informática

TOWARDS AN INFRASTRUCTURE FOR IMPROVED USER EXPERIENCES

Resumen / *Abstract*
Se propone una experiencia para el usuario más enriquecedora e integrada, basada en la idea de *lifestreams* o flujos de tiempo de vida que fue propuesta primeramente por David Gelernter. En ella las diferentes partes de una experiencia del usuario (mensajes de correo electrónico, temas de discusión entre grupos, paginas Web, tarjetas electrónicas, etc.) son organizadas activamente po la computadora de acuerdo con las reglas definidas por el usuario. Estas reglas permiten que la aplicación *lifestream* sean almacenadas activamente y auxilien al usuario en el procesamiento de los datos, en lugar de su almacenamiento pasivo. Se presenta cómo esta experiencia del usuario puede ser implantada parcialmente mediante el uso de las infraestructuras existentes tales, como JavaMail Application Programming Interface (API), e Internet Message Access Protocol (IMAP *mail servers*. También se propone una arquitectura basada en un software libre (*open-source*), que cuando sea implantada, podría dar una visión más completa y directa del *lifestreams*.

We propose a richer and more integrated user experience, based on the idea of lifestreams o streams of living time first proposed by David Gelernter, where the various parts of a user's experience (email messages, newsgroup posts, web pages, electronic business cards, etc.) are actively organized by the computer according to user-defined rules. These rules allow the lifestream application to actively store and assist in managing user data, instead of passively storing it. We show how this user experience can be partially implemented by using existing infrastructures such as the JavaMail Application Programming Interface (API), and Interne Message Access Protocol (IMAP) mail servers. We also propose an architecture based on open source software, which, when implemented, would more fully and directly implement the vision o lifestreams.

Palabras clave / *Key words*
Lifestreams (flujos en vivo), correo electrónico, servidores de correo electrónico, interacción hombre máquina, arquitectura de máquinas

Lifestreams, email clients, email servers, human-computer interaction, system architecture

Douglas W. Bass, *Graduate Programs in Software, University of St. Thomas, St. Paul, Minnesota, USA*
e-mail:dbass@stthomas.edu
Jeffrey D. Hemminger, *Graduate Programs in Software, University of St. Thomas, St. Paul, Minnesota, USA*
e-mail:jdhemminger@stthomas.edu
Tolga Umut, *Graduate Programs in Software, University of St. Thomas, St. Paul, Minnesota, USA*
e-mail:tumut@stthomas.edu

Recibido:octubre del 2002
Aprobado:diciembre del 2002

1. INTRODUCTION

Human interaction with computers is generally based on some set of analogies or similarities between components of a computer system and some less technical devices. For example, whe the first widely available graphical user interface (Apple Macintosh) was introduced in 1984, the analogies were created that 1. the monitor was like a desktop; 2. the output of an application was like a paper document, and 3. computer directories were like file folders. Graphical user interfaces were generally considered to be an improvement over command line interfaces (DCL, DOS, JCL Unix shells). Graphical user interfaces were considered to be an improvement because, amon other things, they explicitly implemented the above analogies, which were only implied by comman line interfaces. While programmers could equate DOS pathnames with file folders, it was very non intuitive for general users to do so.

While there have been exceptions (most notably, OS/2 Workplace Shell, an object-oriented user interface), graphical user interfaces have become a generally accepted part of computing. We believe computer users focus on the positive aspects of having an explicit implementation of a particular set of analogies. A set of analogies contains a set of implicit limitations. We believe the implicit limitations of the set of analogies implemented by graphical user interfaces have gone generally unnoticed by computer users, to their detriment.[1]

In Section 2, we will examine these implicit limitations in greater detail. In Section 3, we offer an alternative set of analogies, and describe a possible alternative user experience, based on the idea of lifestreams, first proposed by David Gelernter of Yale University.[2] In Section 4, we describe how such a user experience can be implemented using existing computer infrastructures. In Section 5, we describe how computer infrastructures could be modified to more efficiently implement this user experience. In Section 6, we restate our conclusions and suggest directions for future research.

2. IMPLICIT LIMITATIONS OF TRADITIONAL COMPUTING ANALOGIES

When graphical user interfaces were first introduced, an icon of a file folder was used to signify a directory. The fact that folders could be nested inside other folders suggested an analogy between a computer and a file cabinet. The limitation of this analogy is that a file cabinet has no capability of examining its contents, making copies of those contents, and filing the copies in new folders, while a computer does. Therefore, using a computer as a file cabinet, that is, as a passive storage facility, is a waste of a computer's capability to actively manage its contents based on user-defined rules and directives.

Another consequence of the analogy between computers and file cabinets is that there is a many-to-one relationship between files and folders. A folder may contain many documents or messages, but a document or email message is usually filed in one and only one folder. An office worker with a 20-page document wouldn't put a reproduction of the document in multiple folders in a file cabinet, even if those folders were related to the document. It would be a waste of paper and file cabinet space. Instead, the office worker might put the document in one folder, and put a single sheet of paper in the other folders, giving the location of the document. This is essentially what a user of Microsoft Windows does when a document is put in one folder, and shortcuts are put in other folders. But this is a time-consuming process. When a folder is created, the kinds of documents that it is to contain should be defined, and when a document is filed, shortcuts should be placed automatically in all folders to which the document pertains. In other words, there should be a many-to-many relationship between documents and folders.

When a paper document is filed in a folder, the interaction is finished. A paper folder contains no information whatsoever about the items in a folder and their potential interrelationships. A folder contains no information about how its contents have changed over time. It only contains a list of its current contents. This is a waste of a computer's capability for examining and managing documents, and displaying properties of their relationships. A computer folder, or any equivalent unit of organization, should display information about both its contents and the changes of its contents over time.

In summary, there is a set of analogies which have described personal computing for almost twenty years. We believe the implicit limitations of these analogies now outweigh their positive benefits. In the next two sections, we describe an alternative set of analogies and a possible user experience based on those analogies.

3. 1 AN ALTERNATIVE SET OF ANALOGIES FOR PERSONAL COMPUTING

Instead of making a one-to-one analogy between a single computer document and a single paper document, users could consider the outputs of applications, to be like a group with a large number of members, where the number is sufficiently large that the members are not individually named. For example, the outputs of applications could be considered to be like a large herd of sheep or cattle. In a more extreme example, the OceanStore Project at the University of California at Berkeley has considered outputs of applications to be like drops of water in an ocean.[3] Paper documents in file folders don't have names, as what they are is self-evident from their content. The names of outputs of applications will become another aspect which is handled by the operating system, in much the same way that the exact location of a file on a hard disk is handled by the operating system.

If outputs of applications are like unnamed members of a large group, then the computer could be considered to be like someone or something which ensures that the members of this large group go to their intended destination. For example, if the outputs of applications are like a large herd of sheep, then the computer is like a sheep dog, herding them to the desired pen. If the outputs of applications are like a large herd of cattle, then the computer is like a cowboy or vaquero, making sure the herd moves to its destination. If the outputs of application are like drops of water, then the computer is like a plumbing system which ensures water arrives in various repositories.

If outputs of applications are like unnamed members of a large group, and the computer is the agent by which the members of that group go to their intended destination, then the user could be considered as the owner of the group, and the instructor of the agent.

As an alternative to files and folders, lifestreams were proposed by David Gelernter of Yale University.[2] A lifestream is a sequence of various electronic documents (email message, web pages, business cards, Usenet newsgroup posts, etc.), organized by time, and searchable by content.[4] A lifestream is designed to contain a person's total electronic life.[5] Lifestreams are divided into substreams, which are smaller lifestreams organized around a particular topic. The difference between substreams and folders is that once the topic of the substream is defined, documents are added to the substream automatically.

3. 2 A POSSIBLE USER EXPERIENCE BASED ON LIFESTREAMS

When using lifestreams, a typical user would launch a lifestream client. This could be done by either launching the client as an application (as one would launch the email clients pine or mutt in a Linux environment, for example) or accessing a web-based client through a browser (as one would access a Microsoft Outlook web client, for example). A lifestream server would be accessed by standard transport protocols.

The lifestream server would then notify the user of updated substreams, that is, substreams which have had documents added to them since the last session. The documents are added to these substreams by the lifestream server as they arrive. The additions could consist of unread email messages from existing participants in a given substream. If documents from new participants are received, this would be displayed in some manner, just as the host of a social function would like to know when new guests arrive. The additions could be posts to a newsgroup on a particular topic. The additions could be web pages which were visited during the user's last browsing session, or discovered by a web indexer.

The user would spend the rest of the session interacting with various substreams. Interacting in this context means opening, creating, modifying and deleting. Opening a substream involves observing the properties of the substream, such as the number of participants, the amount of new documents and participants, and whether activity in this substream is increasing or decreasing. One of the properties of a substream would be the list of participants in the substream, so that a user could send a message to all the participants in the substream. Opening a substream also involves examining the documents in a substream, responding to email messages and newsgroup posts, browsing web pages, etcétera.

Creating a substream involves selecting a substream name and selection criteria for documents. This would be very similar to what is done when using the advanced search features of a search engine, except that the selection criteria could be applied to certain parts of documents, such as the subject line of an email message, for example. Modifying a substream involves changing either the name or the selection criteria. It should be noted that the user would not have to reorganize the substream after the selection criteria had been changed, as the lifestream server would perform this function. Finally, deleting a lifestream involves deleting a set of selection criteria. The documents themselves would not be deleted, but would remain in the main lifestream for future reorganization.

4. 1 DESIGN ISSUES FOR IMPLEMENTING LIFESTREAMS USING EXISTING INFRASTRUCTURES

For the initial prototype of a lifestream client, we chose to develop an email client that could communicate with existing email servers. While the University of St. Thomas uses Microsoft Exchange Server, a lifestream client could be constructed to work with other email servers, such as Lotus Domino or sendmail Implementing a lifestream client to interface with an existing serve structure posed a number of design issues before development

The first design issue for this project is persistently storing substream properties, so they are not lost when the client application is closed. Lifestream clients give sets of selection criteria to newly created substreams. Folders on contemporary mail services are roughly equivalent to substreams in a lifestream client, but folders on email servers will not recognize substream properties. A substream can contain messages, just as a folde contains messages. The difference is that substreams have properties that folders do not, such as selection criteria, information on the number of documents and participants information on the changes to the contents over time, etc Substream properties can take many forms, but generally guide the behavior of the substream by analyzing messages fo matching properties. If a substream property matches with given message, that message is added to the substream. The current solution is to write the substream properties to a text file called the substreams properties file, and store the file on the client end.

The second design issue is utilizing the folder-tree structure commonly used by email servers. A true implementation of lifestream server would require a single lifestream as opposed to multiple message folders. In a contemporary email server a folde hierarchy is the default setting. This hierarchy will contain folders such as INBOX and SENT MAIL. A lifestream client would contain a single "folder", the main lifestream. This main stream would contain all messages in a time-ordered list. A lifestream client will require a basis for working around the existing folder- tree structure. The solution for this design issue was a class to iterate through the hierarchy in a standard way to view all messages on the server for the particular user.

A third design issue is implementing the many-to-many relationship between messages and substreams. A lifestream client creates substreams that contain messages that also exist in other substreams. The messages contained in substreams will actually be links (URL) to the messages on the server. We envision that 1. the total number of messages in a given substream will be significantly smaller than the total number of messages in the main lifestream, and 2. the number of substreams in which a given message participates will be significantly smaller than the total number of substreams. Therefore, a substream could contain information as to which messages it contains, while a message could contain information as to which substreams contain it

Γhis information would represent an implicit sparse matrix to nanage the relationship between messages and substreams.

The prototype was developed using the JavaMail API.[6] The lavaMail API provided a free, portable, and easy to use API for he development effort. JavaMail creates a session object (an nstance of class session) to authenticate the user to the email server, a store object which contains the server's folders, and older objects which contain email messages. A session object is obtained from the server, a store object is obtained from the session object, folder objects are obtained from the store object, ınd message objects are obtained from the folder objects.

Before developing a client several open source clients were considered, including ICEMail[7] and Pooka.[8] While these two were selected because both were well documented and written in lava, there are a multitude of open source clients written in a variety of languages.[9] Our final decision was to extend a very simple example and focus on the design issues before creating ın elaborate client.

4. 2 A LIFESTREAM CLIENT PROTOTYPE

The client we developed was based on the simple client example available in the JavaMail tutorial.[10] The client allows the ıser to view messages, create substreams, and give substreams properties to match messages. The current prototype allows the ıser to specify an email address and/or a subject line as a substream property. This means that the substreams will look or the specified properties (an email address and/or a subject ine) in the folder hierarchy on the server and in incoming nessages.

The current lifestream client has also the ability to convert existing e-mail server's folder tree structure into a partial lifestream structure when the lifestream client is started. During the mplementation of this conversion, the main consideration is the consistency between the server side folder list and the substream properties file.

When the client is first started and the user is logged in to the server, the application converts folders into substreams. This process consists of three parts: 1. the client creates a list of substream objects by reading the substream properties file; 2. inconsistencies between the e-mail server's folder list and the substream object list are identified, and 3. the inconsistencies are resolved and the substream properties file is updated.

Part 1: Creating the substream object list as the properties file is read

As shown in figure 1, the substream properties file is a text file, in which the names of the substreams and their properties are recorded. As the client reads through the text, a substream object is created for each substream name, and its properties are associated with the substream object. The object is then put into a collection. At this point, it is assumed that the substream is not represented as a folder on the email server. The loop ends when the client reads the EXIT command in he substream properties file.

```
SUBSTREAM inbox/test2        ⎤        → Name of the
substream                    ⎤
                             |
SUBJECT null                 |
CONDITION null               ⎦        → Properties of the
substream
FROM jdhemminger@stthomas.edu

***
SUBSTREAM inbox/test1
SUBJECT java
CONDITION null
FROM jdhemminger@stthomas.edu
***
EXIT
```

Fig. 1 *The substream properties tex file.*

Part 2: Checking for inconsistencies between substreams and server folders

Users can log on to the email server using email clients other then the lifestream client and change the folder list. Therefore, there may not be an exact match between the substreams in the substreams properties file, and the folders on the email server. Three consistency issues have been identified. These considerations and their causes are listed in table 1.

Due to the issues listed above, we took a pessimistic approach (that the inconsistencies will always occur) to enforce consistency between the email server folder list and the substream properties file. The assumption is that the substreams do not exist as folders on the email server until proven otherwise.

The inconsistencies will be identified as the substreams collection is compared to the folder list on the email server. The substream objects have a property which signifies if they are represented as a folder on the server. If the email server has a folder which is not a substream, then a new substream object is created with default properties to account for it.

Part 3: Updating the substream properties file

The third part of the consistency check is to update the substream properties file. This is done when the user ends the lifestream client session. The substream objects which are not represented as folders on the server are removed from the collection, and the substream objects represented on the server are written to the substream properties file.

The functionality provided by the client includes the ability to modify and delete existing substreams. Creating a substream opens a dialog box which prompts for the information required to create a new substream. The user must provide the client with the name of the substream, and zero or more properties from the choices offered (from email address, and/or subject line). Future versions will offer more selection criteria. The client then creates a substream object with the specified properties, and a folder with the specified name on the server. The collection of substream objects is updated. These will be written to the substream properties file when the application is closed.

Once the folder is created, the substream object is sent to the search iteration class where it walks through the server's folder hierarchy to find matching messages. Each folder of messages is matched against the substream properties, and message matches are copied into the new folder on the server. It should be noted that messages were only copied into folders for the client prototype. Future versions will use links to each of the messages. A list of folders can be obtained from the store object. An iterator object steps through each of the folders to check for matches. Matching is very simple with the JavaMail API. The substream properties are strings that are used to create search term objects (instances of class searchTerm). A method of the search term object can be called on a given folder to search through the messages and return a list of message objects with matching criteria. This is a list of individual message objects, and can be iterated through to populate the substream.

Modifying an existing substream can take on many different forms. When a user begins using the client for the first time, it is likely that there are folders on the server created with a different client. These folders become default substreams, or substreams without properties, described earlier when the client checks for consistency. Modifying these substreams means adding properties.

In other cases the user may wish to rename the substream, or modify some of the substream properties. This can all be accomplished via the modify substream option. When a substream property (other than the substream name) is changed, the messages in the substream are matched with the new properties, then the search iteration class is called again to match the new properties with the server folder structure. Messages in the substream which do not match the new criteria are removed.

Deleting a substream deletes the folder from the server and removes the substream name and properties from the substream properties file. The messages in the folder are deleted permanently, as happens on most conventional email clients. Therefore the client eliminates the folder from view, the substream properties are removed from the persistent storage, and the folder is eliminated from the server.

The prototype was successful in dealing with all of the desigr considerations but one. The search iterator tackled the issue o dealing with the server's folder structure relatively easily, and the text file dealt with persistent storage very well. The essentia limitation is the folder structure used by the email server itself While it is possible to create a lifestream client to access a conventional email server, the discrepancies between folders anc substreams create some difficulties. It is easier and more efficien to develop a lifestream server, than it is to develop a lifestrean client to use a conventional email server.

5. AN INFRASTRUCTURE EXPLICITLY DESIGNED FOR SUPPORTING LIFESTREAMS

The next phase of the project will involve implementing a complete email service, to make full use of the benefits o lifestreams. The project will involve creating a web-based clien and mail server to support a single lifestream rather than a folde hierarchy. The first stage of this project will implement an emai server and deploy a web server for the email client. The wet client will be developed with Java Server Pages.[11] JSP technology allows a mix of regular, static HTML and dynamically generatec content from servlets, all in one. JSPs actually generate servlets behind the scene, but it was more natural to write regular HTMI and Java, instead of a large number of Java println statements in a servlet. JSPs have the advantage of being platform independent as opposed to Active Server Pages, and use Java rather thar VBScript- a much more powerful language. Just as many open-source email clients exist, many open-source email servers exist such as RH Email Server,[12] Hamster,[13] and Iloha Mail.[14] We have started a concurrent implementation of the RH Email Serve running on Linux and Microsoft's Exchange Server running or Windows 2000 Server.

There are several options available for open-source wet application servers. Currently we are running Apache Tomca 4.1.0, available at the Apache Jakarta Project.[15] Tomcat is very easy to use and set up, and has worked well as a developmen environment. Other web application servers include Skunl Web ,[16] jo!,[17] and the Jetty HTTP Server.[18]

TABLE 1
Possible inconsistencies between substream objects and e mail server folders

Inconsistency	Cause
The substream properties file can have substreams, which do not exist on the email server as folders	Substreams that exist on the email server as folders can be deleted using other email clients, such as outlook express
The email server can have folders that exist as substreams on the substream properties file	Substreams that exist on the email server as folders have not been deleted during the use of other email clients
The email server can have folders that do not exist as substreams on the substream properties file	Folders can be created on the email server using other email clients

The second stage will increase substream functionality. The :lient will be enhanced so that more selection criteria can be ıpplied to other attributes of email messages. A search engine will be added to index the keywords of all messages. This will ıllow the user to add keywords to a substream property, and hereby search messages based on what the messages are 'about". An example of this is shown in table 2.

In this example the substream would contain links to email nessages from tumut@stthomas.edu that contained the keywords Java, email clients, and lifestreams. The client will assume multiple keywords to be connected by a logical OR, but will give the user he ability to use other operators such as AND, and NOT.

A promising search engine solution is the Apache Lucene project. This open source solution has excellent indexing capabilities (200mb per hour for batch and incremental adds), ınd has been designed to be easily integrated into the email server.[19] The Lucene project was designed to make it simple to ntegrate into such applications as an email service. The search engine will read through messages and create a dynamic index of hem. This index will be searched by the client for substream properties. Matches will generate links from messages into the substreams.

TABLE 2
Some selection criteria for substreams

Property Description	Example
Messages from	Tolga Umut, tumut@stthomas.edu
Messages about	Java, "email clients", lifestreams

Another reason for using Lucene, instead of JavaMail's matching capabilities, is that in the future, our lifestreams will contain outputs from many different applications.

A third phase of implementation will be to add a database to monitor user behavior. This will help to conduct user group testing, and provide an easy form of analyzing use to improve the service at a later date. The concept behind this is to log user behavior, such as clicks, to map how the user uses the client. This type of information will allow us to improve the client with new agents or by developing machine learning to interact with the user. We expect users to find messages which don't fit any substream selection criteria. The user should be able to implicitly modify the selection criteria of a substream simply by assigning an unassigned message to that substream. If a new keyword becomes a frequent part of new incoming messages, the client would offer to create a new substream for these messages. Improvements to the user interface design will also be easily pulled from this database.

In summary, a possible future architecture explicitly designed for the support of lifestreams is shown in figure 2.

6. CONCLUSIONS AND FUTURE RESEARCH

We have shown that the desktop metaphor implicit restricts users 1. to use computers for passive storage as opposed to active management of content; 2. to a many-to-one relationship between application outputs and directories, as opposed to a many-to-many relationship, and 3. to limited vision of the contents of directories. We offered alternative analogies to the desktop metaphor, described an improved user experience based on these alternative analogies, and described an architecture for implementing this experience using open-source solutions.

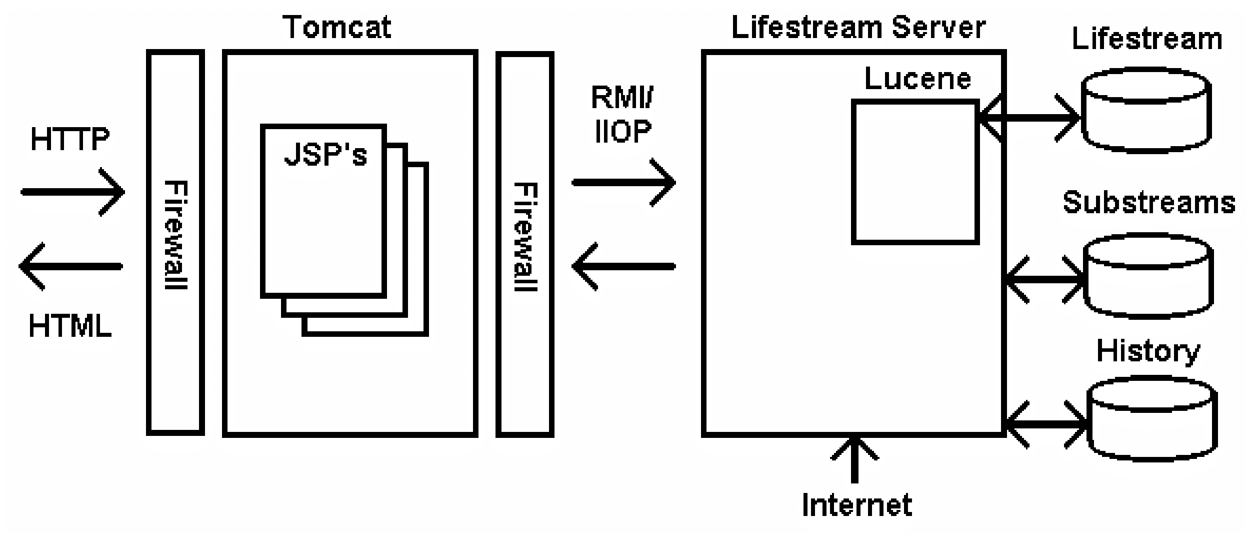

Fig. 2 *An architecture for supporting lifestreams.*

Future research includes 1. the development of clients and servers to fully implement the analogies given; 2. testing the clients and servers with typical users, and 3. expanding the range of application outputs accessible to the clients and servers.

REFERENCES

1. **FERTIG, S.; E. FREEMANANDD. GELERNTER:** "Lifes-treams: An Alternative to the Desktop Metaphor", In*ACM SIGCHI Conference on Human Factors in Computing Systems Conference Companion (CHI '96)*, pp. 410 - 411, ACM Press, 1996
2. **GELERNTER, D.:** "The Cyber-Road not Taken", *The Washington Post*, April, 1994.
3. **KUBIATOWICZ, J.:** "OceanStore: Global Scale Persistent Storage", Frontiers in Distributed Systems Workshop, Aspen, Colorado, June, 2000.
4. **FREEMAN, E. AND D. GELERNTER:** "LifeStreams: A storage Model for Personal data", *ACM SIGMOD Bulletin* 25, 1, pp. 80-86, March, 1996.
5. **FREEMAN E. AND S. J. FERTIG:** "Lifestreams: Organizing your Electronic Life", *In AAAI Fall Symposium: AI Applications in Knowledge Navigation and Retrieval* Cambridge, MA, November, 1995.
6. Sun Microsystems, JavaMail API, http://java.sun.com/ products/javamail/
7. ICEM@IL, http://www.icemail.org
8. **POOKA, A.:** Java Email Client, http://suberic.net/pooka/
9. Sourceforge.net, http://sourceforge.net/
10. JavaMail Tutorial, http://java.sun.com/products/javamail/
11. Sun Microsystems, Java Server Pages, http://java.sun.com/ products/jsp/
12. RH Server Development Project, http://rhems.sourceforge.net
13. Hamster Classic**:** http://www.tglsoft.de/misc/ hamster_en.htm
14. Iloha Mail http://ilohamail.org/main.php
15. The Jakarta Site-Apache Tomcat, http://jakarta.apache.org/tomcat
16. SkunkWEB http://skunkweb.sourceforge.net/about.html
17. Tagtraum Industries, http://www.tagtraum.com/
18. Jetty Java HTTP Servlet Server http://jetty.mortbay.org/jetty/ index.html
19. **GOETZ., B.:** "The Lucene Search Engine; Powerful, Flexible and Free", *JavaWorld*, September, 2000, http:// www.javaworld.com/javaworld/jw-09-2000/jw-0915-lucene.html

2º Encuentro Iberoamericano de Estudiantes de Computación, Informática y Sistemas

ENECIS 2003

Del 17 al 20 de Marzo del 2003
Instituto Superior Politécnico José A. Echeverría y Palacio de las Convenciones.
Ciudad de La Habana, Cuba