- Dynamic addition of TSM's can be accomplished using Java's existing executable-content mechanism.

The proxy side will probably be prototyped in Perl and eventually moved to Posix-compliant ANSI C.

4.3 Future: Mainstream Platforms

Once we have validated the API, we plan to develop a real POSIX proxy and real GloMop drivers for as many platforms as we can, including Magic Cap, Newton, and Windows 95 (WinSock).

notification threshold. For example, a bandwidth drop of 30% might cause a change in the default refinement parameters for images, requesting a smaller sized thumbnail or a shallower colormap, but a drop of 90% would be enough to notify the client.

3.6 Remote Notify

The client can also request notification from the proxy when a remote event of interest occurs, e.g. the arrival of new email or the completion of an agent. We have not yet specified a mechanism for remote event notification.

4.0 Implementation Plans & Current Status

4.1 Phase 1: Sony MagicLink Client, Perl-based Proxy

We have implemented an MH client on the Sony MagicLink, a Perl-based proxy on Solaris, and Perl-based TSM's (type-specific modules) for GIF images and MIME-encoded email. The client is fully integrated with the user's MH inbox, with changes made while offline reflected back to the inbox on next connect. We have designed and implemented (in Perl) an internal API for communication between the proxy and a TSM, described in the document *perl_api.txt*. Client-proxy communication is done over TCP, with the MagicLink using the Berkeley Annex dialin to establish a *telnet* connection on the appropriate port at 2400 baud. (Magic Cap does not yet support SLIP or PPP.)

Currently the MagicLink mail client is monolithic: it implements the client GUI code and the communication with the proxy in a single module. The ultimate goal, of course, is to implement communication with the proxy as a separate system service, with which clients communicate via the API described in this document. The client can handle textual and graphical MIME parts; images are distilled to 4 grays and 480x256 pixels, to match the ML's hardware display capabilities. Details of this implementation are described in the document *mlclient.fm*.

The MagicLink client was written by Steve Gribble, the email and GIF TSM's were written by David Gourley, and the proxy was written by Armando Fox.

4.2 Phase 2: Java

A Java-based mail reader for MH has already been developed by David Gourley, and is being retrofitted to work with this API (which should actually simplify the client considerably). The Java class library includes abstractions for using TCP and UDP over sockets.

The advantages of developing the initial GloMop library in Java include:

- Cross-platform portability.
- We can distribute the demo of the protocol implementation and mail reader using the APP: tag in a Java-aware browser.

It's not clear what should happen on non-threaded systems if GetChunk needs to block.

Chunks spend their entire lives in the Chunk Manager, to avoid unnecessary block copies. The POSIX GloMop implementation will probably not be able to avoid copying: given our desire to run a single persistent GloMop layer to which many applications can connect, GloMop will probably have to run as a separate process with which applications communicate via IPC. However, most PDA's that support object programming systems (including the MagicCap and Newton OS's) essentially pass around lightweight object handles in a shared address space. In any event, the client application always receives a valid handle to the chunk data—whether the copy occurs is a platform-dependent implementation issue.

3.3 Refinement

Refinement is done by calling GetDocument using the document locator embedded in each chunk, and appropriate refinement axis values (or a gross refinement value between 0 and 255). Client applications need not support the display of "embedded" documents (though they can). The way this would appear to the user is that they are viewing a Web page with embedded graphics, they click on a graphic and request a maximum refinement, and a new window then opens showing the refined graphic.

The process of refining a chunk may result in that distilled chunk being broken up into a number of new chunks. A "smart" application could therefore allow the user to select a subregion for refinement, and pass the subregion parameters as refinement axis to the appropriate GetDocument call. GloMop would then request only those chunks (or sub-blocks) corresponding to the selected region.

For example, consider a large (both in physical dimension or in colour depth) image of Soda Hall, that had previously been distilled and refined as a color-reduced thumbnail. The user may select an interesting region of the thumbnail, and request a size (or color-depth) refinement of that region.

3.4 Dispose Document

Dispose Document closes a document's adaptive pipe: any outstanding chunk fetches are cancelled, preloaded chunks never requested are thrown away, and the document is generally forgotten about. In practice, some document chunks may remain cached by the Cache Manager (whose specification is TBD) if there is a strong possibility that future GetDocument calls may reference the chunks.

3.5 Local Notify

The client can request notification of a network event. A network event is defined as some network characteristic exceeding a threshold or moving outside a client-specified window of values.

The default QOS parameters in the GloMop's Network Manager need to encode the automatic behaviors that should be invoked for network events that do not exceed the

(to the extent to which it can be rendered on the client, e.g. modulo color information). Thus, fine control can be exerted over refinement when the refinement axes for a type are well understood (by the human user or the client application, which may provide a series of "dials" in the interface to control refinement), or a gross measure of quality when the axes are not well understood.

3.0 Document Downloading and Callbacks

3.1 Get Document

The GetDocument call supplies:

- a DL for the requested document
- QOS Prefs (See the separate *Quality of Service* document for details)
- The address of a callback routine for handling document arrival or errors
- A callback threshold: the number of chunks of the document that must arrive before the callback routine can do anything useful
- A preload count: the maximum number of chunks that should continue to be retrieved in the background after the callback routine has been called

Note that the preload count and callback threshold are only honored within the GetDocument call. Subsequent calls to GetChunks will either immediately return a chunk that had been preloaded by the GetDocument call, or will result in a single chunk request being transmitted to the proxy.

A chunk of a document is *available* when it has been received by GloMop, *and after the appropriate client-side SSM has done local postprocessing*. For example, "postprocessing" for a GIF might consist of rendering it into a platform-native buffer; "postprocessing" end-to-end-encrypted unformatted text consists of decryption.

The GetDocument call returns quickly with a request ID that is used to identify the adaptive pipe associated with the newly opened connection. The request ID indicates that the request has been successfully transmitted to the proxy , but does not guarantee that the document can be successfully retrieved or relayed back to the client. The callback routine is called when something happens.

A successful call to GetDocument results in the requested document being *opened*.

3.2 Get Chunks

GetChunks retrieves one or more postprocessed chunks of a document whose connection has already been opened by GetDocument. The connection is referenced by the request ID. If the selected chunk has already been preloaded, GetChunk returns it; if not, GetChunk may have to request the chunk from the proxy and block until it arrives. (This is not a problem on multithreaded or multitasking systems; a surprising number of the "lightweight" PDA's, including MagicLink, Newton and Zoomer, support threads.)

now divided into chunks that tile the image. The user can then select a particular chunk (image subregion) for viewing on the mobile device.

To implement refinement, each chunk of a document carries with it a *document locator* that tells the proxy what source document should be opened if refinement is requested (i.e. it points to the source document from which the distilled representation was created). This results in a nice economy of mechanism: the same mechanism used for chunking documents extends directly to refinement.

2.5 Subtype Specific Modules

An SSM is a client-side or proxy-side software module that can transcode a received subtype into something the client can render. For example, the Magic Cap operating system has its own native format for displayed images; transcoding is required in order to display, e.g., a GIF image on the screen of a Magic Cap device.

SSM's are identified by platform name and names of subtypes they transcode between. For example, MagicCap:GIF:ML designates an SSM that transcodes GIF to the Magic Cap's image format; Newton:HTML:Views transcodes HTML into formatted text in a data structure that can be displayed by Newton Views.

SSM's are referenced symbolically and can be added dynamically.

2.6 QOS Preferences

The QOS Preferences structure allows the client to specify desired values of QOS parameters for the transmission of each document. Typically, the client will have a "template" of QOS values for each document type, only occasionally overriding them on a per-document basis. The Network Management layer reconciles these parameters with the statistical model of the network's characteristics, allowing the TSM's to negotiate a document kind for this request. Detailed discussion of the QOS mechanisms is deferred to another paper.

2.7 Refinement Parameters

Refinement parameters are also specified in the QOS Prefs structure. There is a constant, relatively small number of *refinement axes* (exact number TBD), which define the *refinement space*. The intepretation of the axes is specific to each type, and in some cases, specific subtypes of a given type: for example, JPEG allows extremely cheap scaling of dimensions by powers of 2, so one of the refinement axes might be mapped to "power of 2 resolution" for JPEG images.

Each subtype also exports a "gross quality" function that maps points in the interval [0,255] to points in the subtype's refinement space. A value of 0 represents a "handle" (i.e. reference to the data but no actual data; for example, the name of a chapter, or a description of an image such as the ALT tag provides in HTML, along with the document locator necessary to open the original). A value of 1 represents the smallest, coarsest refinement that is still semantically useful, and 255 represents the original document

2.2 Documents and Chunks

A document is a logically whole collection of data as seen by the user. A web page is a document, as is a report with embedded figures.

Each document is structured as one or more logical *chunks*, which may be of the same or differing type, depending on the document. For example, a report document contains some text chunks and some image chunks. The notion of document is abstract and in particular is independent of the subtype encoding of its chunks. The *document table of contents* describes the type of each chunk and may also give a “digest description” of the chunk data. For example, the DTOC for a report might list subsection names and figure captions as the digested descriptions of the chunks.

We say that a document is *open* while the proxy and a client application (really, the GloMop layer on behalf of some application) are maintaining state information about the document. This notion is useful because the client may not wish to retrieve all the chunks of a given document, and may want to do refinement (q.v.) on the chunks it is interested in.

2.3 Document Locator

A *document locator* is a handle that tells the proxy how to retrieve a specific source document. DL’s are specified by extending the HTTP URN syntax. Normally the client will construct a DL to pass to the proxy in order to retrieve a representation of some specific document. The proxy can also pass DL’s back to the client, which can be used by the client to request refinement (q.v.); these DL’s will have been constructed by the proxy for return as part of a chunk, and will have URN’s that look something like: `glomop://`*document_identifier*/*chunk_identifier*.

2.4 Refinement

Normally, each chunk of a document is a distilled representation of the data in that portion of the original document. The distillation depends on the chunk type and various network parameters, and spans a range from the most impoverished representation that is still semantically useful up to the original representation. For example, PostScript-encoded text can be distilled to plain text, plain text plus line break information, formatted rich text (e.g. HTML or RTF), or left as PostScript. A *refinement* is a richer representation of some portion of a chunk: the chunk is the unit of refinement. When refinement for some chunk is requested, that chunk is “expanded” and becomes new, separate document.

The motivation for this approach is that refinement of a distilled chunk may result in an object that is too large to deal with as a single unit and can profitably be chunked. For example, suppose a chunk contains a (distilled) representation of what was originally a large color image. The user may be interested in “zooming in” on a small region of the image, and perhaps improving the color depth on that region. The distilled image was small enough to deal with as a single chunk, but the original image is not. Hence, refinement results in a new document being opened corresponding to the original image, but

GloMop Client API v1.0

GloMop Group
<glomop@full-sail.cs.berkeley.edu>

This internal living document describes the API between client applications and protocol modules in the "GloMop layer".

1.0 Overview: Asynchronous Retrieval of Typed Documents

The mobile client inhabits a document-centric world: the API is centered around the exchange of documents—well-defined, finite collections of bytes. The paradigm is asynchronous document retrieval: the client requests a document, registers a callback function to be called when some predetermined portion of the document has arrived, and goes off to do other work in the meantime. Documents are collections of *chunks*, each of which contains data of a single abstract type.

2.0 Type, Subtype, Documents, and Chunks

2.1 Types and Subtypes

Type refers to the abstract type of a homogeneous collection of data. The total number of types in the world is small: text, image, video, audio, application (code), perhaps a few others in the future. Subtype refers to a *particular encoding* of content: GIF, PostScript, QuickTime. Type and subtype are thus approximately equivalent to MIME type and subtype, though we plan to introduce a number of new subtypes (e.g. subtypes corresponding to native image formats on various platforms) and perhaps some new types that are qualitatively different from any existing types (e.g. hierarchically-layered documents such as maps).