

Designing an Effective Data Synchronization Model for Collaborative Mobile Software

Kevin McGehee

Montgomery Blair High School
51 University Blvd East
Silver Spring, MD 20901 USA

Jerry Alan Fails

University of Maryland
Human Computer Interaction Lab
College Park, MD 20742 USA

ABSTRACT

I investigated the design of an efficient and effective data synchronization model for use in software known as the *Storyteller*. The *Storyteller* is a children's collaborative narrative creation system developed for Windows Mobile devices. The *Storyteller* software required a model that would allow multiple devices to interact wirelessly, enabling each device to share access to a particular digital “story.” My implementation was written in C# using the Microsoft .NET 2.0 Application Programming Interface (API) and utilized a client-server model. Communication between the devices is handled using TCP/IP socket communication. Virtual data objects within the program are transferred using efficient binary serialization methods that minimize network overhead. Case studies evaluating the *Storyteller* software as a whole have shown that my current network model is relatively reliable and robust.

INTRODUCTION

Lilly and Josh are working together on a creative writing script for their film class. They have been keeping individual brainstorming lists on their own laptops, occasionally sending their lists to each other or, during class, consolidating and merging them. It is even more inconvenient to keep separate copies when they begin writing the script itself. So when Lilly and Josh have time, they add to a particular section and send the full script to the other for approval. When they are in the same room, they can work on the same laptop, and can discuss and add text to the one master copy. In both situations, only one master copy of the script exists at any point in time. But what happens if Lilly and Josh simultaneously decide to add to the same section? The two copies would need to be merged and the process may be tedious, or at least uninviting. And how often should they send their copy to each other? Should they still send it even if they take a break, but are in mid-sentence?

The general public is able to take advantage of digital collaborative workspaces that allow people to interact across vast distances. Employees telecommute and gain access to important files while on the go; friends and families can use communication software to see and interact with each other in real time; anyone connected to the Internet can use wiki software to contribute to and gain knowledge from a global database, such as in *Wikipedia*. Such methods of collaboration have not been heavily researched as they apply to children. However, children are not exempt from the benefits of digital collaborative interaction. Research done in the Human Computer Interaction

Laboratory (HCIL) of the University of Maryland found that digital interaction “may be very appropriate” when used during field trips with younger children (grades K-4), emphasizing the importance of mobility in such peer interaction (Chipman et al., 2006). The researchers (my mentor and I) together produced an application called the *Storyteller*, a piece of software that runs on the Windows Mobile platform which enables children to create digital narratives using pictures, sound, and text. Using this software, children are able to construct stories and share them wirelessly with other children using similar mobile devices, working together to produce a joint story. One concept behind the product is that it encourages children to *produce* content which deviates from most children's portable devices on the market today that focus on *consuming* content. In addition to simply allowing children to create stories, the product encourages children to work together and learn from each other while composing a narrative. The goal of the joint research was to determine the optimal collaborative methods for use with children as well as the efficacy of this type of product for a mass market.

In order to research these collaborative interactions among children, a framework was required that would support multiple connected devices and would allow individual devices to communicate with each other. The problem of a framework was even more challenging when the program had to run on mobile devices, which have limited processing power and memory. The objective of my research was to create a solid foundation for such collaborative applications. I produced a framework capable of running on these mobile devices and dealing with the intrinsically variable

wireless network range of mobile devices while still retaining the information needed to sustain a digital children's story. The proposed network component of the project utilizes a server-client framework for sharing the data model. The framework transfers data over a custom socket connection. Program data is serialized over the connection using an efficient custom binary serialization schema.

BACKGROUND

Collaborative frameworks are not new innovations; many products have been created in order to allow people to communicate with each other while working on a digital document. One such product is Google Docs & Spreadsheets, an online office suite produced by Google. Using a web browser, multiple individuals can simultaneously access a single file over the Internet, revising the file with each other in real time. The interface contains not only a chat window so that the collaborators can communicate if not physically located near each other, but also has a revision history of who changed which part of the document for tracking purposes. With this system there is only one master copy of the document on the server, so clients cannot work on it offline and have their changes take effect when they come back online, an integral requirement of my research (*Google Docs & Spreadsheet*, n.d.).

Another collaborative framework is Concurrent Versioning System (CVS), used with many programming projects to keep track of source code. CVS keeps a master copy of all data included

in a collection of files and anyone can “check out” the files to their own local copy. After editing their own copy, users can “check in” changes which will be saved to the master copy. CVS also includes utilities to help merge conflicts that may occur if two people have made edits to the same area. Conflict merging only works with text files, where changes can be readily seen, not with formats such as pictures or audio. This principle needed to be extended to work with a data structure containing a full story in the *Storyteller* project.

Data synchronization is also not a new concept. Large databases such as MySQL depend on synchronization techniques to establish and maintain redundant mirrors, but in most cases, only one database is able to accept changes, making such techniques ineffective for a collaborative interface with multiple clients. In this configuration, the master database server propagates new information to the mirror slave servers by sending only the necessary commands to update the old information, which minimizes the amount of network traffic while still relaying the update ("Replication," n.d.).

A concept that has not been thoroughly explored is implementing a collaborative environment on mobile devices. Devices may not always be connected to the other devices involved in the document sharing processes, making the merging process even more complex.

PROBLEM

My research outlines the design of a data synchronization model that allows multiple people using mobile devices working on the same set of information to collaborate and share the information efficiently and effectively across a wireless network with smooth transitions into an offline working mode and back again. For example, in an online file sharing program, each computer working on the file has its own copy downloaded from another user. But if one user wanted to work on a document with another user or multiple users at the same time, the communication between the separate computers must be managed in order to ensure that all people are looking at and editing the same document simultaneously.

The hardware requirements were a concern for the purposes of the *Storyteller* project. The researchers made the decision that the mobile device that would be most appropriate for controlling a program requiring complex interactions would be a Pocket PC. Pocket PCs are small, light, and have built-in wireless network capabilities and touch screens for easy data manipulation. The program was required to work on Pocket PC devices that were available to the lab, and most of these devices ran the operating system Windows Mobile 2003. The server had no direct hardware limitations.

In addition to running on the Windows Mobile platform, the program had to run on a traditional Windows XP machine which was designated as the server computer; thus, the *Storyteller* required

a language that was cross-platform. C# was chosen because it was leveraged by the powerful .NET 2.0 framework, an API that was high-level enough to allow quick prototyping of features but powerful enough to provide flexibility in our implementation. My component was also written in C# so that the program itself could communicate with clients across the wireless network.

DESIGN APPROACH

In designing and developing the *Storyteller* project, an iterative design process was used. There were two primary stages of design with an informal performance evaluation following each design stage involving the use of a volunteer team of children participating in a summer day camp at the lab. The design and theory evolved over the course of the project, with major design decisions and subsequent revisions.

My contribution to the *Storyteller* project can be broken down into two main subcomponents of the data synchronization architecture: serialization and transmission. In order to transmit a virtual program object in the *Storyteller* program (for example, a page in a book with a picture and text), the data must first be serialized, or converted into a format that can be sent over a network connection or saved to disk. After the object is serialized, the device must then have a protocol to transmit this serialized data stream to its intended recipient.

INITIAL DESIGN

First, a basic communication model for the devices needed to be chosen; most notably, whether the devices would operate as independent nodes on a peer-to-peer network, or through a central server. A design with a central server was chosen for several reasons. This design reduces the problems associated with maintaining multiple connections to multiple devices. The content on the server prevails, and any conflicts that the clients have are resolved via a central location. A central server also makes the job of logging and behind-the-scenes management of the network much easier than in a group of independent clients, as a change to the server can easily influence all the connected clients. The architecture thus consists of two main components: multiple clients and one server, as seen in *Figure 1*. All interaction among clients takes place through the server; no client directly addresses another client.

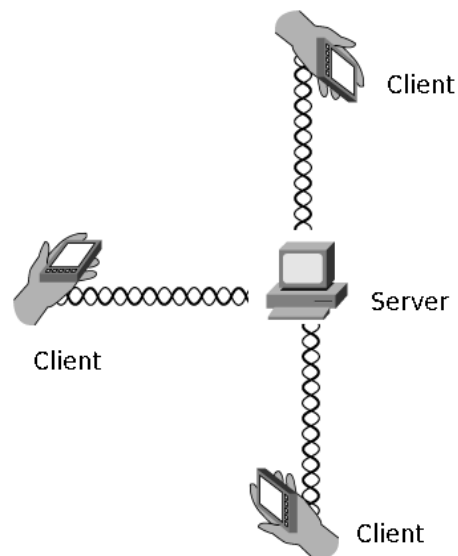


Figure 1: Sample network diagram ("Wireless clients connect," n.d., modified by the author).

Because of the researchers' decision to use a server-client framework, the server must always maintain the master copy of the data structures, but the clients must be updated in a way such that the acquisition of new data does not interfere with the user interface and user experience.

The initial design incorporated an existing Windows API called Microsoft Message Queuing (MSMQ) in order to both serialize and transmit data between the clients and the server. This API “enable[d] applications running at different times to communicate across heterogeneous networks and systems that may be temporarily offline” (*Microsoft Message Queuing*, n.d.). This method of communicating appeared to meet our needs: it worked on both the clients (Pocket PC devices) and the server (Windows XP) and allowed messages to be queued in a fashion that they would be sent when a network connection was established, perfect for the transient device connections.

The MSMQ API allowed for messages to be serialized with Extensible Markup Language (XML), a “simple, very flexible text format” that allows “the exchange of a wide variety of data on the Web and elsewhere” (*Extensible Markup Language*, 2006). One advantage of XML is that objects that have been serialized can be easily inspected before they are deserialized. For example, a story that is being developed using the *Storyteller* program can be written out in such a format that it can be viewed in a web browser to check for consistency and changes. Using a traditional binary serialization format, this cannot occur because only the program knows how the data is written out to the file, and usually it is not in a human-readable format. The disadvantage, however, is that in

most cases XML takes longer to read and write and uses up more space than an equivalent binary serialization scheme. However, since MSMQ did not support binary serialization on mobile devices, the networked objects were designed to be compatible with XML serialization. Some types of objects, though, could not be converted into a text format; pictures and sound were thus transferred independently of the rest of the communication using a separate binary protocol.

Upon testing this in an introductory session with the children around the A.V. Williams Building in the University of Maryland, I found the MSMQ messaging system to be unreliable and insufficient in its ability to quickly transfer information from the clients to the server. Even when the clients had a strong wireless signal, the queued messages of the client activity sometimes took several minutes to send or did not send at all (and the device had to be restarted). Thus, the server was not notified of the individual changes in an acceptable period of time and the information did not propagate to other users effectively. One such example was of a child, Sam, who typed a story into a text field in the program only to have it not appear on the server's copy because his device never sent the update to the server. All of the children were disappointed when their individual changes were not propagated or saved.

Problems also existed due to the two separate systems for communicating data; occasionally, a message referencing a data file such as a picture would be received and executed before the corresponding data file was sent to the device. This caused the devices to wait for the data file

upon receiving the message, causing information to travel slower than optimal between devices.

REVISED DESIGN

After the initial design session, the MSMQ system was discarded and a novel network section was written. For transmission, I wrote a server-client interface to establish connections between nodes. All nodes know how to send data to another node; outgoing messages establish a connection to the destination node. The architecture, however, must be able to deal with transitioning between online and offline states, and no guarantees can be made about a node's connectivity at any point in time, except the server's, which is always connected. Therefore, messages are queued to be sent and the queue is regularly attempted to be cleared, thereby sending messages sequentially to the destination node. This allows messages queued on the server to an offline client to be immediately sent when the client comes back online. Likewise, if any changes are made on the client in an offline period, they will be queued and sent as soon as the client regains network connectivity.

Because I was now dealing with raw TCP/IP connections, I needed another way of serializing the virtual data objects. I chose not to use an XML-based method, but instead a third-party class called *CompactFormatter* that enabled the use of an efficient custom binary serialization schema (Scotto, 2005). Although this binary format was unable to be easily inspected without the program, I had hoped that it would allow the messages to be sent and interpreted quicker than a corresponding XML message. Because I was no longer bound to a text-base format, I could

transfer data objects such as images or sound files in the same way as the other transferable messages. I rewrote the way data objects were stored and handled so that any data object was kept solely in program memory and did not have a corresponding file on disk.

The main performance evaluation of my project took place at Fort McHenry National Park and Historic Shrine located in Baltimore, Maryland. Two sessions using the *Storyteller* technology were conducted. In the morning, three pairs of two children were given a Pocket PC device with the beginnings of a shared digital narrative, a “story starter,” as shown in *Figure 2*. The children explored the park and recorded observations using the *Storyteller* software. The wireless router connected to the server was located in a centralized location such that the areas explored by the children would have wireless connectivity throughout the time period.



Figure 2: A desktop view of a "story-starter."

The networked session lasted for approximately 70 minutes, with usage by the children for about 35 minutes. During this time period, the server logged all activity and regularly saved a copy of the current state of each story. In total, the server accepted 66 changes to the story. Data in each scene was successfully changed, indicating that the attachment of the data files to the change messages worked relatively well. However, in a few instances, the server did not receive a copy of a data file. In these cases, the server attempted to save the data file to disk but was unable to do so; the file was then likely never transferred to any other client.

A total of 176 attempts to communicate with a disconnected client were made by the server. Most of the disconnection times of the clients were less than 30 seconds and were caused by periodic variability in the wireless signal within the visited buildings. One group, however, went outside of the designated area within wireless coverage for about two and a half minutes and re-joined the session without any trouble. At 12:46 PM, one device stopped responding to any messages, corresponding to the two children's departure from the wireless range. It regained the connection at 12:48 PM with three queued messages that were successfully delivered within the first few seconds that it started responding. The children with this device then proceeded to add the words to the National Anthem to a page, and the changes were made without any trouble, just as if the device had never disconnected from the network.

In the afternoon session, two older children, Sam and Jonah, participated in a more open and

slightly more complicated activity. They each had a Pocket PC and could switch between three separate copies of a scavenger hunt “story-starter.” They worked against each other to identify each of several items that were described or pictured on the digital pages. The data synchronization model withstood a higher density of traffic from each device than in the morning. In a little over a half hour, 68 changes were made to two of the three stories. Jonah consistently attempted to sabotage Sam's work. For example, at 1:52 PM, Sam recorded a sound for one of his scenes, and at 1:58 PM, Jonah recorded over it. Eventually, the activity broke down when the children were unable to concentrate on the task, but the network remained operative. The main defect identified in this activity was the need to revise the collaborative methods in which the children were able to interact so that one child could not erase another's work at the touch of a button.

DISCUSSION

My proposed model has shown to be reliable and robust. Multiple Pocket PCs were able to run the *Storyteller* with many large multi-page stories, illustrating that my model was able to deliver the content of dense stories with an acceptable use of memory on the devices. Children were able to use the devices to collaborate on a story in a real-life mobile setting, and changes that were made on one device were successfully transferred to others, showing that the framework does execute commands successfully. One device transitioned offline for a period of several minutes and then back again, which showed that the current model can handle interruptions in network connectivity. My component of the *Storyteller* software enables it to be used to carry out truly unique research

that will benefit children's educational environments by providing a framework for developing novel collaborative interactions. The *Storyteller* software without collaborative methods is just a digital storybook, but with my data synchronization model, it will yield unique and beneficial information about the effect of collaboration on children's education. Initial research has shown that digital collaboration is beneficial for children, and with a streamlined synchronization model, further research can be done to determine optimal types and methods of digital collaboration, bridging the gap between the digital and physical world for children (Chipman et al., 2006).

CURRENT AND FUTURE WORK

There were several major changes that took place as a result of the evaluation at Fort McHenry. The *CompactFormatter* class would not correctly transfer complex recursive data types; a temporary fix was implemented at the last minute for the evaluation at Fort McHenry. As a result, I wrote serialization methods that mimicked the functionality of the *CompactFormatter*, but were specialized for the purposes of the objects the *Storyteller* serialized and deserialized. Although not as easily extendable as the *CompactFormatter* implementation, my implementation removed the overhead of a third-party library dependency. In the future, this simple class could be extended to be more robust and easy to use.

The other major change was in the handling of data file objects. As shown in the data obtained from Fort McHenry, sometimes data objects would not be transferred correctly when sent as a

stream within the messages, and quite often the program would run out of memory as it was forced to hold large bitmaps in its RAM. Therefore, I revised the handling of data files to maximize the efficiency of receiving and sending data objects at the cost of speed. Objects now read and write directly to files, and only load into memory when they are actively called within the program. This may cause a data object to take a bit longer to become accessible, but it ensures that no memory is used storing objects that are not currently in use. However, a more ideal solution would include a uniform method to reference and access files that are on the hard drive of any of the individual devices and ensure that the files are the same.

The need for a more appropriate method to resolve conflicts was identified. Conflicts may occur because the program is required to work both connected to and disconnected from the network that contains the server. There are scenarios in which story data could be unintentionally overwritten due to the way the changes are submitted to the server from the client. For example, if Client A goes offline and changes a text field while Client B remains online and changes the same text field, the changes submitted when Client A reconnects to the network would override the changes made by Client B. Included in the current release is a detailed conflict resolution protocol that makes sure that if a conflict occurs, all clients end with the same data model without interrupting the users' experience with the program; in previous versions, the last submitted change would override any previous changes. The conflict resolution manager will first display to the client with the conflicting update that his or her last update conflicted with the server's version. It will prompt the

user to either “branch” the story, effectively creating a copy with their overridden changes, or to let the server decide how best to apply the update. By refusing to branch his or her own version of the story, the user forfeits his or her work, and unless a clean update can be applied (an update in which no new work is overwritten), then the change is dropped. For some types of changes to the data model, the conflicting client's change can still be made; in other cases, it is simply dropped and the client will revert to the server's copy. More work must be put into the conflict resolution methods in order to determine how effective this procedure is with children, and whether or not an alternate method would foster a greater collaborative initiative. The option to branch a story may confuse the children, or they may still unintentionally lose their work; research studies including these collaborative methods should be conducted in order to determine the usability of such an interface.

ACKNOWLEDGEMENTS

I would like to thank Jerry Fails for his mentoring and support throughout my research. Without his creative ideas and initiative, the *Storyteller* project would not have been made a reality. I would also like to thank the entire staff at the Human Computer Interaction Lab for their support of our project. My gratitude is extended to Glenda Torrence, my high school research advisor, who has guided me through the research project process.

REFERENCES

- Chipman, G., Druin, A., Beer, D., Fails, J. A., Guha, M. L., & Simms, S. (2006, March). *A case study of Tangible Flags: A collaborative technology to enhance field trips*. College Park, MD: University of Maryland, Human Computer Interaction Lab. Retrieved November 6, 2006, from University of Maryland, Human Computer Interaction Lab Web site:
<http://hcil.cs.umd.edu/trs/2006-03/2006-03.pdf>
- Extensible markup language*. (2006, September 11). Retrieved November 6, 2006, from World Wide Web Consortium (W3C) Web site: <http://www.w3.org/XML/>
- Google docs & spreadsheet tour*. (n.d.). Retrieved November 6, 2006, from Google Web site:
<http://www.google.com/google-d-s/tour1.html>
- Microsoft message queuing*. (n.d.). Retrieved November 6, 2006, from Microsoft Corporation Web site: <http://www.microsoft.com/windowsserver2003/technologies/msmq/default.msp>
- Replication. (n.d.). *MySQL 5.0 reference manual* (chapter 6). Retrieved November 6, 2006, from MySQL AB Web site: <http://dev.mysql.com/doc/refman/5.0/en/replication.html>
- Scotto, A. (2005). *About the compact formatter*. Retrieved November 6, 2006, from <http://www.freewebs.com/compactFormatter/About.html>
- Wireless clients connect to the LAN via an AP. (n.d.). *Cisco 802.11 wireless networking quick reference* (figure 1-1). Retrieved November 12, 2006, from Cisco Web site:
<http://safari.ciscopress.com/158705227X/ch01>