A Web-Services Architecture Designed for Intermittent Connectivity to Support Medical Response to Disasters

Steve Brown¹, William Griswold, PhD¹,², and Leslie A. Lenert, MD MS²,³
University of California, San Diego, CA, Department of Computer Science and Engineering, ²California Institute for Telecommunications and Information Technology and ³Veterans Affairs San Diego Healthcare System, San Diego, CA

To support mobile computing systems for first responders at mass casualty sites, as part of the WIISARD (Wireless Internet Information System for Medical Response in Disasters) project, we have developed a data architecture to gracefully handle an environment with frequent network failure and, multiple writers that also supports rapid dissemination of updates that could be critical to the safety of responders. This is accomplished by allowing for a subset of the overall information available in a disaster scene to be cached locally on a responder’s device and locally modified with or without network access. When the network is available, the local subset of the model is automatically synchronized with a server that contains the full model, and conflicts are resolved. When changes from a device are committed, the changes are instantly sent to any connected devices where the local subset would be modified by the changes.

The WIISARD project aims to enable responders at disaster scenes to handheld and tablet computers to coordinate care at a mass casualty site. Mass casualty settings are far from ideal conditions for a network, so a system must be designed to expect network problems. When the WIISARD system is deployed, networked portable 802.11b access points are distributed around the scene. Coverage is not expected to be universal and temporary problems with network network are anticipated. These issues should not prevent the responders from using WIISARD applications to care of victims or impair the performance of interfaces.

To achieve this end we are developing a architecture based on mediation of connections with web applications by client access libraries. A client access library exists on the responder’s device and acts as the gateway between the applications and the server, reducing the complexity of the applications. Applications are users of the client access library and need little to no knowledge of the origin or state of the data they are working on. The client access library is responsible for handling the details associated with opportunistic connection to the server, synchronizing and persisting of the data model and local modifications, and notifying applications of changes to the data model and conflicted local modifications. Its three main pieces are a Simple Object Access Protocol (SOAP) conduit for connecting to the server, a local subset of the full data model on the server, and a command queue for recording local commands that are not yet committed to the server. The command queue contains command objects that are instances of the command pattern. As the local data cache is modified by the application, command objects are generated and added to the queue.

The server component of WIISARD has three main pieces: a conduit for SOAP access by clients such as the client access library; the application server, which accepts and processes incoming commands via the SOAP conduit and listens to and forwards the state of objects the client is interested in; and the database, which is the base repository for the data in the system, that also logs and notifies the server of changes made.

To prevent command replay due to the network connection being lost before acknowledgement of a committed command reaches the client and the client subsequently resubmits the SOAP messages are required to have a unique identifier for the session assigned by the client before sending. In that way, the server can reject messages it has already seen.

While there is a network connection, these command objects are executed on the server and removed from the queue when they are acknowledged and committed. When updates are sent from the server to the client, these queued commands are undone, the updates are merged, and then the commands are replayed. Failures in this step signal a conflicted local modification and are presented to the application for resolution. Data that is known to exist but not present on the local device is represented in the local model and can be mapped by the application as needed.