress addressBook (list domain name pass)))
        state)
    (if error
        (mv error state)
        (mv "Wrote temp_address-book.xml successfully" state))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; server-email.lisp
;
; Created on February 17, 2013 by Wesley R. Howell.
;
; Purpose:
; This file will execute the email transition between clients. In order
; for this to happen, the incoming email will need to be opened, then the
; to field will need to be parsed into a list. Once this list is generated
; we can then send a copy of the email message to the receipent clients.
;
; CHANGE LOG:
; 2/27/2013 - Fixed the IO issue and now we can Call an XML file and
;           Output a file from the same funciton.
;             Note! This only pull the first message in the XML file!
; 2/26/2013 - Updated references to be relative instead of absolute for
;             xml-scanner.lisp, io-utilities.lisp and list-utilities.lisp.
; 2/22/2013 - Finished the email input from file to file out
; 2/21/2013 - Worked on parsing XML documents and getting the email information
;             From the XML file
; 2/19/2013 - Moved from ./modules/user to ./modules/email sub directory.
; 2/18/2013 - Added to SVN Repo
; 2/18/2013 - Updated
; ------------------------------------------------------------------------
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(in-package "ACL2")
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Base Lines
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun break-at-set (delimiters xs)
  (if (or (not (consp xs))
          (member-equal (car xs) delimiters))
      (list '() xs)
      (let* ((first-thing (car xs))
             (break-of-rest (break-at-set delimiters (cdr xs)))
             (prefix (car break-of-rest))
             (suffix (cadr break-of-rest)))
          (list (cons first-thing prefix) suffix))))

(defun break-at (delimiter xs)
   (break-at-set (list delimiter) xs))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun getEmailXML (email)
  (if (endp email)
      nil
      (let* ((hd (concatenate 'string
                              "<?xml version='1.0'?>"
                              "<!DOCTYPE user SYSTEM '../../../dtd/messages.dtd'>"
                              "<email>"))
             (to (concatenate 'string
                              "<to>" (car (car (car email))) "@"
                              (car (cdr (car (car email))))
                              "</to>"))
             (from (concatenate 'string
                                "<from>" (car (cadr email)) "@"
                                (car(cdr (cadr email)))"</from>"))
             (sub (concatenate 'string
                               "<subject>" (cadr (cdr email)) "</subject>"))
             (msg (concatenate 'string
                               "<content>" (cadr (cdr (cdr email)))
                               "</content>"))
             (ft (concatenate 'string "</email>"
                              )))
              (list hd to from sub msg ft))))
```

```
;;getContactStructure string
;;This function will generate a contact structure
;;The delimiter right now is set at "," but can easily be changed to
;;the "@" symbol
;;string - the string to get the contact structure from
(defun getContactStructure (str)
  (let* ((chrs (coerce str 'list))
         (name (car (break-at #\@ chrs)))
         (domain (cdr (break-at #\@ chrs))))
        (list  (coerce name 'string)
               (coerce (cdr (car domain))
                       'string))))

;;getEmailStructure file
;;This function will generate the email data structure based on the
;;FIRST email message in the tokenized XML file list.
;;file - the xml file on the computer to parse
(defun getEmailStructure (xml)
  (let* (;(xml (cdr (getEmailXMLTokens file)))
         (to (car (car (cdr (cdr xml)))))
         (from (car (car (cdr (cdr (cdr (cdr (cdr xml))))))))
         (sub (car (car (cddddr  (cddddr xml)))))
         (msg (car (car (cddddr (cddddr (cdddr xml)))))))
    (list (list (getContactStructure to))
                (getContactStructure from) sub msg)))


;(getEmailStructureList xml)
;This funciton will take the email XML tags and create a list
;of email messages.
;Warning! - this requires the xml format to the exact format as stated
;on the project wiki
;xml - the Tokenized list of xml tags
(defun getEmailStructureList (xml)
      (list (getEmailStructure xml)
      )
)

;getEmail file state
;;;Entry Point;;;
;This function's goal is to open an email XML file,
;parse the xml file and output each individual message to a file.
;The funciton below, runEmailOut, should be the recusive call for this
;function
;file - the file path to the file to open
;;state - the ACL2 state
(defun getEmail (str)
   (getEmailXML (getEmailStructure str))
   )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; route-email.lisp
;
; Created on February 28, 2013 by Wesley R. Howell.
;
; Purpose:
; This file will execute the email transition between clients. In order
; for this to happen, the incoming email will need to be opened, then the
; to field will need to be parsed into a list. Once this list is generated
; we can then send a copy of the email message to the receipent clients.
;
; CHANGE LOG:
; 3/4/2013  Added 5 lines to the RWEmail function to take in a timestamp
;           and dynamically write the email to the appropriate directory
;           based on the <to> tag in the email message.
; 2/28/2013 Created this file to handled the IO and outside dependencies
;           of the server-email functions.
; ---------------------------------------------------------------------
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(in-package "ACL2")

;;Outside dependencies for this module
(include-book "../../../include/xml-scanner" :uncertified-okp t)
(include-book "../../../include/io-utilities" :uncertified-okp t)
(include-book "../server-email" :uncertified-okp t)
(set-state-ok t)

;;(writeEmail email fOut state)
;;Writes an email message based on the email data structure with
;;a single email address. Calls the getEmailXML function to generate
;;the output text.
;;email - the email data structure to output
;;fOut - the file path for the outputted file
;;state - the ACL2 state
(defun writeEmail (email fOut state)
  (mv-let (error-close state)
          (string-list->file
           fOut
           (getEmailXML email)
           state
           )
          (if error-close
              (mv error-close state)
              (mv (string-append ", output file: " fOut)
                  state)
           )))


;;;getEmailXMLTokens file
;;;This function will return the list of XML tokens for the email XML
;;;file - the filepath to the XML file
(defun getEmailXMLTokens (file state)
  (mv-let (input-xml error-open state)
          (file->string file state)
     (if error-open
         (mv error-open state)
         (mv (cdr (cdr (tokenizeXML input-xml))) state))))

;(rwEmail fin fout state)
;This function is the entry point to parse a single email message from the
;XML file "fin"
;fin - input XML file
;fout - output XML filename
;state - ACL2 state
(defun rwEmail (#|fin|#input-as-string fout state)
;  (mv-let (input-as-string error-open state)
```

```
;     (file->string fin state)
;     (if error-open
;         (mv error-open state)
         (mv-let (error-close state)
                 (string-list->file
                  (concatenate 'string "store/email/"
                               (car (cdr (getContactStructure (car(car (cddddr (tokenizeXML input-as-      ↙
    string)))))))
                                                                "/"
                               (car (getContactStructure (car(car (cddddr (tokenizeXML input-as-          ↙
    string))))))
                   ;Separate this into the structure ../domain/user
                               "/msg_"
                               fout
                               ".xml")
                                  (getEmail
                                         (cdr (cdr
                                          (tokenizeXML input-as-string)))
                                         ;These two cdr's remove the two XML
                                         ;headers for the file
                                     )
                                      state)
           (if error-close
               (mv error-close state)
               (mv "Success File has been written!"
                    state))));))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; mailing-list.lisp
; Created on February 19, 2013 by Matthew A. Crist.
;
; The data structure that will be allowed for each mailing list.  A
; mailing list defines a group of users that are subscribed to a user's
; mailing list and will be emailed en masse when the user sends an email
; to this "feed".
;
; Data Structure:
; ------------------------------------------------------------------------
; ("My Mailing List" ; The name of the mailing list
;  ; The mailing list recipients
;  (("localhost" "matthew.crist") ("localhost" "wesley.howell"))
;  ; The owner and verification of this list (for sending)
;  (("matthew.crist" "simulation")))
;
; XML File Format: (Store)
; ------------------------------------------------------------------------
; <?xml version='1.0'?>
; <!DOCTYPE mailing-list SYSTEM "../../../dtd/mailing-list.dtd">
; <mailing-list>
;    <list-name>My Mailing List</list-name>
;    <addresses>
;       <address>
;          <domain>localhost</domain>
;          <name>matthew.crist</name>
;       </address>
;       <address>
;          <domain>localhost</domain>
;          <name>wesley.howell</name>
;       </address>
;    </addresses>
;    <owners>
;       <owner>
;          <name>matthew.crist</name>
;          <password>simulation</password>
;       </owner>
;    </owners>
; </mailing-list>
;
; CHANGE LOG:
; ------------------------------------------------------------------------
; 2013-03-17    -   Actually added some code to this file.
; 2013-02-19    -   Initial file conception.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(in-package "ACL2")

; (findOwner owners owner)
; Locates an owner (by name) to ensure they are in this list.
; owners - the list of owners for a list
; owner  - the owner that will be sought.
(defun findOwner (owners owner)
    (if (endp owners)
        nil
        (if (equal (car owner) (cadar owners))
            (car owners)
            (findOwner (car owners) owner))))

; (verifyOwner owners owner)
; Predicate to determine if the owner has correct credentials to use this
; mailing list.
; owners    -   The data structure that houses the list of owners.
; owner  -  The owner that is trying to be verified.
(defun verifyOwner (mailing-list owner)
    (if (or (endp mailing-list) (endp owner))
        nil ; Can't find what isn't
```

```
        (let* ((owners (caddr mailing-list))
                ; Verify the username
                (found  (findOwner owners owner)))
            (if (equal found nil)
                nil
                ; Verify the password
                (if (equal (cadr found) (cadr owner))
                    t           ; Success!
                    nil)))))    ; Invalid credentials

; (isOwner mailing-list owner)
; Predicate to determine if the owner has the credentials to modify or
; send to the people contained in this mailing-list.
; mailing-list - the mailing list to be used.
; owner        - the credentials for log in of the owner.
(defun isOwner (mailing-list owner)
    (if (or (endp mailing-list) (endp owner))
        nil
        (let* ((owners (caddr mailing-list)))
            ; Force this check or can hack password if they know a name
            (if (equal (findOwner owners owner) nil)
                nil
                t))))

; (addOwner mailing-list owner)
; Adds an owner to the mailing list, given they do not already exist.
; mailing-list - the list in which to add the owner.
; owner        - the owner that will be added.
(defun addOwner (mailing-list owner)
    (if (or (endp mailing-list) (endp owner))
        mailing-list
        (let* ((name   (car   mailing-list))
                (subs   (cadr  mailing-list))
                (owners (caddr mailing-list))
                (found  (findOwner owners owner)))
            (if (equal found nil)
                (list name subs (append owners (list owner)))
                mailing-list))))

; (removeOwner owners owner)
; Removes an owner from the list of owner, given the name is correct.
; owners - the list of owners in which to remove the owner.
; owner  - the owner that will be removed.
(defun removeOwner (owners owner)
    (if (or (endp owners) (endp owner))
        owners
        (if (equal (caar owners) (car owner))
            (cdr owners)
            (cons (car owners) (removeOwner (cdr owners) owner)))))

; (hasOwners mailing-list)
; Verifies that this mailing list has at least one owner!  This function
; will be used to determine if the list needs to be deleted due to
; mismanagement or intentional removal for list deletion.
(defun hasOwners (mailing-list)
    (if (endp mailing-list)
        nil
        (if (endp (caddr mailing-list))
            nil
            t)))

; (changePassword mailing-list owner new)
; Changes the password for an owner of this mailing list.
; mailing-list - the list that the owner should be able to be verified by
; owner        - the old credentials for the owner
; new          - the new password for the owner
(defun changePassword (mailing-list owner new)
    (if (or (endp mailing-list) (endp owner))
```

```
        mailing-list
        (let* ((name   (car   mailing-list))
               (subs   (cadr  mailing-list))
                (owners (caddr mailing-list))
                (ok     (verifyOwner mailing-list owner)))
            (if ok
                ; Can't change what we cannot verify
                mailing-list
                (let* ((newowners (removeOwner owners owner))
                   (newowner  (list (car owner) new)))
                    (list name subs (append newowners (list newowner)))))))))

; (hasSubscribers mailing-list)
; Predicate to determine if there are recipients that exist in the list.
; mailing-list - the mailing list that will be checked for recipients.
(defun hasSubscribers (mailing-list)
    (if (endp mailing-list)
        nil
        (if (endp (cadr mailing-list))
            nil
            t)))

; (isSubscriber subscribers subscriber)
; Locates a subscriber in the list of subscribers.
; subscribers - the list of subscribers to search.
; subscriber  - the subscriber that is being sought.
(defun isSubscriber (subscribers subscriber)
    (if (endp subscribers)
        nil
        (if (equal (car subscribers) subscriber)
            t
            nil)))

; (removeSubscriber subscribers subscriber)
; Removes a subscriber from a list.
; subscribers - the list in which the subscriber will be removed.
; subscriber  - the subscriber that will be removed.
(defun removeSubscriber (subscribers subscriber)
    (if (or (endp subscribers) (endp subscriber))
        subscribers
        (if (equal (car subscribers) subscriber)
            (cdr subscribers)
            (cons (car subscribers)
                    (removeSubscriber (cdr subscribers) subscriber)))))

; (unsubscribe mailing-list subscriber)
; Removes a subscriber from a mailing list.
; mailing-list - the list in which the subscriber will be removed.
; subscriber   - the subscriber that will be removed.
(defun unsubscribe (mailing-list subscriber)
    (if (or (endp mailing-list) (endp subscriber))
        mailing-list
        (let* ((subscribers (cadr  mailing-list))
                 (name        (car   mailing-list))
             (owners      (caddr mailing-list))
                 (ok          (isSubscriber subscribers subscriber)))
            (if ok
                (let* ((newsubs (removeSubscriber subscribers subscriber)))
                    (list name newsubs owners))
                mailing-list))))

; (subscribe mailing-list subscriber)
; Adds a subscriber to a mailing list.
; mailing-list - the list in which the subscriber will be added.
; subscriber   - the subscriber that will be added to the mailing list.
(defun subscribe (mailing-list subscriber)
    (if (or (endp mailing-list) (endp subscriber))
        mailing-list
```

```
        (let* ((subscribers (cadr  mailing-list))
               (name        (car   mailing-list))
                (owners      (caddr mailing-list))
                (subbed      (isSubscriber subscribers subscriber)))
            (if subbed
                mailing-list
                (list name (append subscribers (list subscriber)) owners)))))

; (renameMailingList mailing-list new-name owner)
; Renames a mailing list, given the user has the credential to do so.
; mailing-list - the mailing list that will be renamed.
; new-name     - the new name of the mailing list.
; owner        - the credentials to verify the user has access.
(defun renameMailingList (mailing-list new-name owner)
    (if (or (endp mailing-list) (endp owner))
        mailing-list
        (let* ((subscribers (cadr  mailing-list))
                 (owners      (caddr mailing-list))
                 (ok          (verifyOwner mailing-list owner)))
            (if ok
                (list new-name subscribers owners)
                mailing-list))))
```

Appendix B: ACL2 Test and Theorem Files

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; address-book_tests.lisp                                                      ;
; Created by Matthew A. Crist on April 7, 2013 (augmented)                     ;
; This file holds the tests for the logic on the address-book data             ;
; structure.  This file should be tested using the Racket IDE with             ;
; dracula.                                                                     ;
;                                                                              ;
; CHANGE LOG:                                                                  ;
; ---------------------------------------------------------------------------- ;
; 2013-04-07 - Added heading section for this file.  File was created          ;
;              previously, but was never formalized in documentation.          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(include-book "address-book")
(include-book "doublecheck" :dir :teachpacks)
(include-book "testing" :dir :teachpacks)

; Theorem:
; If the address is in the address book, then adding the address to
; the address book results in the same address book structure.
;(defthm address-is-in-book-adding-returns-original-book-thm
;  (implies (and (listp aB)
;                (isInAddressBook aB a))
;           (equal aB (addAddress aB a)))
;  :rule-classes (:rewrite :forward-chaining))

; Test on above theorem
(defproperty address-is-in-book-adding-returns-original-book-tst
  (domain      :where (stringp domain)
               :value (random-string)
   name        :where (stringp name)
               :value (random-string)
   pass        :where (stringp pass)
               :value (random-string)
   address     :where (listp address)
               :value (list domain name pass)
   addressBook :where (listp addressBook)
               :value (list address))
  (implies (isInAddressBook addressBook address)
           (equal addressBook (addAddress addressBook address)))
  :rule-classes (:rewrite :forward-chaining))

(check-properties)

; Check Expects on above theorem
(check-expect (addAddress '((1 2 3)) '(1 2 3)) '((1 2 3)))

; Theorem:
; If the address is not in the address book, then adding the address
; to the address book will return an address book with the length of
; the original address book + 1. (admits)
;(defthm address-is-not-in-book-add-length-thm
;  (implies (and (listp addressBook)
;                (not (isInAddressBook addressBook address)))
;           (equal (+ (length addressBook) 1)
;                  (length (addAddress addressBook address))))
;  :rule-classes (:rewrite :forward-chaining))

; Test on the above theorem
(defproperty address-is-not-in-book-add-length-tst
  (domain      :where (stringp domain)
               :value (random-string)
   name        :where (stringp name)
               :value (random-string)
   pass        :where (stringp pass)
               :value (random-string)
   domain-not  :where (stringp domain-not)
               :value (random-string)
```

```
   name-not    :where (stringp name-not)
               :value (random-string)
   pass-not    :where (stringp pass-not)
               :value (random-string)
   address     :where (listp address)
               :value (list domain name pass)
   address-not :where (listp address-not)
               :value (list domain-not name-not pass-not)
   addressBook :where (listp addressBook)
               :value (list address))
  (implies (not (isInAddressBook addressBook address-not))
           (equal (+ (length addressBook) 1)
                  (length (addAddress addressBook address-not))))
  :rule-classes (:rewrite :forward-chaining))

(check-properties)

; Check-expects on the above theorem
(check-expect (length (addAddress '((1 2 3)) '(4 5 6))) 2)


; Theorem:
; If the address is in the address book, then removing the address from
; the address book will return an address book with the length of the
; original address book - 1.
;(defthm address-is-in-book-remove-length-thm
;  (implies (and (listp addressBook)
;                (not (equal address nil))
;                (not (equal (length addressBook) 0))
;                (isInAddressBook addressBook address))
;           (equal (- (length addressBook) 1)
;                  (length (removeAddress addressBook address))))
;  :rule-classes (:rewrite :forward-chaining))


; Tests on the above theorem
(defproperty address-is-in-book-remove-length-tst
  (domain      :where (stringp domain)
               :value (random-string)
   name        :where (stringp name)
               :value (random-string)
   pass        :where (stringp pass)
               :value (random-string)
   address     :where (listp address)
               :value (list domain name pass)
   addressBook :where (listp addressBook)
               :value (list address))
  (implies (isInAddressBook addressBook address)
           (equal (- (length addressBook) 1)
                  (length (removeAddress addressBook address))))
  :rule-classes (:rewrite :forward-chaining))

(check-properties)

; Check-Expects on above example
(check-expect (length (removeAddress '((1 2 3) (4 5 6)) '(1 2 3))) 1)

; Theorem:
; If the address is not in the address book, then remove the address from
; the address book results in the original address book (lengths are
; equivalent).
;(defthm address-is-not-in-address-book-remove-returns-original-book-thm
;  (implies (and (listp addressBook)
;            (not (isInAddressbook addressBook address)))
;           (equal (length addressBook)
;                          (length (removeAddress addressBook address))))
;  :rule-classes (:rewrite :forward-chaining))
```

```
; Tests on the above theorem
(defproperty address-is-not-in-address-book-remove-returns-original-book-tst
  (domain      :where (stringp domain)
               :value (random-string)
   name        :where (stringp name)
               :value (random-string)
   pass        :where (stringp pass)
               :value (random-string)
   domain-not  :where (stringp domain-not)
               :value (random-string)
   name-not    :where (stringp name-not)
               :value (random-string)
   pass-not    :where (stringp pass-not)
               :value (random-string)
   address     :where (listp address)
               :value (list domain name pass)
   address-not :where (listp address-not)
               :value (list domain-not name-not pass-not)
   addressBook :where (listp addressBook)
               :value (list address))
  (implies (not (isInAddressbook addressBook address-not))
           (equal (length addressBook)
                  (length (removeAddress addressBook address-not))))
  :rule-classes (:rewrite :forward-chaining))

(check-properties)

; Check expect on the above test
(check-expect (length (removeAddress '((1 2 3)) '(4 5 6))) 1)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;server-email-test.lisp
;
;Created By: Wesley R. Howell
;
;This file will contain the Theorems and test for the server-email logic
;definitions in the server-email.lisp file
;
;--Change Log-------------------------------------------------------------
;
; 2/28/2013 - File added to SVN and initial theorems proven
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(include-book "server-email")

;(contactStructure-not-nil)
;Theorem to prove that a returned contatct structure cannot be nil
;if a correct string is passed into the function
(defthm contactStructure-not-nil
  (implies (stringp str)
           (equal (listp (getContactStructure str))
                        t)))

;(contactStructure-contents-are-strings)
;Theorem to prove that the contents of the atoms in a contact structure are
;strings
(defthm contactStrucutre-contents-are-strings
  (implies (stringp str)
           (AND(equal (stringp (car (getContactStructure str))) t)
               (equal (stringp (car (cdr (getContactStructure str)))) t))
               ))

;(contactStructure-domain-is-a-string)
;Theorem to prove that the contact structure's name is a string, the above
;Theorem contactStructure-contents-are-strings builds off this proof.
(defthm contactStructure-domain-is-a-string
  (implies (stringp str)
           (equal (stringp (car (cdr (getContactStructure str))))
                  t)))

;(emailStructure-is-a-list)
;Theorem to prove that the emailStructure is returned from the function
;getEmailStructure
(defthm emailStrucutre-is-a-list
  (implies (listp xml)
           (equal (listp (getEmailStructure xml)) t)))

;(emailStrucutre-ToField-a-list
;Theorem to prove the email's to structure is a list based on the
;data structure based on the project documentation.
(defthm emailStructure-ToField-a-list
  (implies (listp xml)
           (equal (listp (car (getEmailStructure xml)))
                  t)))

;(getEmail-takes-list-returns-strings-on-header)
;Theorem to prove that the getEmail functions takes a list and returns
;a list of strings in XML Format.
(defthm getEmail-Takes-list-returns-strings-on-header
  (implies (listp str)
           (equal (stringp
                   (car(getEmail str)))
                  t)))

;(getEmail-returns-list-of-strings)
;Theorem built of the previous theorem that shows that each element in the
;list is a string.
```

```
(defthm getEmail-Returns-list-of-strings
  (implies (listp str)
           (AND (equal (listp (getEmail str)) t)
                (equal (stringp (car (getEmail str))) t))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;client-email-test.lisp
;
;Created By: Wesley R. Howell
;
;This file will contain the Theorems and test for the client-email logic
;definitions in the client-email.lisp file
;
;--Change Log-------------------------------------------------------------
;
; 3/25/2013 - Finished Theorems and Proofs. All Proofs Admit
; 3/14/2013 - File added to SVN and initial theorems proven
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(include-book "client-email")

;(contactStructure-not-nil)
;Theorem to prove that a returned contatct structure cannot be nil
;if a correct string is passed into the function
(defthm contactStructure-not-nil
  (implies (stringp str)
           (equal (listp (getContactStructure str))
                        t)))

;(contactStructure-contents-are-strings)
;Theorem to prove that the contents of the atoms in a contact structure are
;strings
(defthm contactStrucutre-contents-are-strings
  (implies (stringp str)
           (AND(equal (stringp (car (getContactStructure str))) t)
               (equal (stringp (car (cdr (getContactStructure str)))) t))
               ))

;(contactStructure-domain-is-a-string)
;Theorem to prove that the contact structure's name is a string, the above
;Theorem contactStructure-contents-are-strings builds off this proof.
(defthm contactStructure-domain-is-a-string
  (implies (stringp str)
           (equal (stringp (car (cdr (getContactStructure str))))
                  t)))

;(emailStructure-is-a-list)
;Theorem to prove that the emailStructure is returned from the function
;getEmailStructure
(defthm emailStrucutre-is-a-list
  (implies (listp xml)
           (equal (listp (getEmailStructure xml)) t)))

;(emailStrucutre-ToField-a-list
;Theorem to prove the email's to structure is a list based on the
;data structure based on the project documentation.
(defthm emailStructure-ToField-a-list
  (implies (listp xml)
           (equal (listp (car (getEmailStructure xml)))
                  t)))


;(toStringTakesListAndConvertsToString
;Theorm to test conservation of string properties for this
;function, similar to the chrs->string in the list utils
(defthm toStringTakesListAndConvertsToString
  (implies (listp toString)
           (stringp (splitToString toString)
                    )
           )
  )
```

```
;(splitToStructisList)
;Theorem to prove that the item returned is a list
;of strings that is not empty
(defthm splitToStructureList
  (implies (listp toList)
           (AND
            (listp (splitToStruct toList))
            (stringp (car (splitToStruct toList))))
   )
)


;(splitToFieldReturnsListofTos)
;Theorem to show that the splitToField function takes
;The inputed email addresses and parses the string into
;a list of valid email addresses (as strings).
;The returned type will be the same as the above theorem
;Note: The above theorm is encapsulated here as well.
(defthm splitToFieldReturensListofTos
  (implies (stringp toString)
           (AND (listp
                 (splitToField toString))
                (stringp
                 (car (splitToField toString))))
           ))


;(genEmailProducesMessage)
;Theorm to test that the email structure
;returned from generateEmailFromStrings is the correct
;email structure ((list) str str str)
(defthm genEmailProducesMessage
  (implies (AND (stringp msg)
                (AND (stringp sub)
                (AND (stringp to)
                (stringp from))))
                (AND (listp
                 (generateEmailFromStrings to from sub msg))
                (listp
                 (car (generateEmailFromStrings to from sub msg))))
           ))


;(multRecipProducesXMLList)
;Theorm to prove that the output is correct on multRecip
;based on the toList
;The format will be (XML_1, XML_2, ... , XML_n) where XML
;is the prepared string format from the getEmailXML function.
(defthm multRecipProducesXMLList
  (implies (AND (listp toList)
                (AND (stringp from)
                     (AND (stringp sub)
                          (stringp msg))))
           (AND (listp (multRecip toList from sub msg))
                (stringp (car
                          (multRecip toList from sub msg))))))


;(emailEntryPoint)
;Tests the email function since it is the entry point for the client
;side test. This theorem will need to prove that the subsequent calls
;to the email functions returns the correctly formatted list in order.
;for the IO module to output correctly. Hence, we need to prove the same steps
;as in multRecip with the extra constraint that "to" is a string.
(defthm emailEntryPoint
  (implies (AND (stringp to)
                (AND (stringp from)
```

```
                    (AND (stringp sub)
                         (stringp msg))))
          (AND (listp (email toList from sub msg))
               (stringp (car
                         (email to from sub msg))))))


;(emailTextReturnsListofString)
;Test that the email text returns a list built strings for either,
;console display or output.
(defthm emailTextReturnsListofString
  (implies (listp tokxml)
           (AND (listp (getEmailText tokxml) )
                (stringp (car (getEmailText tokxml))))))


;(emailHTMLReturnsListofStrings)
;Test that the email html output is a correct list for the IO
;funciton. This will be the pre
(defthm emailHTMLReturnsListofString
  (implies (listp tokxml)
           (AND (listp (getEmailHTML tokxml) )
                (stringp (car (getEmailHTML tokxml))))))
```

Appendix C: Java Source Code Files

```
package modules.email.action;

import java.io.*;
import java.util.*;
import java.net.*;

public class GetEmail {

    public final static String OUTPATH = "store/email/outbox/";
    public final static String INPATH = "incoming/email";

    public static void getEmail (String name, String domain, String password){
        //Get Email Messages from Server, Save to incoming folder

        Socket server = null;
        BufferedWriter out = null;
        BufferedReader in = null;

        try {
            System.out.println("Opening socket...");
            server = new Socket("localhost", 20002);
            System.out.println("Connection successful!");
            out = new BufferedWriter(new OutputStreamWriter(server.getOutputStream()));
            in = new BufferedReader(new InputStreamReader(server.getInputStream()));

            System.out.println("Sending login information to the server.");

            if(server.isConnected()) {
                out.write("<?xml version='1.0'?>");
                out.write("<verify>");
                out.write("<domain>"+domain+"</domain>");
                out.write("<name>"+name+"</name>");
                out.write("<password>"+password+"</password>");
                out.write("</verify>");
                out.newLine();
                out.flush();    // flush should write
                server.shutdownOutput();

                int count = 1;
                String response = "";
                while(!(response = in.readLine()).contains("END")) {
                    if(response != null){
                        System.out.println(response);
                    try{
                        FileWriter fstream = new FileWriter("incoming/email/msg_"+count+".xml");
                        BufferedWriter fout = new BufferedWriter(fstream);
                        fout.write(response);
                        fout.close();
                        count++;
                    } catch (Exception e){
                        System.err.println("Error: "+e.getMessage());
                    }
                    }

                }   // end while loop
            } else {
                System.out.println("Connection with server could not be established.");
            }   // end if-else

            //out.close();
            in.close();
            server.close();

        } catch(Exception e) {
            e.printStackTrace();
        }

       try{
```

```
            Thread.sleep(2000);
        } catch (InterruptedException e){
             e.printStackTrace();
        }

         //Parse the imported files Originally found in the shell script



        int datecnt = 1;
        File folder = new File(INPATH);
        folder.mkdirs();
        File[] listOfFiles = folder.listFiles();
        for (File f : listOfFiles){
             if(f.isFile() && !f.isHidden()){
                 //Build the ACL2 script
                 String unique = (new Date().toString()) +"_" + datecnt;
                 unique = unique.replace(' ', '_');
                 unique = unique.replace(':', '_');
                 String script = "(in-package \"ACL2\")(include-book \"modules/email/action/rw-email\"" +
                         " :uncertified-okp t) (readEmail \"incoming/email/"+f.getName()+"\" \""+unique+"\" ↙
    state)";

                 try{
                 //Run on ACL2
                 // Initialize ACL2 and dump its output to the log
                 System.out.println("Executing ACL2 runtime for Email Generation...");
                 ProcessBuilder processBuilder = new ProcessBuilder("acl2");
                 File log = new File("logs/acl2_log.txt");
                 processBuilder.redirectErrorStream(true);
                 processBuilder.redirectOutput(log);

                 Process process;

                 process = processBuilder.start();

                 PrintWriter procIn = new PrintWriter(process.getOutputStream());

                 // Write the ACL2 to the process, close ACL2
                 procIn.println(script);
                 procIn.println("(good-bye)");
                 procIn.flush();
                 procIn.close();

                 } catch(IOException e) {
                     e.printStackTrace();
                 }
             }

                try{
                    Thread.sleep(3000);
                } catch (InterruptedException e){
                     e.printStackTrace();
                }

             f.delete();
             datecnt ++;
        }



    }
}
```

```
package modules.email.action;

import java.io.*;
import java.util.*;
import java.net.*;


/**
 * This file will replace the send-message.sh shell script and be included in the ACL2 email client
 * package.
 * @author Wesley R. Howell
 *
 * Original header:
 * #Created By: Wesley R. Howell
 * #Shell Script to Process an email message
 * #invocation ./route-email.sh "inputFile"
 * where inputFile is the email message in the
 * incoming/email folder on the server
 *
 */

public class SendEmail {

        public final static String OUTPATH = "store/email/outbox/";

    /**
     * Function that replaces Shell Script for Sending Emails.
     * @param to
     * @param from
     * @param sub
     * @param msg
     */
    public static void sendMessage(String to, String from, String sub, String msg){
        //Build the ACL2 script
        String script = "(in-package \"ACL2\")(include-book \"modules/email/action/rw-email\"" +
                " :uncertified-okp t) (writeMessage \""+to+"\" \""+from+"\" \""+sub+"\" \""+msg+"\" state)" ↙
    ;

        try{
        //Run on ACL2
        // Initialize ACL2 and dump its output to the log
        System.out.println("Executing ACL2 runtime for Email Generation...");
        ProcessBuilder processBuilder = new ProcessBuilder("acl2");
        File log = new File("logs/acl2_log.txt");
        processBuilder.redirectErrorStream(true);
        processBuilder.redirectOutput(log);

        Process process;

        process = processBuilder.start();

        PrintWriter procIn = new PrintWriter(process.getOutputStream());

        // Write the ACL2 to the process, close ACL2
        procIn.println(script);
        procIn.println("(good-bye)");
        procIn.flush();
        procIn.close();

        } catch(IOException e) {
            e.printStackTrace();
        }

        try {
            Thread.sleep(4000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
```

```
}
//Split Emails with new Java file
File folder = new File(OUTPATH);
File[] listOfFiles = folder.listFiles();


for (File f : listOfFiles){
    if(f.isFile() && !f.isHidden()){
        //Open the file and get contents.
        ArrayList<String> contents = new ArrayList<String>();
        StringBuilder strb = null;
        BufferedReader reader = null;

        try {
            //Use the string builder to append each email to the string,
            //Add the string to the list when it reaches the end
            //Start a new string builder for each new email in the file.
            reader = new BufferedReader(new FileReader (f));
            String line = null;

            try {
                while ((line = reader.readLine()) != null){
                    if(line.contains("<?xml version")){
                        if(strb != null){
                            contents.add(strb.toString());
                        }
                        strb = new StringBuilder();
                    }
                    strb.append(line);
                }
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        contents.add(strb.toString());

        //Remove the file since we do not need it anymore.
        f.delete();

        int count = 1;

        for(String str : contents){

            /*
              We will now send each email to the server
            */

            Socket server = null;
            PrintWriter out = null;
            BufferedReader in = null;

            try {
                System.out.println("Opening socket...");
                server = new Socket("localhost", 20005);
                System.out.println("Connection successful!");
                out = new PrintWriter(server.getOutputStream(), true);
                in = new BufferedReader(new InputStreamReader(server.getInputStream()));

                out.println(str);

                out.flush();
                out.close();
```

```
                        in.close();
                        server.close();

        } catch(Exception e) {
            e.printStackTrace();
        }
                }
            }

        }


    }

}
```

```
package modules.user.register;

import java.io.*;
import java.util.*;
import java.net.*;

/**
 * Send Request.java
 * This file replaces the send request shell scripts. This file will call
 * The ACL2 functions that generate the XML request to send to the server.
 * Once a request has been made by ACL2, we will loop through all the request
 * and send to the Server using port 20001
 * Original Author Adam Ghodratnama
 * @author Wesley R. Howell
 *
 */

public class SendRequest {

    public final static String OUTPATH = "store/user/requests/register/";
    public final static String INPATH = "incoming/email";

    public static void sendRequest (String name, String domain, String password){
        String unique = (new Date().toString()) +"_1";
        unique.replace(' ', '_');
        unique.replace(':', '_');
        //Create request using ACL2
        String script = "(in-package \"ACL2\")(include-book \"modules/user/register/create-user-request\"" ↙
    +
                " :uncertified-okp t) (createRequest '(\""+domain+"\" \""+name+"\" \""+password+"\") \""+  ↙
    unique+"\" state)";

        try{
        //Run on ACL2
        // Initialize ACL2 and dump its output to the log
        System.out.println("Executing ACL2 runtime for Email Generation...");
        ProcessBuilder processBuilder = new ProcessBuilder("acl2");
        File log = new File("logs/acl2_log_request.txt");
        processBuilder.redirectErrorStream(true);
        processBuilder.redirectOutput(log);

        Process process;

        process = processBuilder.start();

        PrintWriter procIn = new PrintWriter(process.getOutputStream());

        // Write the ACL2 to the process, close ACL2
        procIn.println(script);
        procIn.println("(good-bye)");
        procIn.flush();
        procIn.close();

        } catch(IOException e) {
            e.printStackTrace();
        }


       try{
           Thread.sleep(3000);
       } catch (InterruptedException e){
            e.printStackTrace();
       }


        //Send to the server.
        Socket server = null;
        PrintWriter out = null;
```

```
        BufferedReader in = null;

        try {
            System.out.println("Opening socket...");
            server = new Socket("localhost", 20003);
            System.out.println("Connection successful!");
            out = new PrintWriter(server.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(server.getInputStream()));

            //Open directory and get request and send it.

            File folder = new File(OUTPATH);
            File[] listOfFiles = folder.listFiles();

            for (File f : listOfFiles){
                if(f.isFile() && !f.isHidden()){
                    BufferedReader reader = null;

                    try {
                        reader = new BufferedReader(new FileReader (f));
                        String line = null;
                        try {
                            while ((line = reader.readLine()) != null){
                                System.out.println(line);
                                out.println(line);
                            }
                        } catch (IOException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                        }

                    } catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                }
            }


            out.flush();
            out.close();
            in.close();
            server.close();

        } catch(Exception e) {
            e.printStackTrace();
        }

    }
}
```

```
/**
 * RegisterUser.java
 * Created by Matthew A. Crist on March 29, 2013.
 *
 * This class is a revamp of the old shell scripts to allow for better
 * server integration.
 *
################################################################################
# Created by Matthew A. Crist on March 8, 2013.
# This file will provide the actions required to register a user to the address
# book.  Folders will be created for a store.
#
# Server listening port: 20001
#
# CHANGE LOG:
#-------------------------------------------------------------------------------
# 2013-03-29    -   Migrated file into java source.
# 2013-03-17    -   Added directory creation on user registration.
# 2013-03-09    -   Removed grep validation for request structure added if check
# 2013-03-09    -   Added grep validation for request structure
# 2013-03-09    -   Encapsulated all values in function register_user
# 2013-03-08    -   Added support for network connectivity.
# 2013-03-08    -   Initial conception of this file.
################################################################################*/

package modules.user.register;

import java.io.*;
import java.util.*;
import java.net.*;
import java.nio.channels.FileChannel;

public class RegisterUser {
    public static void main(String[] args) {
        boolean listening = true;

        try {
            // Acquire the listening port for connection to client.
            ServerSocket server = new ServerSocket(20003);

            while(listening) {
                // Wait until the client connects
                Socket client = server.accept();

                // Handles for input and output streams relating to the socket connection
                PrintWriter out = new PrintWriter(client.getOutputStream(), true);
                BufferedReader in = new BufferedReader(new InputStreamReader(client.getInputStream()));

                // Buffers
                String input, store="", request="";

                // Read the input from the connection
                while((input = in.readLine()) != null) {
                    System.out.println(input);
                    request += input;
                }   // end while

                // Read the contents of the address-book currently stored
                BufferedReader reader = new BufferedReader(new FileReader("store/address-book/address-book. ↙
    xml"));
                while((input = reader.readLine()) != null) {
                    store += input;
                }   // end while

                // The ACL2 code to execute.
                String acl2 = "(include-book \"modules/user/register/register-user\")" +
                              "(in-package \"ACL2\")"            +
                              "(registerUser \"" + request + "\" \"" + store + "\" state)";
```

```
            // Initialize ACL2 and dump its output to the log
            System.out.println("Executing ACL2 runtime for RegisterUser...");
            ProcessBuilder processBuilder = new ProcessBuilder("acl2");
            File log = new File("logs/user/register/acl2_log.txt");
            processBuilder.redirectErrorStream(true);
            processBuilder.redirectOutput(log);

            Process process = processBuilder.start();
            PrintWriter procIn = new PrintWriter(process.getOutputStream());

            // Write the ACL2 to the process, close ACL2
            procIn.println(acl2);
            procIn.println("(good-bye)");
            procIn.flush();
            procIn.close();
            System.out.println("FinishedACL2");
            // Old store is old address-book file and new store is newly generated
            File oldStore = new File("store/address-book/address-book.xml");
            File newStore = new File("store/address-book/temp_address-book.xml");

            // Response header information
            String response = "<?xml version='1.0'?>"                        +
                              "<!DOCTYPE response SYSTEM 'dtd/reponse.dtd'>" +
                              "<response>";

            //System.out.println("Old Store Size: " + oldStore.length());
            //System.out.println("New Store Size: " + newStore.length());

            // Determine if there was a change.
            // If entry was added, the length > that old length.
            if(oldStore.length() < newStore.length()) {
                // Replace old file with new file
                FileChannel src = new FileInputStream(newStore).getChannel();
                FileChannel dest = new FileOutputStream(oldStore).getChannel();
                dest.transferFrom(src, 0, src.size());
                src.close();
                dest.close();

                // Extract data from request XML
                String name   = request.substring(request.indexOf("<name>")+6, request.indexOf("</name> ↙
"));
                String domain = request.substring(request.indexOf("<domain>")+8, request.indexOf("</    ↙
domain>"));

                // Create the store directory for the user's emails
                File storeDirectory = new File("store/email/" + domain + "/" + name + "/");
                storeDirectory.mkdirs();

                response += "<message>ACCEPT</message>";
            } else {
                // Remove new file as it is pointless
                newStore.delete();
                response += "<message>REJECT</message>";
            }   // end if-else

            response += "</response>";

            // Writeback the response to the client
            //out.print(response);
            out.flush();

            // Close all streams
            out.close();
            in.close();
            client.close();
        }   // end while
    } catch(Exception e) {
```

```
            e.printStackTrace();
        }   // end try/catch
    }   // end function main
}   // end class RegisterUser
```

```
/**
 * SendEmail.java
 * Created on March 31, 2013 by Matthew A. Crist.
 *
 * This is an updated module from old shell scripts to the new Java method.

###############################################################################
# Created By: Wesley R. Howell
# Modified by: Matthew A. Crist (2013-03-08)
# Shell Script to Process an email message
#
# invocation: ./route-email.sh "inputFile"
#       inputFile - is the email message in the incoming/email folder on the
#                 server.
#
# Server listening port: 20005
#
# CHANGE LOG:
#------------------------------------------------------------------------------
# 2013-03-31    -   Modified file to use new Java invocation.
# 2013-03-23    -   Changed path to module from send-email to send.
# 2013-03-08    -   Added network listening capabilities to script and changed
#                 loop to infinite loop until kill command receive as well as
#                 removed the need for multiple files.
###############################################################################

 */

package modules.email.send;

import java.io.*;
import java.net.*;
import java.util.*;

class SendEmail {
    public static void main(String[] args) {
        boolean listening = true;

        try {
            // Listen for connection via server port 20005
            ServerSocket server = new ServerSocket(20005);

            while(listening) {
                // Wait until the client connects
                Socket client = server.accept();

                // Handles for input and output streams relating to the socket connection
                PrintWriter out = new PrintWriter(client.getOutputStream(), true);
                BufferedReader in = new BufferedReader(new InputStreamReader(client.getInputStream()));

                // Buffers
                String input, store="", request="";

                // Read the input from the connection
                while((input = in.readLine()) != null) {
                    request += input;
                }   // end while

                // Write the input out to a temporary file for shifting.
                String fileName = (new Date()).toString();
                // Since I do not like whitespace and : will make the console go apeshit
                fileName = fileName.replace(':', '_');
                fileName = fileName.replace(' ', '_');

                // The ACL2 string that will be executed.
                String acl2 = "(in-package \"ACL2\")" +
                              "(include-book \"modules/email/send/route-email\" :uncertified-okp t)" +
                              "(rwEmail \"" + request + "\" \"" + fileName + "\" state)";
```

```
                // Initialize ACL2 and dump its output to the log
                System.out.println("Executing ACL2 runtime for SendEmail...");
                ProcessBuilder processBuilder = new ProcessBuilder("acl2");
                File log = new File("logs/email/send/acl2_log.txt");
                processBuilder.redirectErrorStream(true);
                processBuilder.redirectOutput(log);

                Process process = processBuilder.start();
                PrintWriter procIn = new PrintWriter(process.getOutputStream());

                // Write the ACL2 to the process, close ACL2
                procIn.println(acl2);
                procIn.println("(good-bye)");
                procIn.flush();
                procIn.close();
            }   // end while loop
        } catch(Exception e) {
            e.printStackTrace();
        }   // end try-catch
    }   // end function main
}   // end class SendEmail
```

```
/**
  * VerifyUser.java
  * Created by Matthew A. Crist on March 28, 2013.
  * This file is designed to replace the verify-user.sh script for more flexibility.
  * PREVIOUS DOCUMENTATION:
   # verify-user.sh
        # Created on March 23, 2013 by Matthew A. Crist.
        # This file will invoke the script required to verify that a user should
        # have access to an inbox and open a connection to send those files to the
        # client.
        #
        # NO LONGER VALID vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
        # THIS MODULE RELIES HEAVILY ON THE CREATED DIRECTORIES FOR MAIL FOLDERS
        # ON USER REGISTRATION.  IF THIS FOLDER DOES NOT EXIST, THE VERIFICATION
        # PROCESS IS POINTLESS.  A RESPONSE WILL BE USED ON THE SERVER SIDE
        # ACCEPT(user.verify) or REJECT(user.verify) TO RESPOND BACK TO THE CLIENT
        # IF IT IS ACCEPTABLE TO SEND/RECIEVE THE EMAIL IN THEIR INBOX.  WHEN THAT
        # EMAIL IS SENT, IT IS REMOVED FROM THE SERVER ENTIRELY.
        # ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
        #
        # CHANGE LOG:
        # ------------------------------------------------------------------------
        # 2013-03-31    -       Changed the purpose of the script to acquire emails.
        # 2013-03-29    -       Converted file from shell script to java program.
        # 2013-03-23    -       Initial conception of this file.
*/

package modules.user.verify;

import java.io.*;
import java.util.*;
import java.net.*;
import lib.*;
```

```
public class VerifyUser {
        public static void main(String[] args) {
                boolean listening = true;

                try {
                        ServerSocket server = new ServerSocket(20002);

                        while(listening) {
                                System.out.println("User verification bound to post 20002.\n");
                                Socket client = server.accept();
                                System.out.println("User verification request accepted.  Processing...");

                                BufferedWriter out = new BufferedWriter(new OutputStreamWriter(client.
    getOutputStream()));
                                BufferedReader in  = new BufferedReader(new InputStreamReader(client.
    getInputStream()));

                                String input, store = "", request = "";

                                // For all input received, write it to the request buffer.
                                while((input = in.readLine()) != null) {
                                        request += input;
                                }       // end while loop

                                // Need to acquire the contents of the address book
                                BufferedReader reader = new BufferedReader(new FileReader("store/address-
    book/address-book.xml"));
                                while((input = reader.readLine()) != null) {
                                        store += input;
                                }       // end while loop

                                // The ACL2 command that will be executed.
                                String acl2 = "(include-book \"modules/user/verify/verify-user\")" +
                                                                        "(in-package \"ACL2\")" +
                                         "(set-state-ok t)" +
                                                                        "(set-guard-checking :none)" +
```

```
                                        "(testUser \"" + request + "\" \"" + store + "\" state)";

                                // Proceed to spur the ACL2 process and place a wrapper on the IO
                                System.out.println("Executing ACL2 runtime for User Verification...");
                                ProcessBuilder processBuilder = new ProcessBuilder("acl2");
                                File log = new File("logs/user/verify/acl2_log.txt");
                                processBuilder.redirectErrorStream(true);
                                processBuilder.redirectOutput(log);
                                Process process = processBuilder.start();
                                PrintWriter procIn = new PrintWriter(process.getOutputStream());

                                // Write the ACL2 to the process, exit ACL2 and close the socket
                                procIn.println(acl2);
                                procIn.println("(good-bye)");
                                procIn.flush();
                                procIn.close();

                                // Flag for the security check
                                boolean proceed = false;

                                // Read in the contents of the file and see if one line contains ACCEPT
                                BufferedReader tRead = new BufferedReader(new FileReader("incoming/user/    ↙
verify/server-action.xml"));
                                String failBuffer = "";

                                System.out.println("Determining if login information is correct.");

                                while((input = tRead.readLine()) != null) {
                                        // Because I am lazy and don't want to parse the XML
                                        if(input.contains("ACCEPT")) {
                                                proceed = true;
                                                System.out.println("User verified successfully!");
                                        } else {
                                                failBuffer += request;
```

```
                                        }       // end if-else
                                }       // end while

                                tRead.close();

                                // Determine if the login ws good!
                                if(proceed) {
                                        String name   = request.substring(request.indexOf("<name>")+6,      ↙
request.indexOf("</name>"));
                                        String domain = request.substring(request.indexOf("<domain>")+8,    ↙
request.indexOf("</domain>"));

                                        File emailDirectory = new File("store/email/" + domain + "/" + name ↙
 + "/");

                                        System.out.println("Sending emails from " + emailDirectory.getPath  ↙
());

                                        // It better be a damn directory, but incase someone has leet hacks
                                        if(emailDirectory.isDirectory()) {
                                                File[] emails = emailDirectory.listFiles();

                                                String transmit = "";

                                                System.out.println("Writing emails to client.");
                                                // Read the contents of each email and transmit them to the ↙
 client.
                                                for(int i = 0; i < emails.length; i++) {
                                                   if(!emails[i].isHidden()){
                                                           BufferedReader eRead = new BufferedReader(new    ↙
FileReader(emails[i]));
                                                           String eTmp = "";
                                                           while((eTmp = eRead.readLine()) != null) {
                                                                   transmit += eTmp;
                                                           }    // end while

                                                           eRead.close();
```

```
                                        // Write email to client
                                        out.write(transmit);
                                        out.newLine();
                                        // Reset the buffer
                                        transmit = "";
                                    }    // end if
                                }       // end for

                                out.write("END");
                            } else {
                                // Create the directory since it should be there!!!
                                emailDirectory.mkdirs();

                                System.out.println("There was an internal server error:  ↙
Inbox does not exist!\n");
                                out.write("END");
                            }       // end if-else
                        } else {
                            out.write(failBuffer);
                            out.newLine();
                            out.write("END");
                        }       // end if-else

                        // Close our connections
                        out.close();
                        in.close();
                        client.close();
                    }       // end while loop

                    server.close();
                    System.exit(0);
                } catch(Exception e) {
                    System.out.println(e.getMessage());
                    e.printStackTrace();
```

```
                }       // end try/catch

        }       // end function main

}       // end class VerifyUser
```

```
import java.awt.Image;
import java.awt.Toolkit;
import javax.swing.*;


public class ACL2Email {

        /**
         * main.java
         * Entry point for ACL2 email window.
         * @author Wesley R. Howell
         * @param args
         */

        public static void main(String[] args) {



                emailWindow email = new emailWindow();
                email.setSize(1024, 800);
                Image icon = Toolkit.getDefaultToolkit().getImage("icon.gif");
                email.setIconImage(icon);
                email.setVisible(true);
        }

}
```

```
import modules.email.action.*;
import modules.user.register.*;
import javax.swing.*;

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

import java.io.InputStreamReader;import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;

import javax.swing.border.EtchedBorder;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.Toolkit;
import java.awt.Color;


/**
 * email Window
 * This file will generate a window for the ACL2 email system. This is a client based window and
 * will call the shell scripts we generated for the ACL2 program
 *
 * mainly, this is a fancy way of entering data instead of using the console and navigating folders.
 *
 * The Icons in this applications are open source obtained from http://openiconlibrary.sourceforge.net/
 * The license stipulates that the icons are free to use, modify and redistribute.
 *
 * March 13, 2013
 * @version 1.0
 * @author Wesley R. Howell
 *
 */

public class emailWindow extends JFrame {

        private static final long serialVersionUID = 1L;
        private  File [] listofFiles;
        private String [] names;
        private String currentUser     = "matthew.crist";
        private String currentDomain   = "localhost";
        private String currentPassword = "simulation";

        /**
         * Run The Email Window object. Most of the code will be handled here.
         */
        public emailWindow() {
                getContentPane().setBackground(new Color(192, 192, 192));
                setIconImage(Toolkit.getDefaultToolkit().getImage("lib/icon.gif"));
                /**
                 * Close the application when the window closes.
                 */
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                /**
                 * Setup Code
                 */
                setTitle("ACL2 Email System\n");
                getContentPane().setLayout(null);
                final JList list = new JList();
                list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
                list.setBorder(new EtchedBorder(EtchedBorder.RAISED, null, null));
                final JLabel lblBlah = new JLabel();
                lblBlah.setBackground(Color.WHITE);
```

```
		JButton btnGetEmail = new JButton("Get Email");
		btnGetEmail.setIcon(new ImageIcon("lib/email_open.png"));


		/**
		 * Get Email Button Action
		 */
		btnGetEmail.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {
				String s;
//
				//GetEmail.getEmail("matthew.crist", "localhost", "simulation");
				GetEmail.getEmail(currentUser, currentDomain, currentPassword);
				try{
				Thread.sleep(2000);
				} catch (InterruptedException e){
				    e.printStackTrace();
				}
				//try {
//					Process p = Runtime.getRuntime().exec(
//							"sh /Users/w_howell/code/spring-2013-se2-dijkstra
    /client/modules/email/email-action/java-email.sh");
//					BufferedReader err = new BufferedReader (
//							new InputStreamReader (p.getErrorStream()));
//					while ((s = err.readLine()) != null) {
//						System.out.println(s);
//					}
//				  }
//				  catch (Exception e) {
//					e.printStackTrace();
//				  }

				//Update Jlist
				File folder = new File("store/email/inbox");
				folder.mkdirs();
			File [] rawContents = folder.listFiles();
			listofFiles = new File[rawContents.length];
			names = new String[rawContents.length];
		       // Map<String, File> listforgui = new HashMap<String,File>();
			for (int i = 0; i < rawContents.length; i++){
				if(rawContents[i].isFile() && !rawContents[i].isHidden()){
					listofFiles[i] = rawContents[i];
					names[i] = getEmailHeader(listofFiles[i]);
				}
			}
			list.setListData(names);
			}
		});
		btnGetEmail.setBounds(16, 6, 117, 29);
		getContentPane().add(btnGetEmail);




		/**
		 * Load the Files in the inbox and add to the list of input
		 */
		File folder = new File("store/email/inbox");
	File [] rawContents = folder.listFiles();
	listofFiles = new File[rawContents.length];
	names = new String[rawContents.length];
       // Map<String, File> listforgui = new HashMap<String,File>();
	for (int i = 0; i < rawContents.length; i++){
		if(rawContents[i].isFile() && !rawContents[i].isHidden()){
			listofFiles[i] = rawContents[i];
			names[i] = getEmailHeader(listofFiles[i]);
		}
	}
```

```
		/**
		 * Read in the email message when it is clicked
		 */
		list.addListSelectionListener(new ListSelectionListener() {
			public void valueChanged(ListSelectionEvent arg0) {
				//Open file and get text!

				BufferedReader reader;
				try {
					StringBuilder  stringBuilder = new StringBuilder();
					if(list.getSelectedIndex() != -1){
					reader = new BufferedReader( new FileReader (listofFiles[list.
getSelectedIndex()]));
					 String         line = null;

					    String         ls = System.getProperty("line.separator");

					    while( ( line = reader.readLine() ) != null ) {
					        stringBuilder.append( line );
					        stringBuilder.append( ls );
					    }
					}

					    lblBlah.setText(stringBuilder.toString());
				} catch (FileNotFoundException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				} catch (IOException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				}

			}
		});
		list.setBounds(16, 44, 315, 612);
		list.setListData(names);
		getContentPane().add(list);


		/**
		 * Extra "eye candy" in the window
		 */
		JPanel panel = new JPanel();
		panel.setBackground(Color.WHITE);
		panel.setBorder(new EtchedBorder(EtchedBorder.RAISED, null, null));
		panel.setBounds(336, 44, 682, 681);
		getContentPane().add(panel);
		panel.setLayout(null);


		/**
		 * Setup for Email Viewer
		 */
		lblBlah.setVerticalAlignment(SwingConstants.TOP);
		lblBlah.setBounds(6, 6, 670, 669);
		panel.add(lblBlah);

		JLabel lblNewLabel = new JLabel("");
		lblNewLabel.addMouseListener(new MouseAdapter() {
			@Override
			public void mouseClicked(MouseEvent arg0) {
				try {
					String url = "http://en.wikipedia.org/wiki/ACL2";
				 java.awt.Desktop.getDesktop().browse(java.net.URI.create(url));
```

```
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
        });
        lblNewLabel.setIcon(new ImageIcon("lib/logo.gif"));
        lblNewLabel.setBounds(16, 688, 83, 74);
        getContentPane().add(lblNewLabel);
        JLabel lblAcl = new JLabel("<html><h2>ACL2 Email System</h2><p>2013 Team Dijkstra \n
University of Oklahoma</p></html>\n");
        lblAcl.setVerticalAlignment(SwingConstants.TOP);
        lblAcl.setBounds(121, 688, 183, 74);
        getContentPane().add(lblAcl);

        /**
         * New Email Message Button
         */
        JButton btnNewMessage = new JButton("New Message");
        btnNewMessage.setIcon(new ImageIcon("lib/email_edit.png"));
        btnNewMessage.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        newMessage msg = new newMessage();
                        msg.setSize(602,487);
                        msg.setVisible(true);
                }
        });
        btnNewMessage.setBounds(142, 6, 132, 29);
        getContentPane().add(btnNewMessage);

        JLabel lblcodegooglecompspringsedijkstra = new JLabel("<html><font size=4><a href=https://
code.google.com/p/spring-2013-se2-dijkstra//p/>code.google.com/p/spring-2013-se2-dijkstra</a>\n    </
font></html>");
        lblcodegooglecompspringsedijkstra.addMouseListener(new MouseAdapter() {
                @Override
                public void mouseClicked(MouseEvent arg0) {


                        // open the default web browser for the HTML page
                        try {
                                String url = "https://code.google.com/p/spring-2013-se2-dijkstra/";
                         java.awt.Desktop.getDesktop().browse(java.net.URI.create(url));
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
        });

        /**
         * Link to Google Page
         */
        lblcodegooglecompspringsedijkstra.setVerticalAlignment(SwingConstants.TOP);
        lblcodegooglecompspringsedijkstra.setBounds(688, 737, 330, 25);
        getContentPane().add(lblcodegooglecompspringsedijkstra);


        /**
         * Register Email button
         */
        JButton btnRegister = new JButton("Register");
        btnRegister.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                currentUser = JOptionPane.showInputDialog("Input your name");
                currentDomain  = JOptionPane.showInputDialog("input your domain");
                currentPassword  = JOptionPane.showInputDialog("Input your Password");

                SendRequest.sendRequest(currentUser, currentDomain, currentPassword);
```

```
            }
        });
        btnRegister.setIcon(new ImageIcon("lib/user_add.png"));
        btnRegister.setBounds(901, 6, 117, 29);
        getContentPane().add(btnRegister);

JButton btnNewButton = new JButton("Delete Message");
btnNewButton.setIcon(new ImageIcon("lib/email_delete.png"));
        btnNewButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
        int currentlySelected = list.getSelectedIndex();
        listofFiles[currentlySelected].delete();


        //Update Jlist
                    File folder = new File("store/email/inbox");
        folder.mkdirs();
        File [] rawContents = folder.listFiles();
        listofFiles = new File[rawContents.length];
        names = new String[rawContents.length];

        for (int i = 0; i < rawContents.length; i++){
                        if(rawContents[i].isFile() && !rawContents[i].isHidden()){
                                listofFiles[i] = rawContents[i];
                                names[i] = getEmailHeader(listofFiles[i]);
                        }
        }
        list.setListData(names);
            }
        });
        btnNewButton.setBounds(16, 657, 315, 29);
        getContentPane().add(btnNewButton);


}

/**
 * Gets the email header for listing in the JFrame list
 * @param file
 * @return
 */
@SuppressWarnings("resource")
private String getEmailHeader(File file){
        String rv = "";
        BufferedReader reader;
        try {
                reader = new BufferedReader( new FileReader (file));
                        String         line = null;
                        String subject = "";
                    String name ="";

                    int number = 0;
                    while( ( line = reader.readLine() ) != null ) {
                        if(number == 1){
                                subject = line.substring(4, line.indexOf("</h1>"));

                        }
                        if(number == 3){
                                name = line.substring(11, line.indexOf("</h2>"));
                        }
                        number++;
                    }
                    rv = name + "  -  " + subject +"    -  " + file.getName().substring(4);
        } catch (FileNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        } catch (IOException e) {
                // TODO Auto-generated catch block
```

```
                        e.printStackTrace();
                }

                return rv;
        }
}
```

```
import modules.email.action.*;
import javax.swing.JDialog;
import javax.swing.JTextField;
import javax.swing.JTextArea;
import javax.swing.JLabel;
import javax.swing.SwingConstants;
import javax.swing.JButton;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import javax.swing.ImageIcon;
import java.awt.Color;

/**
 * New Message Dialog box
 *
 * Creates a new new email message dialog and allows a user to input a message
 * Then calls a shell script to send the message to ACL2 for message generation and
 * processing
 *
 * @author Wesley R. Howell
 *
 */

public class newMessage extends JDialog {
        /**
         *
         */
        private static final long serialVersionUID = 1L;
        private JTextField txtFdTo;
        private JTextField txtFdFrom;
        private JTextField txtFdSubject;
        private JTextArea textArea;
        public newMessage() {
                getContentPane().setBackground(Color.LIGHT_GRAY);
                setResizable(false);
                setTitle("New Message - ACL2 Email System\n");
                addWindowListener(new WindowAdapter() {
                        @Override
                        public void windowClosed(WindowEvent e) {
                                setVisible(false);
                        }
                });
                getContentPane().setLayout(null);

                txtFdTo = new JTextField();
                txtFdTo.setBounds(105, 65, 469, 28);
                getContentPane().add(txtFdTo);
                txtFdTo.setColumns(10);

                txtFdFrom = new JTextField();
                txtFdFrom.setBounds(105, 107, 469, 28);
                getContentPane().add(txtFdFrom);
                txtFdFrom.setColumns(10);

                txtFdSubject = new JTextField();
                txtFdSubject.setBounds(105, 154, 469, 28);
                getContentPane().add(txtFdSubject);
                txtFdSubject.setColumns(10);

                textArea = new JTextArea();
                textArea.setBounds(30, 213, 544, 187);
                getContentPane().add(textArea);
```

```
		JLabel lblTo = new JLabel("To:");
		lblTo.setHorizontalAlignment(SwingConstants.LEFT);
		lblTo.setBounds(32, 71, 61, 16);
		getContentPane().add(lblTo);

		JLabel lblFrom = new JLabel("From:");
		lblFrom.setHorizontalAlignment(SwingConstants.LEFT);
		lblFrom.setBounds(32, 113, 61, 16);
		getContentPane().add(lblFrom);

		JLabel lblSubject = new JLabel("Subject:");
		lblSubject.setHorizontalAlignment(SwingConstants.LEFT);
		lblSubject.setBounds(32, 160, 61, 16);
		getContentPane().add(lblSubject);

		JLabel lblMessage = new JLabel("Message:");
		lblMessage.setBounds(32, 190, 61, 16);
		getContentPane().add(lblMessage);

		/**
		 * Send Button Action
		 */
		JButton btnSend = new JButton("Send");
		btnSend.setIcon(new ImageIcon("lib/email_go.png"));
		btnSend.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {

				/**
				 * Send to ACL2
				 *
				 */
				// String s;
				String to = txtFdTo.getText();
				String from = txtFdFrom.getText();
				String sub = txtFdSubject.getText();
				String cont = textArea.getText();
//
				SendEmail.sendMessage(to, from, sub, cont);

//				  String[] cmdArray = {"sh", "/Users/w_howell/code/spring-2013-se2-dijkstra
    /client/modules/email/email-action/send-message-java.sh", to, from, sub, cont};
//
//				  try {
//					  Process p = Runtime.getRuntime().exec(cmdArray);
//
//					  BufferedReader err = new BufferedReader (
//						  new InputStreamReader (p.getErrorStream()));
//					  while ((s = err.readLine()) != null) {
//						  System.out.println(s);
//					  }
//				  }
//				  catch (Exception x) {
//					  x.printStackTrace();
//				  }

				setVisible(false);
			}
		});
		btnSend.setBounds(457, 428, 117, 29);
		getContentPane().add(btnSend);

		/**
		 * Cancel Button Action
		 */
		JButton btnCancel = new JButton("Cancel");
		btnCancel.setIcon(new ImageIcon("lib/cancel.png"));
		btnCancel.addActionListener(new ActionListener() {
```

```
                        public void actionPerformed(ActionEvent arg0) {
                                setVisible(false);
                        }
                });
                btnCancel.setBounds(328, 428, 117, 29);
                getContentPane().add(btnCancel);

                JLabel lblNewLabel_1 = new JLabel("<html><h2>Create New Message:</h2></html>");
                lblNewLabel_1.setBounds(30, 18, 238, 28);
                getContentPane().add(lblNewLabel_1);

                JLabel lblAclEmailSystem = new JLabel("<html><font size=2>ACL2 Email System <br>\n2013 Team ↙
    Dijkstra <br>\nUniversity of Oklahoma");
                lblAclEmailSystem.setBounds(30, 410, 124, 47);
                getContentPane().add(lblAclEmailSystem);
        }
}
```

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.util.*;
import javax.swing.*;

import lib.*;

public class Server extends JFrame {
    private ArrayList<Process> activeProcesses;
    private ModuleManager manager;

    /*
    * Default constructor for this class.
    */
    public Server()
    {
        // Initialize the module manager and load the modules
        this.manager = new ModuleManager();

        // Holder for the currently active processes
        this.activeProcesses = new ArrayList<Process>();

        this.initUI();
    }   // end constructor

    /**
     * Creates the GUI for the server to interface with the user.
     */
    private void initUI() {
        JMenuBar _menuBar = new JMenuBar();

        // Server actions
        JMenu     _server        = new JMenu("Server");
        JMenuItem _startServer   = new JMenuItem("Start Server", new ImageIcon("lib/accept.png"));
        JMenuItem _stopServer    = new JMenuItem("Stop Server", new ImageIcon("lib/stop.png"));
        JMenuItem _restartServer = new JMenuItem("Restart Server", new ImageIcon("lib/arrow_refresh.png"));
        JMenuItem _exit          = new JMenuItem("Exit Program");

        // Menu action listeners
        _startServer.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                startup();
            }   // end method actionPerformed
        });

        _stopServer.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                shutdown();
            }   // end method actionPerformed
        });

        _restartServer.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                restart();
            }   // end method actionPerformed
        });

        _exit.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                shutdown();
                System.exit(0);
            }   // end method actionPerformed
        });

        _server.add(_startServer);
        _server.add(_stopServer);
```

```
        _server.add(_restartServer);
        _server.addSeparator();
        _server.add(_exit);

        JMenu _modules = new JMenu("Modules");
        JMenuItem _manageModules   = new JMenuItem("Management");
        _modules.add(_manageModules);

        _manageModules.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                // Add through new module addition
                manager.getManageModuleDialog();
            }   // end method actionPerformed
        });

        JMenuItem _help = new JMenuItem("Help");
        _help.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                try {
                    Desktop.getDesktop().browse(new URI("https://code.google.com/p/spring-2013-se2-dijkstra ↙
"));
                } catch(Exception e) {
                    e.printStackTrace();
                }   // end try/catch
            }   // end method actionPerformed
        });

        _menuBar.add(_server);
        _menuBar.add(_modules);
        _menuBar.add(_help);

        JPanel _layoutPanel = new JPanel(new BorderLayout());
        JScrollPane scroller = new JScrollPane(ServerConsole.console);
        _layoutPanel.add(scroller, BorderLayout.CENTER);

        this.setJMenuBar(_menuBar);
        this.add(_layoutPanel);
        this.setBounds(0, 0, 600, 300);
        this.setVisible(true);

        // When the frame is exited, we need to shut down the services
        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                shutdown();
                System.exit(0);
            }   // end method windowClosing
        });
    }   // end method initUI

    /**
     * Starts all the processes.
     */
    public void startup() {
        ArrayList<Module> modules = this.manager.getModules();

        ServerConsole.post("System startup in progress...\n");

        // Cycle through all available modules and start their process
        for(int i = 0; i < modules.size(); i++) {
            try {
                // Process to initiate
                ProcessBuilder pb = new ProcessBuilder("java", modules.get(i).getInvocation());
                // Enable logging for startup file
                File startLog = new File("logs/startup.txt");
                pb.redirectErrorStream(true);
                pb.redirectOutput(ProcessBuilder.Redirect.appendTo(startLog));

                // Start the process and add it to the list of active processes
```

```
                Process p = pb.start();
                activeProcesses.add(p);

                ServerConsole.post("Module Listening: [" + modules.get(i).getName() + "]; Port: [" +
    modules.get(i).getPort() + "]\n");
            } catch(Exception e) {
                e.printStackTrace();
            }   // end try/catch
        }   // end for loop
    }   // end method startup

    /**
     * Shuts down all the processes.
     */
    public void shutdown() {
        ServerConsole.post("System shutdown in progress...\n");
        for(int i = 0; i < activeProcesses.size(); i++)
        {
            this.activeProcesses.get(i).destroy();
            this.activeProcesses.remove(i);
        }   // end for loop
    }   // end method shutdown

    /**
     * Restarts all the processes.
     */
    public void restart() {
        this.shutdown();
        this.startup();
    }   // end method restart

    /**
     * Main entry point for this application.
     */
    public static void main(String[] args) {
        new Server();
    }   // end function main
}   // end class Server
```

```
package lib;

import javax.swing.JTextArea;

public class ServerConsole {
    public static JTextArea console = new JTextArea();

    public static void post(String text) {
        ServerConsole.console.setText(ServerConsole.console.getText().concat(text));
    }   // end method post
}   // end class ServerConsole
```

```
package lib;

/*******************************************************************************
  * ModuleManager.java
  * Created on March 26, 2013 by Matthew A. Crist.
  *
  * This class will manage to loading of module information from the
  * modules.xml file that is stored in the ./config directory.
  *
  * CHANGE LOG:
  * ---------------------------------------------------------------------------
  * 2013-03-26  -   Initial conception of this file.
  *
  ******************************************************************************/

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class ModuleManager extends JDialog {
    public ArrayList<Module> modules = new ArrayList<Module>();

    private JDialog    _addDialog;
    private JDialog    _modDialog;
    private JTextField _fileInput;
    private JTextField _nameInput;
    private JTextField _portInput;
    private JButton    _removeModule;
    private JButton    _modifyModule;
    private JButton    _saveModules;
    private JList      moduleList;
    private JPanel     _addLayout;
    private JPanel     _manageLayout;

    private Module _temp;

    /**
     * Default, unargumented constructor for this class.
     */
    public ModuleManager() {
        modules = loadModules("config/modules.xml");
    }   // end default, unargumented constructor

    /**
     * Acquires the dialog that will allow a user to manage the current modules
     * that are registered with the server.
     */
    public void getManageModuleDialog() {
        _modDialog    = new JDialog();
        _manageLayout = new JPanel(null);

        DefaultListModel<String> listModel = new DefaultListModel<String>();
        moduleList = new JList(listModel);
        moduleList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        JScrollPane scroller = new JScrollPane(moduleList);

        // Add all the modules to the list model
        for(int i = 0; i < this.modules.size(); i++) {
            listModel.addElement(this.modules.get(i).getName());
        }   // end for loop

        JButton _addButton = new JButton("Add", new ImageIcon("lib/brick_add.png", "Add Module"));
        _removeModule = new JButton("Remove", new ImageIcon("lib/brick_delete.png", "Remove Module"));
```

```
    _modifyModule = new JButton("Modify", new ImageIcon("lib/brick_edit.png", "Modify Module"));
    _saveModules  = new JButton("Save", new ImageIcon("lib/disk.png", "Save Modules"));
    JButton _cancelButton = new JButton("Cancel", new ImageIcon("lib/cancel.png", "Cancel Action"));

    _removeModule.setEnabled(false);
    _modifyModule.setEnabled(false);

    // List Selection for button activation
    moduleList.addListSelectionListener(new ListSelectionListener() {
        public void valueChanged(ListSelectionEvent lse) {
            if(lse.getValueIsAdjusting() == false) {
                if(moduleList.getSelectedIndex() == -1) {
                    _removeModule.setEnabled(false);
                    _modifyModule.setEnabled(false);
                } else {
                    _removeModule.setEnabled(true);
                    _modifyModule.setEnabled(true);
                }   // end if-else
            }   // end if
        }   // end method valueChanged
    });

    // Add button action listener
    _addButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            _modDialog.setVisible(false);
            getAddModuleDialog();
        }   // end method actionPerformed
    });

    _saveModules.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            boolean status = storeModules("config/modules.xml");

            if(status) {
                JOptionPane.showMessageDialog(((JButton)ae.getSource()).getRootPane(), "Modules updated ↙
 successfully!");
                _modDialog.dispose();
            } else {
                JOptionPane.showMessageDialog(((JButton)ae.getSource()).getRootPane(), "Unable to      ↙
update modules.  Please see console output.", "Module Management Error", JOptionPane.ERROR_MESSAGE);
                _modDialog.dispose();
            }   // end if-else
        }   // end method actionPerformed
    });

    _cancelButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            _modDialog.dispose();
        }   // end method actionPerformed
    });

    _modifyModule.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            Module _selected = getModule((String)moduleList.getSelectedValue());
            getAddModuleDialog(_selected.getName(), _selected.getInvocation(), _selected.getPort());
            _modDialog.setVisible(false);
        }   // end method actionPerformed
    });

    _manageLayout.setBorder(BorderFactory.createTitledBorder("Module Management"));

    scroller.setBounds(20, 20, 200, 300);
    _addButton.setBounds(230, 20, 100, 20);
    _modifyModule.setBounds(230, 45, 100, 20);
    _removeModule.setBounds(230, 70, 100, 20);
    _saveModules.setBounds(230, 300, 100, 20);
    _cancelButton.setBounds(230, 275, 100, 20);
```

```
    _manageLayout.add(scroller);
    _manageLayout.add(_addButton);
    _manageLayout.add(_modifyModule);
    _manageLayout.add(_removeModule);
    _manageLayout.add(_saveModules);
    _manageLayout.add(_cancelButton);

    _modDialog.add(_manageLayout);

    int width  = Toolkit.getDefaultToolkit().getScreenSize().width;
    int height = Toolkit.getDefaultToolkit().getScreenSize().height;
    _modDialog.setBounds(width/2-185, height/2-190, 370, 380);
    _modDialog.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    _modDialog.setVisible(true);
}   // end method getManageModuleDialog

/**
 * Overloaded getAddModuleDialog that will get the "ADD" action to a module.
 */
public void getAddModuleDialog() {
    getAddModuleDialog(null, null, null);
}   // end method getAddModuleDialog

/**
  * Acquires the dialog that will allow a user to register a module with the
  * server for monitoring.  EDIT by default for parameters passed.
  */
public void getAddModuleDialog(String t_name, String t_invoke, String t_port) {
    final String _name = t_name;
    final String _invoke = t_invoke;
    final String _port = t_port;

    _addLayout = new JPanel(null);
    _addDialog = new JDialog();

    JLabel _nameLabel   = new JLabel("Module Name:");
    JLabel _invokeLabel = new JLabel("Script to Invoke:");
    JLabel _portLabel   = new JLabel("Listening Port:");

    _nameInput  = new JTextField();
    _fileInput  = new JTextField();
    _portInput  = new JTextField();

    // Determine if we are editing or it is a new entry
    if((_name != null) && (_invoke != null) && (_port != null)) {
        _nameInput.setText(_name);
        _fileInput.setText(_invoke);
        _portInput.setText(_port);

        // Temporarily store the old name to ensure it's removal if edited.
        _temp = getModule(_name);
        removeModule(_name);
    }   // end if

    JButton _fileSelect       = new JButton("Select...");
    JButton _saveModuleButton = new JButton("Save");
    JButton _cancelButton     = new JButton("Cancel");

    _fileSelect.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            JFileChooser chooser = new JFileChooser(".");
            int val = chooser.showOpenDialog(_addDialog);

            if(val == JFileChooser.APPROVE_OPTION) {
                String filePath      = chooser.getSelectedFile().getPath();
                // Kill the absolute package resolution
                String modulePackage = filePath.substring(filePath.lastIndexOf("server"), filePath.    ↙
```

```
length()-1);
                // Replace the file separators with . for the package name resolution
                modulePackage = modulePackage.replace(System.getProperty("file.separator"), ".");
                // Trim off server. portion of string
                modulePackage = modulePackage.substring(modulePackage.indexOf(".")+1, modulePackage.
length()-1);
                // Trim off the file extension.
                modulePackage = modulePackage.substring(0, modulePackage.lastIndexOf("."));
                _fileInput.setText(modulePackage);
            }   // end if
        }   // end method actionPerformed
    });

    // The save action against the new module.
    _saveModuleButton.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent ae)
        {
            Module module = new Module();
            module.setName(_nameInput.getText());
            module.setInvocation(_fileInput.getText());
            module.setPort(_portInput.getText());

            // Remove the old module if declared and reset the name handler
            if(_temp != null) {
                _temp = null;
            }   // end if

            // Add the module and dispose of the window
            addModule(module);
            _addDialog.dispose();

            getManageModuleDialog();
        }   // end method actionPerformed
    });

    /**
     * ActionListener for the cancel button that will discard any changes that
     * occured to the module, and dispose of the frame.
     */
    _cancelButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            if(_temp != null) {
                addModule(_temp);
                _temp = null;
            }
            _addDialog.dispose();
            getManageModuleDialog();
        }   // end method actionPerformed
    });

    _nameLabel.setBounds(20, 20, 100, 20);
    _invokeLabel.setBounds(20, 45, 100, 20);
    _portLabel.setBounds(20, 70, 100, 20);

    _nameInput.setBounds(130, 20, 150, 20);
    _fileInput.setBounds(130, 45, 150, 20);
    _portInput.setBounds(130, 70, 150, 20);

    _fileSelect.setBounds(290, 45, 100, 20);

    _addLayout.add(_nameLabel);
    _addLayout.add(_nameInput);
    _addLayout.add(_invokeLabel);
    _addLayout.add(_fileInput);
    _addLayout.add(_fileSelect);
    _addLayout.add(_portLabel);
    _addLayout.add(_portInput);
```

```
    JSeparator sep = new JSeparator();
    sep.setBounds(20, 95, 380, 2);
    _addLayout.add(sep);

    _cancelButton.setBounds(190, 105, 100, 20);
    _addLayout.add(_cancelButton);
    _saveModuleButton.setBounds(300, 105, 100, 20);
    _addLayout.add(_saveModuleButton);

    _addLayout.setBorder(BorderFactory.createTitledBorder("Register Module"));

    _addDialog.add(_addLayout);
    _addDialog.setBounds(0,0, 420, 170);
    _addDialog.setResizable(false);
    _addDialog.setVisible(true);
    _addDialog.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
}   // end method getAddModuleDialog

/**
  * Adds a module to the modules list.
 */
public boolean addModule(Module module) {
    boolean ok = true;

    for(int i = 0; i < this.modules.size(); i++) {
        if(this.modules.get(i).getPort().equalsIgnoreCase(module.getPort())) {
            ok = false;
        }   // end if
    }   // end for loop

    // Attempt to add the module.
    if(ok) {
        this.modules.add(module);
        this.storeModules("config/modules.xml");

        return true;
    } else {
        JOptionPane.showMessageDialog(this, "Unable to add module.  There exists a module using the same port as the one you are attempting to register.", "Error Registering Module", JOptionPane.ERROR_MESSAGE);
        return false;
    }   // end if-else
}   // end method addModule

/**
 * Removes a module from the modules list.
  */
public boolean removeModule(String name) {
    boolean removed = false;

    for(int i = 0; i < this.modules.size(); i++) {
        if(this.modules.get(i).getName().toLowerCase().equals(name.toLowerCase())) {
            this.modules.remove(i);
            removed = true;
        }   // end if
    }   // end for loop

    return removed;
}   // end method removeModule

/**
 * Acquires a module by its name.
 */
public Module getModule(String name) {
    for(int i = 0; i < this.modules.size(); i++) {
        if(this.modules.get(i).getName().toLowerCase().equals(name.toLowerCase())) {
            return this.modules.get(i);
```

```
        }   // end if
    }   // end for loop

    return null;
}   // end method getModule

/**
 * Loads the modules from a persistent XML storage file.
 */
public ArrayList<Module> loadModules(String configFile)
{
    ArrayList<Module> modules = new ArrayList<Module>();

    try
    {
        File moduleFile = new File(configFile);
        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        Document doc = dBuilder.parse(moduleFile);

        doc.getDocumentElement().normalize();

        NodeList nodes = doc.getElementsByTagName("module");

        for(int i = 0; i < nodes.getLength(); i++)
        {
            Node node = nodes.item(i);

            if(node.getNodeType() == Node.ELEMENT_NODE)
            {
                Element element = (Element)node;
                Module module = new Module();
                module.setName(element.getElementsByTagName("name").item(0).getTextContent());
                module.setInvocation(element.getElementsByTagName("invoke").item(0).getTextContent());
                module.setPort(element.getElementsByTagName("port").item(0).getTextContent());

                modules.add(module);
            }
        }
    }   // end try
    catch(Exception e)
    {
        e.printStackTrace();
    }   // end catch

    return modules;
}   // end method loadModules

/**
 * Writes the modules out to an XML file for storage.
 */
public boolean storeModules(String configFile)
{
    // Header information for the XML file.
    String xml = "<?xml version='1.0'?>\n"                                  +
                     "<!DOCTYPE modules SYSTEM '../dtd/module.dtd'>\n" +
                     "<modules>\n";

    // Cycle through all modules and compile the XML for output.
    for(int i = 0; i < this.modules.size(); i++)
    {
        xml += "\t<module>\n"                                                   +
                  "\t\t<name>" + this.modules.get(i).getName() + "</name>\n"            +
            "\t\t<invoke>" + this.modules.get(i).getInvocation() + "</invoke>\n"  +
            "\t\t<port>" + this.modules.get(i).getPort() + "</port>\n" +
                  "\t</module>\n";
    }   // end for loop
```

```
        xml += "</modules>";

        try
        {
            FileWriter writer = new FileWriter(configFile);
            writer.write(xml);
            writer.close();
        }   // end try
        catch(IOException ioe)
        {
            ioe.printStackTrace();
            return false;
        }   // end catch

        return true;
    }   // end method storeModules

    /**
     * Acquires the set of modules that are currently stored in the module manager.
     */
    public ArrayList<Module> getModules()
    {
        return this.modules;
    }   // end method getModules
}   // end class ModuleManager
```

```
package lib;

/*******************************************************************************
  * Module.java
  * Created by Matthew A. Crist on March 26, 2013.
  *
  * The class defines the basic characteristics of a module and the requirements
  * that will be enforced when such moduel requires network transmission of
  * information.
  *
  * CHANGE LOG:
  * ---------------------------------------------------------------------------
  * 2013-03-26  -   Initial conception of this file.
  *
  ******************************************************************************/

import java.io.IOException;
import java.net.ServerSocket;

public class Module {
    /** The name of the module. */
    private String name;
    /** The program/script to be invoked upon connectivity. */
    private String invocation;
    /** The port in which to listen for connectivity. */
    private String port;

    /**
     * Default constructor that takes no arguments and initializes this
     * module's parameters to default values.
     */
    public Module() {
        this.name       = "Unknown Module";
        this.invocation = "./";
        this.port       = "0";
    }   // end default constructor

    /**
     * Argumented constructor for this class that will set the name, port
     * and script to be invoked upon successfully connection with the client.
     */
    public Module(String name, String invocation, String port)  {
        this.name       = name;
        this.invocation = invocation;
        this.port       = port;
    }   // end argumented constructor

    /**
     * Sets the name for this module.
     */
    public void setName(String name)    {
        this.name = name;
    }   // end method setName

    /**
      * Sets the invocation script for this module.
     */
    public void setInvocation(String invocation)    {
        this.invocation = invocation;
    }   // end method setInvocation

    /**
      * Sets the port for this module.
     */
    public void setPort(String port) {
        this.port = port;
    }   // end method setPort
```

```
    /**
     * Acquires the name of this module.
     */
    public String getName() {
        return this.name;
    }   // end method getName

    /**
      * Acquires the script to be invoked by this server on connectivity.
     */
    public String getInvocation()   {
        return this.invocation;
    }   // end method getInvocation

    /**
     * Acquires the port in which the server should be listening for this
     * module.
     */
    public String getPort() {
        return this.port;
    }   // end method getPort
}   // end class module
```

Appendix D: PSP File for Main Project

psp.txt

```
name: Team Dijkstra
date: April 16, 2013
program: Email with ACL2
instructor: Dr. Rex Page
language: ACL2, C++, Shell Scripting, XML

actual added lines: 401
actual base lines: 66
actual modified lines: 15
actual removed lines: 0

new objects:

        - name: xmlScanner
          estimated lines: 18
          type: NonIO

        - name: xmlParser
          estimated lines: 18
          type: NonIO

        - name: getSubject
          estimated lines: 9
          type: NonIO

        - name: getFrom
          estimated lines: 9
          type: NonIO

        - name: getTo
          estimated lines: 9
          type: NonIO

        - name: getEmail
          estimated lines: 14
          type: IO

        - name: getInboxFiles
          estimated lines: 18
          type: NonIO

        - name: removeInboxFile
          estimated lines: 18
          type: NonIO

        - name: registerClient
          estimated lines: 18
          type: NonIO

        - name: writeEmail
          estimated lines: 18
          type: NonIO

        - name: writePreferences
          estimated lines: 18
          type: NonIO

        - name: getBlocked
          estimated lines: 14
          type: IO

        - name: getTag
          estimated lines: 14
```

```
  type: IO

- name: parseMessage
  estimated lines: 18
  type: NonIO

- name: parseSpamKey
  estimated lines: 18
  type: NonIO

- name: toServer
  estimated lines: 9
  type: IO

- name: fromServer
  estimated lines: 9
  type: IO

- name: getErrors
  estimated lines: 18
  type: NonIO

- name: generateSpamKey
  estimated lines: 18
  type: NonIO

- name: generateMessage
  estimated lines: 18
  type: NonIO

- name: generateRegRequest
  estimated lines: 18
  type: NonIO

- name: getMessageLength
  estimated lines: 18
  type: NonIO

- name: writeHeaderInfo
  estimated lines: 18
  type: NonIO

- name: parseHeaderInfo
  estimated lines: 18
  type: NonIO

- name: getContentType
  estimated lines: 9
  type: NonIO

- name: setContentType
  estimated lines: 9
  type: NonIO

- name: ReadMsgInput
  estimated lines: 16
  type: IO

- name: readRegInput
  estimated lines: 16
  type: IO

- name: writeToAddressBook
```

```
  estimated lines: 16
  type: IO

- name: writeToClientInbox
  estimated lines: 16
  type: IO

- name: processMessage
  estimated lines: 18
  type: NonIO

- name: processRequest
  estimated lines: 18
  type: NonIO

- name: isSpam
  estimated lines: 9
  type: NonIO

- name: isBlockedUser
  estimated lines: 6
  type: NonIO

- name: parseAddress
  estimated lines: 6
  type: NonIO

- name: createUserGroup
  estimated lines: 14
  type: IO

- name: createMailingList
  estimated lines: 16
  type: IO

- name: updateMailingList
  estimated lines: 16
  type: IO

- name: getAvailableMailingList
  estimated lines: 16
  type: IO

- name: getMailingListRequest
  estimated lines: 9
  type: IO

- name: getMailingListRegistrationRequest
  estimated lines: 16
  type: IO

- name: isInUserGroup
  estimated lines: 18
  type: NonIO

- name: isInMailingList
  estimated lines: 18
  type: NonIO

- name: isRegistered
  estimated lines: 18
  type: NonIO
```

psp.txt

```
- name: isNameAvailable
  estimated lines: 18
  type: NonIO

- name: getMessageLength
  estimated lines: 9
  type: NonIO

- name: readInbox
  estimated lines: 16
  type: IO

- name: storeClientName
  estimated lines: 14
  type: IO


- name: getAddress
  estimated lines: 9
  type: Non IO

- name: email
  estimated lines: 14
  type: IO

- name: inbox
  estimated lines: 16
  type: IO

- name: register
  estimated lines: 16
  type: IO

- name: blockUser
  estimated lines: 18
  type: NonIO

- name: tagMessage
  estimated lines: 18
  type: NonIO

- name: getServerAddressBook
  estimated lines: 16
  type: IO

- name: getLocalAddressBook
  estimated lines: 18
  type: NonIO

- name: updateAddressBook
  estimated lines: 18
  type: NonIO

- name: getMailingLists
  estimated lines: 16
  type: IO

- name: createMailingList
  estimated lines: 16
  type: IO

- name: registerForMailingList
  estimated lines: 16
```

Page 4

```
  type: IO

- name: isRegistered
  estimated lines: 18
  type: NonIO

- name: isNameAvailiable
  estimated lines: 18
  type: NonIO

- name: isInAddressBook
  estimated lines: 18
  type: NonIO

- name: addSpamFilter
  estimated lines: 9
  type: NonIO

- name: addSpamKeyword
  estimated lines: 9
  type: NonIO

- name: addSpamAddress
  estimated lines: 9
  type: NonIO

- name: getDomain
  estimated lines: 6
  type: Non IO

- name: getName
  estimated lines: 6
  type: Non IO

- name: getPassword
  estimated lines: 6
  type: Non IO

- name: registerUser
  estimated lines: 14
  type: IO

- name: getEmailXML
  estimated lines: 18
  type: NonIO

- name: getEmailStructure
  estimated lines: 9
  type: NonIO

- name: getEmailStructureList
  estimated lines: 9
  type: NonIO

-       name: consume
        estimated lines: 6
        type: Non IO

-       name: tag
        estimated lines: 6
        type: Non IO

-       name: pcData
```

```
                estimated lines: 4
                type: Non IO

        -       name: nextToken
                estimated lines: 9
                type: Non IO

        -       name: tokenizeXML
                estimated lines: 3
                type: Non IO

time log:

       - date: Jan 17, 2013
         start time: 10:30AM
         end time: 11:45AM
         phase: Conception
         comment: Team Dijkstra's regular meeting time. We brainstormed project
ideas and decided on a networked chat client and server system.

-       date: January 18, 2013
                start time: 1:00am
                end time: 2:00am
                phase: Testing
                comment: Tested network connectvity for ACL2 using various methods.
Found that through file streaming, we could transfer files through connected network
drives.  We are not able to connect to a specific TCP/IP address or anything without
a directory resolution on the network.  Example: \\MatthewCrist-Laptop would work,
\\127.0.0.1 would work, but http://127.0.0.1 or \\127.0.0.1:80 would not yeild
results.  I have determined that either we will need supplemental language in order
to make the transfer or map network drives.


       - date: Jan 22, 2013
         start time: 10:30AM
         end time: 11:45AM
         phase: Conception
         comment: Team Dijkstra's regular meeting time. Finalized the idea of the
chat system and began initial designs for the proposal.

        - date: Jan 22, 2013
          start time: 7:32PM
          end time: 8:44PM
          phase: Conception
          comment: Worked on updating the team's LoC table to include tImpl from
last semester. Reused the spreadsheet from last semester and added the new data.
Then included the information into the t2 and t3 documents.

       - date: Jan 22, 2013
         start time: 8:53PM
         end time: 10:49PM
         phase: Testing
         comment: Worked on researching networking from within ACL2. There is not
any native support for networking. However, we did discover that we could use
networking if the Operating System supports it. Since our operating systems can be
networked, we can work around this limitation.

-       date: January 23, 2013
                start time: 2:30am
                end time: 4:00am
                phase: Testing
                comment: Attempted to research means by using Common Lisp to
implement a layer for TCP/IP transfer to see if this is a viable alternative.  The
```

solutions that are available seem to be beyond the scope to which we can implement. It appears the network drive route is our best alternative.

- date: Jan 23, 2013
  start time: 8:04PM
  end time: 10:27PM
  phase: Conception
  comment: Worked on writing the content of the Proposal. Finished the sections for the Overview, High-level design, some of the requirements, and began the PROBE estimate.

- date: Jan 24, 2013
  start time: 10:30AM
  end time: 11:45AM
  phase: Conception
  comment: Team Dijkstra's regular meeting time. We worked on the proposal and worked on several proofs of concepts to determine if the project is feasible.

- date: January 25, 2013
  start time: 10:00am
  end time: 10:20am
  phase: Testing
  comment: Researched event handling in ACL2 from previous project regarding "Blue Ball" scenario. Determined that this could be used to implement the client side gestures for the chat messages to be sent and received. Also noted that the primitive nature of the interface would require that we construct many of the GUI element ourselves. I think we should be able to assign functions to these events to read and write files where necessary.

- date: Jan 26, 2013
  start time: 4:04PM
  end time: 6:31PM
  phase: Conception
  comment: Finished the complete proposal with all the specified requirements and PROBE estimate.

- date: Jan 29, 2013
  start time: 10:30AM
  end time: 11:45AM
  phase: Conception
  comment: Team Dijkstra's regular meeting time. Dr. Page addressed concerns about the Project and that GUI's and File IO cannot exist in the same ACL2 program. We began a major re-design of the project and re-wrote most of the project requirements that are to be included in the proposal.

- date: Jan 29, 2013
  start time: 11:45AM
  end time: 2:12PM
  phase: Conception
  comment: Found out that some of the components of the original proposal were not feasible. Worked with Matthew to re-write sections of the proposal to include the new ideas for the project and write out the requirements for the project. The new project will be an email system instead of a chat system, with more emphasis on message delivery and content.

- date: January 29, 2013
  start time: 12:00pm
  end time: 2:30pm
  phase: Conception
  comment: After discussion with Rex Page regarding some of the features of ACL2, he described to us that ACL2 cannot use a GUI and write/read from files at the same time. This has thrown us in a tizzy about writing a chat client.

psp.txt

After reassessment of the situation, we determined that we could salvage much of the information that we derived in previous sessions by using supplemental language, such as C to write the server and client interfaces that would invoke ACL2 executables on demand and have these applications handle the events for both client and server processes. We could use folder monitoring on server side and invoke those modules that are created adding to the modularity of the application itself. Isaac, Wes and I rewrote the design document so that it would be available for Thursday (January 31, 2013) for the presentation. We also determined that Adam should be the one to give the overview of the application since he may want the extra speaking opportunity.

- date: January 30, 2013
  start time: 4:00am
  end time: 5:30am
  phase: Conception
  comment: Created the slides for the presentation that will be held tomorrow (January 31, 2013). Used the design document as a point of reference for bullet points. Also devided the slide information based on role (as opposed to design layer) to "sell" the idea to the end user and convey the purpose of the process we are intending to use.

- date: January 31, 2013
  start time: 8:00am
  end time: 9:30am
  phase: Testing
  comment: Attempted to get Proofpad to function on my windows PC (Microsoft Windows 8) to no success. Attempted to get it to work on my Linux partition as well (Ubuntu 11.04 LTS) with no such luck either. Both result in NullPointerExceptions being tossed back by the application itself. Unable to invoke ACL2 internally as a result. Decided to do some testing with ACL2 command line and get familiar with the environment Noticed that the teachpacks were not certified for io-utilities.lisp and list-utilities.lisp, so I had to include them in a subfolder and include them in the file directly. Cannot use program mode in ACL2 and have to stay in logic mode else the files will not include correctly into the project. Considering DrRacket as the IDE for development as a result.

- date: January 31, 2013
  start time: 1:00pm
  end time: 2:30pm
  phase: Coding
  comment: Started to develop the infrastructure for the server monitor program. Determined that the use of processes and a dependency relationship was necessary to ensure that a process was complete before invoking a dependent processes. Derived an XML DTD for the input format to load modules into the server environment for invocation. Assigned names to modules in the format (program).(content).(action) naming convention, a folder to monitor for files, and a process to invoke when a file is detected. Information has been updated on the wiki for reference on module creation under >> The Server Monitor.

- date: February 1, 2013
  start time: 4:30am
  end time: 5:20am
  phase: Coding
  comment: Did some more work on the Server Monitor. Wrote threading for directory monitoring and process invocation. Have not tied module invocation directly to program yet, as I need to verify that executables can be effectively created from Racket. File size seems to be rather large at the moment for just a hand full of code. May consider an alternative approach.

- date: Feb 3, 2013
  start time: 12:42AM
  end time: 2:09AM
  phase: Conception

comment: Matthew emailed about an issue with the ACL2 EXE's not working correctly and possibly being unable once the project gets large. Spent this time researching how to move the project to UNIX based servers and working with launching ACL2 from the terminal and using shell scripts to call ACL2 lisp files.

- date: February 3, 2013
  start time: 10:00am
  end time: 12:30pm
  phase: Testing
  comment: Determined that executables in ACL2 is not a practical solution. Tried invocations through redirection on the input method for ACL2 since the executable did not take any arguments. So far I have had success. Module invocation can occur through a shell script which does not require dependency checks (which will allow for linear execution.) Reassessing the use of unsynchronized process invocation. Perhaps synchronizing the main processing thread to halt while a process is running could be the best alternative. Invocation of a shell script also alleviates the overhead that may have been involved and we can implement some of the operating system features to transfer information (such as ftp for file transfer).

- date: Feb 4, 2013
  start time: 5:23PM
  end time: 6:47PM
  phase: Conception
  comment: Worked on typing the progress report with the completed task to date and writing out the plans for the task to be completed. Also noted in in the progress report the need to move the system server to a UNIX based system to run the server components

- date: February 5, 2013
  start time: 4:00am
  end time: 5:30am
  phase: Documentation
  comment: Updated progress report with findings over the last two weeks for delivery to Dr. Page and the team. Need to address issues with the change in the group make up, since Isaac will be unable to participate in the project until his personal issues are resolved.

- date: Feb 5, 2013
  start time: 10:30PM
  end time: 11:45PM
  phase: Conception
  comment: Team Dijkstra's regular meeting time. We delivered a progress report to Dr. Page. We worked on converting the Windows based designs to a UNIX file system in order for the system to run.

- date: Feb 5, 2013
  start time: 7:32PM
  end time: 8:35PM
  phase: Conception
  comment: Worked on setting up ACL2 modules to run though the UNIX shell and wrote test scripts to automatically call ACL2 functions from a shell script

- date: February 6, 2013
  start time: 10:00am
  end time: 11:00am
  phase: Coding
  comment: Reorganized server code into better directory management so ensure a structure that can be deployed more effectively. Updated references to be on local scope, as opposed to global folder references for portability purpose.

- date: Februaru 7, 2013
  start time: 6:30am

```
              end time: 8:00am
              phase: Coding
              comment: Wrote scanner portion for the analyzer for XML content that
can be parsed with ACL2.  Added consume, tag, pcData, nextToken, and tokenizeXML
functions.  Entry point is tokenizeXML which takes a string of XML and will return a
list of tokens and the token type.  This should then be added to a stack to verify
xml correctness and can then be used to extract the PCDATA information contained
between the brackets.

      - date: Feb 7, 2013
        start time: 10:30PM
        end time: 11:45PM
        phase: Conception
        comment: Team Dijkstra's regular meeting time. We worked on the XML format
and multi-level design and designed the data structure format for the project. We
then wrote sections of the progress report and assigned an XML module to each team
member.

      - date: Feb 9, 2013
        start time: 5:14PM
        end time: 7:32PM
        phase: Conception
        comment: Worked on designing the XML structure for email messages.
Designed the XML layout, Document type definition and explanation. These topics were
then uploaded to the groups wiki for review, comment, and inclusion in the design
document.

      - date: Feb 11, 2013
        start time: 5:12PM
        end time: 8:04PM
        phase: Conception
        comment: Worked on the team's design document and formatting of the
sections. Took the information from the group's wiki and generated the data
structure and IO format sections from this information. The other sections were
expanded from the initial proposal with updates to relevant sections where the
server information had been changed.

      - date: Feb 12, 2013
        start time: 10:30PM
        end time: 11:45PM
        phase: Conception
        comment: Team Dijkstra's regular meeting time. We reviewed the team design
document and made many changes to the XML format and found several errors. We marked
up the design for submission on Thursday.

      -       date: February 14, 2013
              start time: 12:15am
              end time: 12:30am
              phase: Conception
              comment: Derived the format, in XML, for the transportation of the
registration of a user into the address book for the server.  Also formulated the
steps to be involved in order to invoke the appropriate modules.

      -       date: February 14, 2013
              start time: 12:30am
              end time: 1:45am
              phase: Coding
              comment: Wrote the shell script that acquired the contents of the
source file (ACL2 definitions) and compiled a function invocation to be written to a
temporary file, input directed into ACL2 and removal of the temporary file created.
Considering just including the book to the source file and writing a new temporary
file that invokes the function and includes to appropriate books in order to not
manipulate the source files for unexpected results.
```

psp.txt

```
        -       date: February 14, 2013
                start time: 2:00am
                end time: 4:00am
                phase: Coding
                comment: Created functions for managing address-book.  getAddress,
parseAddresses, getAddressBook acquire values from the XML that was passed to the
script via redirection through the XML that was acquired through the shell
scripting.

        -       date: February 14, 2013
                start time: 6:15am
                end time: 8:00am
                phase: Coding
                comment:        Created the functions for managing output of the
address-book data structure.  addressXML, addressBookXML and getAddressBookXML will
acquire the XML for the structure to be written back to the xml data file.
Determined that I will need to write to a temporary file and delete the original
source, and rename the temporary file to the permanent persistent file in order to
allow for the file to be updated (since I cannot write back to a read file).
Updated shell script to reflect these changes in
./server/modules/user/register/register-user.sh.

        -       date: February 16, 2013
                start time: 1:15am
                end time: 3:00am
                phase: Coding
                comment: Added functions that acquires the contents of the tokens
passed by parsing the XML information.  getDomain, getName, getPassword all use
these tokens in order to extract the information from the registration file that
will be used to determine what information needs to be added to the address-book.

        -       date: February 16, 2013
                start time: 3:00am
                end time: 5:00am
                phase: Coding
                comment: Added functions that will test the address existence (to
prevent duplication) and add/remove an address from the address-book.
isInAddressBook tests the predicate conditions to determine if an address can be
added/removed by addAddress/removeAddress functions.

        - date: Feb 17, 2013
          start time: 3:43PM
          end time: 4:51PM
          phase: Coding
          comment: Worked on designing the server email components of the server
module. Wrote out data flow diagrams to match the data structure format and XML I/O
from the design document, and wrote function prototypes.

        - date: Feb 18, 2013
          start time: 5:05PM
          end time: 6:12PM
          phase: Conception
          comment: Typed the progress report for the presentation tomorrow to give
Dr. Page an update on our projects task and goals. Also updated the design document
to Revision I for submission tomorrow.

        - date: Feb 18, 2013
          start time: 7:23PM
          end time: 10:44PM
          phase: Coding
          comment: Worked on implementing the server email components. Took my
designs on paper from yesterday and worked them into working functions. I was able
```

to finish an email splitter which splits one email sent to many recipients into multiple messages and XML output for email messages.

- date: February 19, 2013
  start time: 9:00am
  end time: 10:30am
  phase: Coding
  comment: Updated shell script to generate static script that will invoke the registerUser function also defined in the register-user.lisp file. Temporary lisp file is generated with static XML information and then redirected into the ACL2 program in order to parse the information. File input is delegated to the shell, file output is delegated to ACL2.

- date: Feb 19, 2013
  start time: 10:30AM
  end time: 11:45AM
  phase: Conception
  comment: Team Dijkstra's regular meeting time. We delivered a progress report to Dr. Page on our task so far. Also we updated our SVN repository to the most recent changes to the groups implementation. Isaac rejoined the group and the remaining time was spend catching him up on the groups progress.

- date: February 19, 2013
  start time: 3:00pm
  end time: 4:00pm
  phase: Coding
  comment: Corrected references the xml-scanner to ignore whitespace since some of the whitespace tokens that were being parsed were causing issues with the interpretation of the xml tokens. user/registration module deemed complete at current point in time.

- date: Feb 22, 2013
  start time: 4:32PM
  end time: 7:11PM
  phase: Coding
  comment: I worked on the ACL2 implementation of the server-email functions and finished the implementation of the components. The functions did not admit correctly and the details are described in the defect log. The issues were traced to file IO so we are still able to run and test the computational functions.

- date: Feb 22, 2013
  start time: 2:59PM
  end time: 4:16PM
  phase: Coding
  comment: I re-worked the file IO functions to accommodate the requirements of the IO utilities functions. This fixed the majority of the admission issues for these functions. Still the getEmail and runEmail functions are the global execution functions still have work needed to get them to admit and run with ACL2.

- date: February 25, 2013
  start time: 8:00am
  end time: 9:30am
  phase: Conception
  comment: Starting in the development of the network connectivity subsystem that will allow for the transmission of information across a network via IP address. Determined previously that the "nc" command in UNIX was sufficient in order to accomplish this task Proceeded to decompose the component and derived the need for a separate "method invocation" language.

- date: February 26, 2013
  start time: 8:00am
  end time: 9:15am
  phase: Conception

```
              comment: Proceeded to develop a rough draft for the diagram that
would be formalized into the design document for network connectivity.  Upon further
investigation, I realized that I need to find a way in order to send a request to
the server to open a sending port after the client has acknowledged opening a
recieving port for data transmission.  This appears to be the most difficult
scenario for the server to overcome.

        - date: Feb 26, 2013
          start time: 10:30AM
          end time: 11:45AM
          phase: Coding
          comment: Team Dijkstra's regular meeting time. We brainstormed bug issues
and found a solution to the Server-Email IO problem. We will resolve multiple email
messages from the command shell and call ACL2 for each individual message.

        - date: Feb 27, 2013
          start time: 12:01PM
          end time: 12:53PM
          phase: Coding
          comment: Worked on the IO contents and designing the Test and Theorems for
the server email module. Had an idea to combine the IO into one function to see if
it fixes the IO issue I'm having with the server-email. Started implementation and
will finish tonight.

        - date: Feb 27, 2013
          start time: 9:54PM
          end time: 10:32PM
          phase: Coding
          comment: I finally have the IO fixed. With one call I can have and input
file in XML format parse and get passed to an output file. Now its time to handled
multiple XML messages in one file.

        - date: Feb 28, 2013
          start time: 10:30AM
          end time: 11:45AM
          phase: Coding
          comment: Team Dijkstra's regular meeting time. We worked on server
implementation and fixed issues with the server-email file. We split it into two
files where one contained the IO functions and the other contained the logic
functions. This was done in order to make proving theorems in ACL2 easier.

        - date: Feb 28, 2013
          start time: 4:02PM
          end time: 5:12PM
          phase: Testing
          comment: I wrote the theorem suite for the server-email functions. These
tested each function in the file at least once. This guarantees that the written
code does what it is needed to do.

        -       date: March 1, 2013
                start time: 12:35am
                end time: 1:45am
                phase: Conception
                comment: Started to develop the basis for the "Server Messaging
Language" that would overcome the need for remote method invocation.  Syntax of
which would be encapsulated in curly brackets and separated by semicolins.  Started
prototyping functions to scan for this language.  Header information would be passed
via first line in the XML transmission over the network and would be saved in a
timestamp relative to UNIX conception date (date +%s.txt).

        - date: March 2, 2013
          start time: 4:53PM
          end time: 5:34PM
```

phase: Coding
comment: I wrote a shell script that dynamically writes a lisp file with the command to call the rwEmail function and output the files to a directory with the clients name in the servers store folder. Each file is names msg_timestamp where the timestamp is the actual timestamp of the message.

- date: March 4, 2013
  start time: 8:00am
  end time: 8:45am
  phase: Conception
  comment: Started to rough draft a client side development plan. Wes and I are considered the "server side" team, while we had hopes of Adam and Isaac being the "client side" team. Being that both Adam and Isaac were gone, Wes and I have had to develop a plan of action toward pushing for project completion. RMI has been temporarily placed on hold until we can solve these issues. Perhaps and explanation of the RAD process and a bit of motivation would be the route to go. Designed a flow chart for an overview of the basic functionality and what components were complete. Decided that I would give the presentation on the update of the status of the project.

- date: March 4, 2013
  start time: 11:49AM
  end time: 1:04PM
  phase: Coding
  comment: Worked on updating rwEmail. I changed the output file from being solely passed in by the parameters to where only a timestamp is required. The output directory is now dynamically generated to pull the <to> tag from the XML and outputs to output each email message into a directory based on the contents of the tag and the naming convention that was implemented earlier for each individual file.

- date: March 5, 2013
  start time: 7:15pm
  end time: 8:30pm
  phase: Coding
  comment: Started development on the RMI language. Instead of developing this as a module, I have decided to place this in the "includes" folder since it would more than likely be used by other modules. Point of entry is getActions function which would return a data structure of actions that will be updated in the next design document. On a side note, we can use this for user authentication where required instead of a separate request for authentication.

- date: March 5, 2013
  start time: 2:30pm
  end time: 2:45pm
  phase: Testing
  comment: Tested out some shell solutions to multiple file traversal, since Wes was stating he needed to find a resolution for this issue, as discussed in our meeting earlier today. Posted a short "how-to" on the for loop and file acquisition. Wes responded shortly after implementing the code to verify its working condition. It appears to be the solution we were looking for. Considering implemenation in other modules as well as the register-user module.

- date: March 5, 2013
  start time: 7:24PM
  end time: 7:48PM
  phase: Coding
  comment: Worked on updating the shell script to include a for loop to process each file that exist in the incoming directory. Also edited, the route-email ACL2 file to change the directory structure of the email output.

- date: March 7, 2013
  start time: 12:00am
  end time: 2:30am

```
              phase: Coding
              comment: Continued development on the RMI language interpreter.
Developed last few functions getActionString, getAction.  Brackets encapsulate the
actions, and semicolins separate the actions.  SERVERACTION denotes a server action
to take place.  CLIENTACTION denotes that a client action should take place.  At the
time of this file conception, MOVEFILE and COPYFILE actions are only considered.
The portion containing the string between the brackets is the module that will be
considered.  This will acquire the "monitor" value from the module registration on
the server side.  In otherwords, the file that contains the "header" information
will be MOVED/COPIED (depending on action) to the monitor for the module that is in
the brackets, thus invoking the monitor process and kick starting the module itself.
 Files that contain monitor information will end with a file extension of
$unix_timestamp.rmi.  When this file is copied, the header information is removed
and the output is the following XML that is contained in the file
($unix_timestamp.xml).


       - date: March 7, 2013
         start time: 7:30PM
         end time: 8:30PM
         phase: Conceptual
         comment:  Creation of the create-user-request.lisp file. Planning how to
generate XML for user access requests

        - date: March 9, 2013
          start time: 7:59PM
          end time: 9:25PM
          phase: Coding
          comment: Worked on the client code for the email module. I wrote the
functions to parse strings into XML files for outgoing messages. I also handled the
need for the client to sent an email to multiple recipients to where a separate XML
file will need to be written for each recipient indicated in the to field of the
email xml.

        - date: March 10, 2013
         start time: 5:30PM
         end time: 7:30PM
         phase: Coding
         comment:  Completion of createRequests function in
create-user-request.lisp. Creation of the create-user-request.sh script. This script
invokes the create-user-request.lisp file

        - date: March 11, 2013
          start time: 12:04PM
          end time: 12:51PM
          phase: Coding
          comment: I worked on writing IO code for the client side email module.
This included writing code for both incoming and outgoing messages. Since these
needed to be handled differently, there are multiple functions for each of these
requirements.

        - date: March 11, 2013
          start time: 8:13PM
          end time: 9:18PM
          phase: Coding
          comment: I finished the IO code for the client email. It now handles
multiple recipients and will output a single XML file that will need to be processed
to send the multiple email messages. A shell script was written to handled ALL
incoming messages and process them to HTML files for easy reading.

        - date: March 12, 2013
          start time: 11:46AM
          end time: 12:49PM
```

```
          phase: Coding
          comment: The client email code has been finished. Worked on tweaks to get
the output correct and in the correct XML format. Worked on naming of files to
ensure unique naming for each generated file. Finished a shell script to split the
email output that contains multiple recipients.

        -       date: March 13, 2013
                start time: 9:00am
                end time: 10:30am
                phase: Coding
                comment: Began adding shell commands to extract the domain and name
from the registration files to be able to create the directories for the email
storage on the server.

        - date: March 13, 2013
          start time: 5:13PM
          end time: 7:14PM
          phase: Testing
          comment: Tested the client code. Worked on verification of the connection
between the logic module and the IO module. Ensured that the correct functions were
called, and returned the correct structures. This is set up for writing the theorems
for this module. Also added the getHTMLtext to the client logic to allow an HTML
file to formed and written instead of regular plain text for a nicer looking output.

        - date: March 14, 2013
          start time: 6:43PM
          end time: 7:34PM
          phase: Coding
          comment: Wrote the shell scripts to automate the client email processing.
One shell script was made to handle incoming email messages and directs incoming
messages to the inbox folder. The other shell script handled outgoing messages and
places them in the outbox folder.

        - date: March 15, 2013
          start time: 9:46PM
          end time: 10:52PM
          phase: Coding
          comment: Added to the outgoing shell script the capability to parse
multiple emails and create an XML file that contains a single email message with a
unique file name. This shell script will then send the single email script to the
server using the nc command.

        -       date: March 17, 2013
                start time: 12:15am
                end time: 12:48am
                phase: Coding
                comment: Finished adding the directory creation mechanisms to the
shell scripts and tested the user registration process.  It seems to be working
correctly now.

        -       date: March 17, 2013
                start time: 12:50am
                end time: 12:55am
                phase: Testing
                comment: Continued to test the functionality of the user
registration process on the server side.  One thing to note, we cannot have spaces
in our name and domain.  Whitespace has been trimmed.

       - date: March 17, 2013
         start time: 9:30PM
         end time: 10:00PM
         phase: Coding
         comment:  Changed parameters of createRequests function to accept an
```

```
argument for the time stamp of creation used by the create-user-request.sh script

        -       date: March 17, 2013
                start time: 1:00am
                end time: 1:15am
                phase: Coding
                comment: Added and modified lines in the register-user and
address-book lisp files to allow for a user to register with a password.  This will
be used to verify the user can perform actions on the server side.

        -       date: March 17, 2013
                start time: 1:15am
                end time: 3:00am
                phase: Coding
                comment: Created the verify action on the user module.  Defined
utility functions to acquire domain, name, password and the location of the client.
The location will need to be extracted from the original XML from the shell script
since it does not conform the the verification user information.

        -       date: March 17, 2013
                start time: 3:00am
                end time: 6:17am
                phase: Coding
                comment: Created all the functions that will perform the actions on
the mailing list.  subscribe, unsubscribe, and all predicates to allow for these
actions to complete.

        -       date: March 17, 2013
                start time: 6:37am
                end time: 6:45am
                phase: Testing
                comment: After looking at some of the properties from the
address-book_tests file, I realize that I forgot to log these functions into the PSP
logs from last cycle (probably because I did them right before class and lost track
of time).  These have been added to this log.

        -       date: March 23, 2013
                start time: 2:00am
                end time: 2:45am
                phase: Coding
                comment: Added verify-user.sh file to invoke the actions required to
start the user verification process on the server side.  Having a few issues getting
remote-actions.lisp to be accepted into logic (guard checking issues).

        -       date: March 23, 2013
                start time: 12:00am
                end time: 2:50am
                phase: Coding
                comment: Finished the verify-user.sh action file for establishing a
connection between the client and server for mail reception.

        - date: March 24, 2013
          start time: 4:58PM
          end time: 6:18PM
          phase: Testing
          comment: Worked on the theorem suite for the client email module. The
theorems are used to test the data integrity of the logic functions in the client.
We were unable to test the IO functions with theorems since they rely on variant
data. However, the logic could be tested using theorems and we were able to prove
that the client email logic module returns the correctly formatted data based on
correct input.

        - date: March 25, 2013
```

```
          start time: 6:12PM
          end time: 6:49PM
          phase: Coding
          comment: Tweaked the shell scripts for the clients to ensure that the
server has ample time to process a single email message and that the client will not
overload the server by adding too many emails to the queue.

        - date: March 25, 2013
          start time: 7:30PM
          end time: 8:30PM
          phase: Conceptual
          comment:  Creation of the create-block-request.lisp file.  Planning how to
generate XML for requests to block users

        - date: March 25, 2013
          start time: 9:30PM
          end time: 11:00PM
          phase: Coding
          comment:  Completion of the function responsible for generating xml for
creating requests to block users as well as I/O functions

        - date: March 25, 2013
          start time: 11:30PM
          end time: 11:59PM
          phase: Coding
          comment:  Creation of create-block-request.sh script.  This script invokes
the create-block-request.lisp file

         - date: March 27, 2013
          start time: 9:30PM
          end time: 10:30PM
          phase: Coding
          comment:  Creation of create-mailing-list.lisp file. Creation of the XML
Generating functions addressesXML and ownerXML

         - date: March 28, 2013
          start time: 7:00AM
          end time: 8:00AM
          phase: Coding
          comment:  Creation of file output method

         - date: April 1, 2013
           start time: 5:55PM
           end time: 6:47PM
           phase: Coding
           comment: Worked on reworking the Java Gui client to work directly with
ACL2 instead of relying on shell scripts. This allows faster and more secure
connections between remote host for our networking portions.

         - date: April 1, 2013
           start time: 8:59PM
           end time: 11:10PM
           phase: Coding
           comment: Continued to work on the Java integration with ALC2. I was able
to complete most of the Send email functions for the client. All that remains on
this front is sending the file's contents over the network.

         - date: April 2, 2013
           start time: 6:32PM
           end time: 9:48PM
           phase: Coding
           comment: I reworked the client actions into the required Java programs.
They now work with the server and can send and receive information. The scripts that
```

handled the ACL2 function calls were also integrated into the GUI interface and are easily accessible to the user if they are running the interface.

- date: April 2, 2013
  start time: 9:49PM
  end time: 10:04PM
  phase: Testing
  comment: I found a bug with the server code that was already implemented. When looping though all files in a directory, it was picking up extra hidden files as well, This caused the server to crash. I added code to fix this issue.

- date: April 3, 2013
  start time: 5:47PM
  end time: 8:11PM
  phase: Coding
  comment: I finished coding the Java integration for the current ACL2 implementation. The client side of the program can now send and receive emails and User registration. This also includes user verification in order to get email messages.

- date: April 4, 2013
  start time: 4:31PM
  end time: 5:34PM
  phase: Coding
  comment: I added the delete function to the client GUI interface. I also worked on fixing a bug on the transmission of messages. The current issue is that the verification module does not allow for correct exceptions to be processed from the server. If a transmission fails, it does not handle the output correctly and sends an incorrect XML file back to the client.

defect log:

- date: January 18, 2013
  type: Networking
  fix time: 60
  comment: IP resolution cannot occur in ACL2 unless a network drive is mapped, after which you can call it by its network path. Networking drives may be our resolution to this issue.

- date: Jan 22, 2013
  type: Design
  fix time: 60
  comment: Found out that networking is not feasible from within ACL2. To make it natively supported, writing and extending several Common Lisp features would need to be done. We cannot do this in the scope of this project. So we found that using the Operating System's native filesystem and networking support would be much more friendly to deal with once we get to this stage in the project.

- date: Jan 29, 2013
  type: Design
  fix time: 147
  comment: Found out that GUI's and File IO cannot coexist in ACL2. We can have one or the other, but not both. So we had to scrap the GUI portions of the project and replace them with a new idea. The new idea is the current design of the email server and client system. This project is strictly data processing and file IO. This project will be file and text based rather than Visual and Interactive.

- date: January 29, 2013
  type: Conception
  fix time: 150
  comment: ACL2 cannot work with a GUI and IO at the same time. Had to reevaluate how to salvage what we had regarding design. Instead of real time

chat, we would be sending email. Instead of ACL2 interfaces, we would program them in C or C++.

- date: January 31, 2013
  type: Application Support
  fix time: 90
  comment: Unable to get Proofpad to work correctly on any of my computers. Windows 8 and Ubuntu both show NullPointerExceptions when trying to implement ACL2 and Proofpad will not correctly identify with ACL2. Opted to use Dracula instead.

- date: February 3, 2013
  type: Conception
  fix time: 150
  comment: Determined that the size of the executables generated by ACL2 would not be a practical application for our program. Opted to use input redirection into ACL2 prompt and shell script invocation. Ubuntu would be the server platform and the two Macs would be used as clients to send and recieve information.

- date: February 5, 2013
  type: Personnel
  fix time: 15
  comment: With the loss of Isaac from the team, modules had to be prioritized for completion and new due dates had to be set. Determined that I would need to finish the XML Parser as quickly as possible to begin development in order to maintain deadlines.

- date: Feb 5, 2013
  type: Design
  fix time: 124
  comment: We discovered that generating ALC2 executables and invoking these files from an outside source is a troublesome experience and that the generated files are hundreds of megabytes in size. Since we will have several modules for this project, we saw this as a negative side effect of executable files. To solve this problem, we moved all our project to the UNIX platform. This has allowed us to use the UNIX shell environment to generate shell scripts that invoke the ACL2 environment while passing in ACL2 source code files. This reduces the size of the files to kilobytes and streamlines the execution process and eliminating the size of outside programming needed for the original idea to work. Thus we will be executing our ACL2 code through a UNIX shell script and the shell scripts will in turn be executed from the outside programming environment.

- date: Feb 12, 2013
  type: Design
  fix time: 75
  comment: When we looked at the design review. We saw several errors in the XML format that would not pass if it were to be sent though a web browser. We had to work on setting the XML to a correct format and modify the document type definitions to comply with proper XML syntax.

- date: February 15, 2013
  type: Coding
  fix time: 10
  comment: Made the decision to allow shell scripts to take care of much of the IO on the read side as possible and file operations would need to be done by the OS in order to keep correct RWE privs on the file for security purposes. CHMOD properties will need to be determined at a different date, since access to store files has not been completely determined. Best guess is that server will be the only one that needs read/write access to these files and no user group will need execution access.

- date: February 19, 2013

psp.txt

type: Coding
fix time: 20
comment: Issue arose when parsing XML tokens that the whitespace in the document was being identified by #PCDATA token, which was incorrect. This created a case where the next token to be identified was not was predicted thus returning a nil result when processing the address-book xml input. Created a special case that modified around 8 lines of code to check if there exists a whitespace character outside of encapsulating brackets and if so, to ignore those values. Had to remove 4 lines of old predicate for prediction of the next token to be < character.

- date: Feb 22, 2013
type: Coding
fix time: 12
comment: After finishing coding the ACL2 functions, the functions would not admit under normal ACL2 invocation. However, it did work under Dr Racket. The issue was traced to the IO and List utilities files as they were un-certified files within the regular ACL2 environment. Adding the suppression to the certification requirement, the files worked as usual.

- date: Feb 23, 2013
type: Coding
fix time: 52
comment: After fixing the certification issue. Certain functions were still not admitting to ACL2. This was due to illegal arguments as the ACL2 output stated. To fix these arguments, the state variable had to be set and implemented differently than I had intended. I added some safe guards to the variables and added constraints to the functions that depended on them.

- date: March 4, 2013
type: Personnel
fix time: 45
comment: Has to reconsider a new "plan of action" with regards to completion of the project. Current methods seemed to archaic for the current situation as we were unable to adapt due to the relaince on people. Opted for a RAD development solution where SCRUM and Extreme Programming were the foundations for development. Will need to explicitly sit down and speak with team to describe the course of action. Developed a visual aid to describe where we are and what is completed. There seems to be more questions on this as opposed to the completion of the project.

-date: March 5, 2013
type: Coding
fix time: 21
comment: After finishing the IO entry point function on the server email module, We noticed that the XML files were being generated with a comma instead of the @ symbol between names and domains in the email address. Also, the output file for the email was a statically named file. This file needed to be dynamically named with a timestamp.

-date: March 11, 2013
type: Coding
fix time: 24
comment: Email parsing had a one off error that did not have the correct lines returned for the XML file which rendered the outputted file useless.

- date: March 13, 2013
type: Coding
fix time: 20
comment: Was not able to get email to write to the server. Determine that it was an issue that the directories did not exist, thus the information was not being written properly by the ACL2 script, which was a difficult thing to determine since acquiring error information from the runtime environment

would only be possible if we wrote the error to an output log. Determined that I would need to finish the registration process by adding directory creation to the shell script in order to make this function correctly. Until then, we can manually create the directories.

-date: March 14, 2013
type: Coding
fix time: 32
comment: Shell script was not correctly splitting the XML files based on the regular expression. Started using awk to parse the file. However, the file did not have a unique file name and was getting overwritten every time the script was executed.

- date: March 23, 2013
type: Coding
fix time: 35
comment: Type checking for string-listp on XML conversion in the server actions was incorrect. Was using endp and stringp tests, when combined I could use string-listp, which ended up being the required fix and not having to turn off guard checking.

-date: March 24, 2013
type: Testing
fix time: 16
comment: The theorem that tested the email data structure was not passing. This was traced to an error in the proof and not in the code. The error was trying to access an item that was not in the structure, hence the failure of the proof.

- date: Mar 27, 2013
type: Design
fix time: 00
comment: Need to restructure the create-mailing-list.lisp file to handle multiple addresses and multiple owners.

-date: April 1, 2013
type: Coding
fix time: 23
comment: Working on integrating the ALC2 with Java. Having trouble getting Java to see the files that ACL2 has generated. Right now, the current solution is to make the Java sleep for a couple of seconds while ACL2 finishes its processing then resume. Then it sees the files that ALC2 generates

-date: April 2, 2013
type: Testing
fix time: 15
comment: The script that Matthew had written to open all files in a directory was not working. It was needed that the function open only the xml files, since there are hidden files in a directory, I had to modify the function to account for these changes.

Appendix D: PSP File for Additional Features PROBE Estimate

```
name: Team Dijkstra
date: January 31, 2013
program: Additional Features
instructor: Dr. Rex Page
language: ACL2



new objects:

- name: isSecureMessage
  estimated lines: 6
  type: NonIO

- name: postToSecureClient
  estimated lines: 16
  type: I/O

- name: getSecureMessages
  estimated lines: 16
  type: I/O

- name: getSecurityCredential
  estimated lines: 9
  type: NonIO

- name: authorizeSecureClient
  estimated lines: 9
  type: NonIO

- name: registerSecureClient
  estimated lines: 9
  type: NonIO

- name: isSecuredMessage
  estimated lines: 9
  type: NonIO

- name: setSecuredType
  estimated lines: 14
  type: IO

- name: getSecuredOutput
  estimated lines: 18
  type: NonIO
```

```
- name: postSecuredMessage
  estimated lines: 16
  type: IO

- name: inputEncryptionCode
  estimated lines: 9
  type: NonIO

- name: encryptMessage
  estimated lines: 9
  type: NonIO

- name: decryptMessage
  estimated lines: 9
  type: NonIO

- name: openAttachmentFile
  estimated lines: 16
  type: IO

- name: attachmentToBytes
  estimated lines: 18
  type: NonIO

- name: attachmentToXML
  estimated lines: 18
  type: NonIO

- name: addToBody
  estimated lines: 9
  type: NonIO

- name: encryptAttachment
  estimated lines: 18
  type: NonIO

- name: decryptAttachment
  estimated lines: 18
  type: NonIO

- name: xmlReaderForAttachments
  estimated lines: 18
  type: NonIO

- name: sendAttachmentToServer
  estimated lines: 16
  type: IO
```

```
- name: getAttachmentsFromServer
  estimated lines: 14
  type: IO

- name: setAttachmentKey
  estimated lines: 9
  type: NonIO
```

Appendix E: Engineering Procedures and Standards

Overview

The purpose of this document is to define the engineering standards and procedures for Team Dijkstra's Software Engineering II project.

This document will contain information regarding our Source Code Control, Defect Database Design, Coding Style Sheet, Rationale for Language Choice, and updated PROBE Table.

This document will serve as a team reference guide when it comes to actual implementation of the project. This will help keep solidify and help keep our code design and style uniform.

Source-Code Control

To control and maintain our source-code, our group is using Google Code's svn repository. Revision tracking and version control is centralized, and collaboration is simple. Each user has a Google account with which he can checkout the code. Anytime changes are made, svn keeps track and notes it in the repository. Only registered members of the group can upload changes to the repository as a security measure.

Defect Database Design

Each defect will be reported using Google Code's Issues tracker. Each report will consist of the discoverer, a summary, the date, a description of the steps to reproduce the problem, a description of the error, and priority level. Once the issue is added to the database, anyone can read the defect report, work on fixing it, and update the status.

In the defect database, the columns are labeled as follows: ID, Type, Status, Priority, Milestone, Owner, and Summary + Labels. After an issue has been resolved, it will move from being an open issue to a closed issue. The fixed issue will remain in the record for logging purposes. When reporting, there are different labels that users can use to help categorize and organize the defects. First is the type of report. An issue can be a "defect," which is a report of a software defect, or it can be an "enhancement," which is a suggestion for software improvement. There can also be a "review," which is a request for a source code review. The second type of label is the priority level. Priority critical is an issue that must be resolved in order for the project to be functional. Priority high is an issue that is strongly wanted. Priority medium is a normal bug. Priority low is an issue that can be resolved eventually.

Coding Style Sheet

Variable Names

Single Value Variables	*x*	Single valued x.
	x_value	Single valued x.
Multivalue Variables	*xs*	List of xs.
	xs_values	List of xs.

** The names of variables is discretionary. However, the name of the variable should show purpose and application. (Example: tb is not description enough to imply purpose, messageTextbox would be a more appropriate variable name.)

Functions

Data Access	*(getX)*	Acquires the data that the *x* variable contains.
	(setX x)	Sets the value for *x*. *** ACL2 is side-effect free, use let**
Network	*(receiveX)*	Acquires *x* from network resources.
	(sendX x)	Sends *x* to be posted via network resources.
Threading	*(startX x)*	Starts the execution of thread *x*.
	(stopX x)	Stops the execution of thread *x*.
	(sleepX x time)	Halts the execution of thread *x* for *time* milliseconds.
Events	*(getXListener)*	Acquires the listener (handler) assigned to the element *x*.
	(addXListener x)	Adds a listener (handler) to the element *x*.
GUI	*(constructX)*	Visually constructs the rendering for *x*.

** Functions defining so-called "utility functions" are to be named for their purpose with Camelback notation. Example would include (multiplexData xs ys).

Commenting

File Commenting

Commenting headers that are contained within files should be in the following notation:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; /server/utilities.lisp
;
; package server
; Created on January 25, 2013 by Matthew A. Crist.
; Team Dijkstra
;
; This file contains the functions that are required for data processing
; that may not normally be considered a part of the server technology
; but may be classified as a utility function.
;
; FUNCTION
;------------------------------------------------------------------------
; (multiplexData xs ys) - multiplexes the data together into a single ;
;                         string.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; CHANGE LOG:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 0.0.1_20130124 - Initial file conception.
; 1.0.0_20130125 - Release candidate reversioning.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Function Commenting

Commenting functions should echo the function usage, purpose, arguments and return value:

```
; (multiplexData xs ys)
; Multiplexes the list of xs with the list of ys and returns a singular
; list that contains xs and ys.  If xs is greater than ys or ys is greater
; than xs, the remaining members of either list after one reaches length 0
```

```
; is appended to the end of the return list.
;
; xs – the first set to be multiplexed.
; ys – the second set to be multiplxed.
;
; returns – the set of xs and ys where (x1 y1 x2 y2 … xn yn).
```

Inline Commenting	Inline comments are discretionary for the programmer. Comments should be concise and to the point. Over-commenting functions should not occur and combining the purpose for blocks of code into a single comment should suffice.
	
	Appropriate inline commenting:
	
	`; Assigns the first value of the list and rest of the list`
	`(let* (x_value (car xs))`
	`(the_rest (cdr xs)))`

File Naming Convention

Folder Names	Folder names should be named according to the client in which they support. For instance, server files should be located in a subfolder labeling "server" and client files should be in a sub folder labeled "client". Outside resources such as images should be contained within their respective folders in which they apply. Images related to the client will be located in "*/client/images/".* Likewise for server image files: *"/server/images"*.

** For multipart folder names, subfolders should be used: *"/client/gui/images" or "/client/buffer/logs/".*

File Names	The naming convention of files should be done according to the purpose of the functions that it contains. If functions are of utility purpose, we would contain those within a file labeled *"utilities.lisp"*. Likewise, GUI functions should be contained within a file labeled *"gui.lisp"*.

** For multipart file names, underscores should be used to separate words: *"io_utilities.lisp"*. It is recommended, however to use the multipart folder structure instead.

For all code, maximum with should be limited to 75 characters before a new line. New line breaks within the code are at the programmer's discretion, but readability should be maintained.

Rationale for Language Choice

This project will make use of standard ACL2. This section will discuss the reasoning for this choice and why we have chosen standard ACL2 over Modular ACL2.

The main reason for choosing standard ACL2 is the fact that we can compile the source code rather than have it interpreted through Dracula. Running the code this way will allow the code to be executed much faster and efficiently. This will be beneficial when chat logs become extensive and we have multiple clients running off the server program. Since this program has a large I/O portion, the gains from running the executable code is a compelling reason to use standard ACL2.

We plan on using Dr. Racket and Dracula for general development and basic testing. We have chosen this blended environment because of the familiarity with Dr. Racket and the capability of producing executable code from this environment. Another development environment, Proof Pad, was considered, but this environment is primitive and only allows executable code to be ran. We will pass on using this environment due to the unfamiliarity with it and will use Dracula for development and then compiling the code into an executable for running and testing.

The other language choice, Modular ACL2, was not considered since it only runs in Dracula. Modular ACL2 cannot be compiled into an executable and will be slow to execute with our multiple I/O program design. This will become problematic when we decide to use large sets of clients and buffers with the program. Modular ACL2 with Dracula will end up being much slower than running standard ACL2 on the executable code.

PROBE Table

The following PROBE Table is to be used for determining an estimate of the number of Lines of Code for a function. This table was derived from a cumulation of our group members' projects from Software Engineering-I with the addition of the final team project for the semester.

There are four basic types of functions that we will implement. The first is a standard non IO function. Most functions will fall into this category. The second is an IO function. These functions will only be those that open and read/modify a file on the system.

The final two categories are used to estimate the size of our testing suite. The properties category will be used to classify properties and theorems. Check-Expects are the final category and are used for the check-expect statements in the test suite.

The updated PROBE table is as follows:

Function Type	Tiny 7%	Small 24%	Medium 38%	Large 24%	Huge 7%
Non IO Functions	3	4	6	9	18
IO Functions	4	5	9	14	16
Properties	3	4	6	8	14
Check-Expects	2	3	4	8	14