The GloMop Client-Side Architecture

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

1.0 Design Requirements

Because GloMop will run on many diverse platforms that are memory- and CPU-impoverished, we establish the following design requirements:

- "Well-behaved" code that is as platform-independent as possible. Translating between C-like languages should be trivial. Machine dependencies (e.g. memory allocation API) should be isolated into small, easily-rewritable modules.
- Small memory footprint.
- Modest computational requirements.
- Keep It Simple, Stupid: economy of mechanism wherever possible.

2.0 Overview of Design

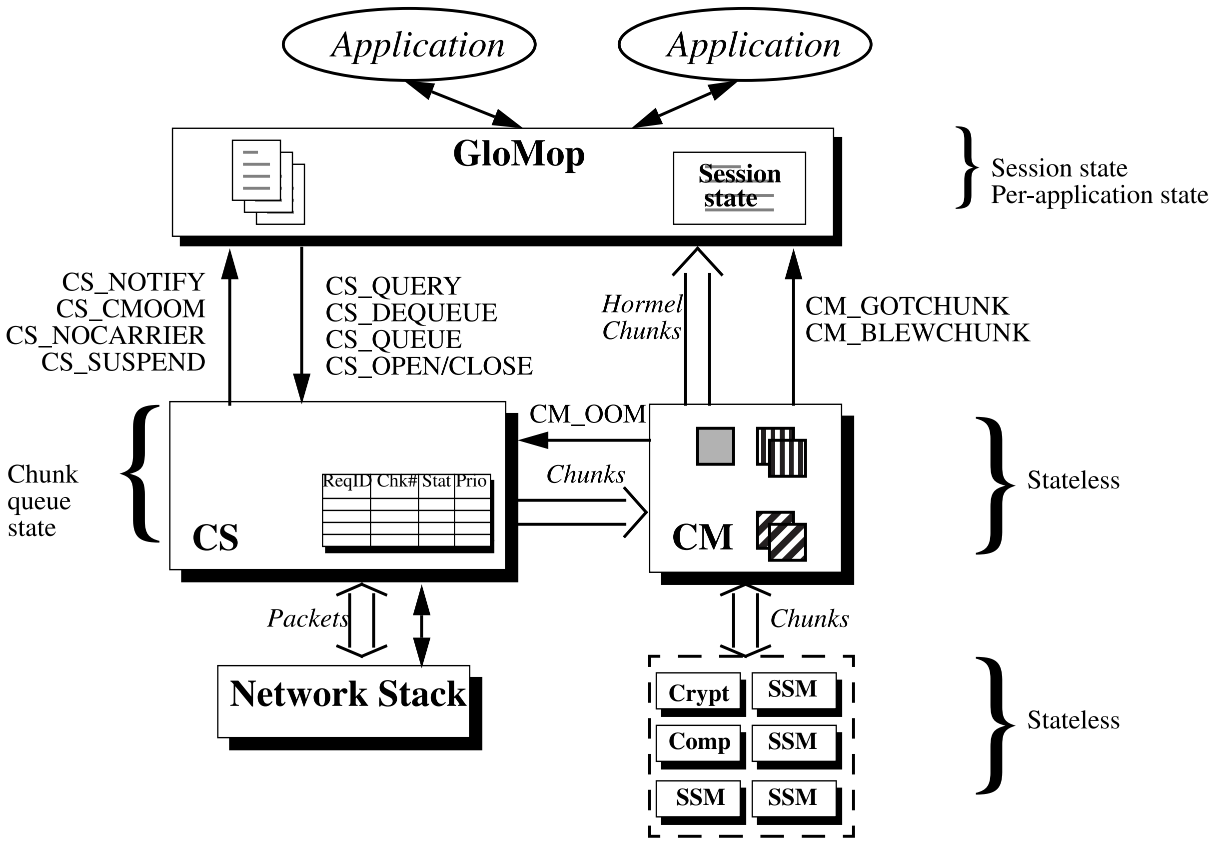
The GloMop components are:

1. GloMop client interface: implements the interface seen by applications, as described in the document *GloMop Client API,* by interacting with the remaining components.
2. Chunk Scheduler: provides chunk delivery and abstraction of the network protocol and interface, and asynchronous signalling of various interesting error and non-error conditions to the GloMop layer (which can forward the signals to applications, if desired). **This component is really closer to the network stack and is being developed by Mark Stemm. The API to this component is still under review and the calls described here should be taken as an approximate functional description, not a complete spec.**

3. Chunk Manager: provides local storage of chunks, transparent postprocessing (decryption, decompression, transcoding to native formats), and notification to GloMop that chunks are ready.
4. Network protocol stack: TCP, or something like it.
5. A future component not shown is the cache, whose contents survive across sessions. Much remains to be worked out for the cache implementation, so we punt on it for the time being.

Figure 1 is a block diagram of the GloMop client side.

FIGURE 1. Overall GloMop client layer architecture



3.0 Notes on the Block Diagram

1. CM=Chunk Manager, CS=Chunk Scheduler, SSM=subtype-specific module. The generic modules *crypt* and *comp* (encrypt and compress) are available for all subtypes. There may be multiple versions of each of these depending on the number of encryption and compression algorithms known by the client.

2. The CM Out Of Memory (CMOOM) error is propagated through the Chunk Scheduler for two reasons. First, CS already requires an out-of-band signalling channel to GloMop, which can be used to deliver CMOOM to the application so it can take appropriate action (or at least display an alert). Second, CMOOM can serve as a hint to CS to stop scheduling chunk requests, since there will be nowhere to store the data.
3. Calls from CS to GloMop, which GloMop can pass on to the client application if the application has asked to be notified of the corresponding events:
   - CS_NOTIFY: An interesting network event has occurred (change of bandwidth, etc.)
   - CS_CMOOM: The Chunk Manager has run out of memory. This is a propagated version of the CM_OOM signal from CM to CS.
   - CS_NOCARRIER: The connection to the proxy was unexpectedly lost.
   - CS_SUSPEND: A handoff is in progress; operation is temporarily suspended. The application need not do anything special, but may want to alert the user.
4. Calls from GloMop to CS:
   - CS_QUERY: Query the status of an outstanding chunk request. Possible replies are Pending, Sent, Acknowledged, InProgress, Flushed.
   - CS_QUEUE: Enqueue a new chunk request.
   - CS_DEQUEUE: Dequeue a chunk request, or all chunk requests corresponding to a particular ReqID.
   - CS_OPEN/CS_CLOSE: Open a connection to the proxy to start a session, and perform two-way authentication. CS_CLOSE ends the session and closes the connection.
5. "Hormel format" refers to a chunk encoding that is ready-to-eat with respect to an application, i.e. a native format data structure.
6. Calls from CM to GloMop:
   - CM_GOTCHUNK: A requested chunk has arrived and has been successfully postprocessed by the appropriate SSM's.
   - CM_BLEWCHUNK: A requested chunk arrived at the CM, but an error occurred while postprocessing it using SSM's.
7. Document state is maintained for each open document; it includes QOS prefs and a record of which chunks have been fetched and which are outstanding for that document. Session state includes authentication, default QOS parameters, and a specification of how to handle various kinds of network state changes (notify user, switch to different default QOS, etc.)

4.0 Chunk Scheduler

CS provides GloMop's only view of the network. It understands the concept of a chunk and can reassemble network packets if necessary to reconstitute chunks. CS manages a priority queue for chunk requests and delivers chunks to the Chunk Manager as they arrive. GloMop can query and delete items from this queue. CS can asynchronously signal certain conditions to GloMop, as described above. CS is also responsible for establishing the initial connection to the proxy, including the exchange of authentication information.

CS performs communication by dealing with an underlying *transport layer* that provides a protocol meeting the following minimum criteria:

1. Packet ordering can be requested if needed. If packets can be 1-2 Kbytes in size, packet ordering will rarely be needed, since a chunk will fit in a packet and GloMop manages data at the chunk level anyway. Possibility: perhaps chunk size should be chosen dynamically by proxy based on network packet (MTU) size.
2. Reliable delivery, or at least reliable error detection.
3. Out-of-band urgent signalling possible.

If multiple network interfaces are being used, CS demultiplexes requests onto them based on request priority, QOS and power consumption considerations.

All protocol details, including flow and congestion control, retransmission, and byte-stream encoding, are hidden from other GloMop components by CS.

5.0 Chunk Manager

When chunks of a particular document have arrived from the proxy, they reside in the Chunk Manager, a local cache within the GloMop layer, until they are delivered to the client application (after postprocessing via an appropriate SSM). Chunks are indexed by the unique ID of the document of which they are a part, and a chunk index (sequence number) within that document. When a chunk is delivered to CM by CS, CM passes the chunk data through the appropriate SSM's if necessary, and then notifies GloMop that the chunk has arrived. Since GloMop maintains per-document state, it can detect when a "threshold" number of chunks of a particular document are ready, and notify the client application. (See the separate document *GloMop Client API* for details on how the client application specifies this threshold to GloMop.) The rationale is that the application doesn't want to be notified until enough of a document has arrived to do something useful with, but the Chunk Manager can be made simpler by not requiring it to manage per-application state.

When a client application is done with a chunk, it signals CM (via GloMop) to *release* the chunk, which frees its memory for new incoming chunks. An application may copy a chunk when the chunk becomes available, and then release the chunk immediately; or it may maintain a pointer into the appropriate CM buffer and manipulate the chunk data there directly (read-only). In any case, after releasing a chunk, the application promises never to dereference that pointer again.

Note that since CM does not forcibly push chunk data to the application, it can run out of chunk memory. Continued operation depends on prudent cooperation among applications. We will see how robust the resulting system will be; the rationale behind this design decision was to avoid forcing an additional copy for each chunk from CM to the application (presumably, chunks already must be copied from CS to CM, since the former probably manipulates a small fixed set of network buffers).

6.0 Decompression, Decryption, and SSM's

The generic transformation library provides commonly required data transformation services such as end-to-end encryption and compression. The chunk manager performs decompression and decryption before returning chunks to the application.

Subtype-specific modules are dynamically loadable software extensions that can transcode chunks into a platform-native format. Their counterparts on the proxy side can transcode source documents into either "standard" or platform-native formats. For example, if the proxy has a proxy-side module that can create MagicCap images, it can transmit images to the PDA that can be displayed without transcoding; if not, it can transmit a "standard" format such as GIF for which the client has an SSM that can transcode to the native format. We have observed significant transcoding latency on the PDA, so a proxy-side transcoder is usually desirable; SSM's allow the PDA to render documents even when the proxy doesn't have the appropriate transcoder.

The separate document *GloMop Proxy-Side Architecture* describes how the proxy determines the target format for a particular document.

7.0 Data Structures

7.1 Chunk

The following describes the encoding of a chunk. The original data contains *m* bytes; the representation in the chunk contains *n* data bytes, where *n* may differ from *m* if end-to-end encryption or compression is used. Multibyte quantities are stored in network byte order. Strings are stored as a length byte followed by up to 254 data bytes; a length byte of 255 indicates a string longer than 255 characters with a terminating null.

Comments: Do we need 4-byte fields? Is 64K bytes enough for an individual chunk and for the ReqID and sequence number?

TABLE 1. Chunk Encoding

Field	# bytes	Data
ReqID	2	Request ID of document of which this chunk is part
Chk	2	Chunk number within document
Type	s+1	Chunk type/subtype encoding as string
RawSize	2	Number of bytes to follow
DL	s+1	Doc locator for refining this chunk, as string
MDS	2	Number of bytes of chunk-specific metadata following; will usually be zero

TABLE 1. Chunk Encoding

Field	# bytes	Data
MD	*s*	Chunk metadata
EncrType	2	Encryption type, 2-letter string; 00 means none
CompType	2	Compression type, 2-letter string; 00 means none
Osize	2	Size of data after decryption & decompression
Dsize	2	Number of data bytes to follow
Data	*n*	Chunk data

8.0 Unresolved Issues

- Should document uploading be incorporated into this overall design, or does the asymmetry of document uploading imply the presence of additional components? (Certainly the SSMs, the transport layer, and the generic transformation library can be reused.)
- As always, how do time-critical stream documents fit into this scheme?
- How many threads are required within GloMop? One to handle requests from applications; one for the Network Manager; maybe one in the low-level network stack; any others? The Network Manager thread can be used for upcalls into GloMop.