the architecture and design were more fluid, and we were learning how to deal with Spring and Hibernate. It helped homogenize Eclipse practices and tricks, too.

The projector was also handy for iteration demos. We could have all the stakeholders in the room, without crowding around a single screen.

(Not to mention how helpful it was for projecting funny YouTube clips up on the wall.)

I knew it wouldn't be hard at all to build something fast but fragile. The real challenge would be making it robust, especially when the whole network would exist in a studio hundreds of miles away. One with no ability to log in remotely to debug problems or clean up after failures. One with small children, distracted parents, and servers sitting at toddlers' eye level. Talk about a hostile environment! Moving bits across the wire would not be enough; we needed atomic file transfer with guaranteed delivery.

The first layer of defense was the protocol itself. For a "put" operation—uploading a file from client to server—the first packet of the request includes the file's MD5 checksum. Once the client sends the last packet, it waits for a response from the server. The server responds with one of several codes: `OK`, `TIMEOUT`, `FAILED_CHECKSUM`, or `UNKNOWN_ERROR`. On anything but an `OK`, the client resends the entire file in what we call a "fast retry." The client gets three fast retries before the transfer fails.

Problems with file transfer will come in two varieties. One type is the "fast transient," a quick problem that will clear itself up, such as network errors. The other type requires human intervention. That means problems will either be cleared up in a few milliseconds, or they will take minutes to hours to correct. There's no point in retrying a fast file transfer over and over again. If it didn't work after the first few attempts, it's not likely to work for quite a while.

Therefore, if the client exhausts all the fast retries, it puts the file transfer job in a queue. A background job wakes up every 20 minutes looking for pending file transfer jobs. It tries each job again, and if it fails again, it goes right back into the queue. Using Spring's scheduling support made this "slow retry" almost trivial to implement.

This mix of fast and slow retries lets us decouple maintenance and support on the server from the clients. There's no need to "cold boot" an entire studio for upgrades or replacements.

Fast and robust

The local and remote image repository and their associated file transfer mechanics became a seriously tough slog. Once it was done, though, the whole thing could upload images to the server faster than they could be read from the memory card. Downloading them on another machine was fast enough that users never perceived any activity at all. The client would download all the thumbnails for an album during the transition from one screen to the next. Downloading the screen-sized images for full-size display could be done during a mouse click. This speed let us avoid the user frustration of "loading" dialogs.