Auto-Updating for Client-Server Software

Yili Zhang, Stephen Huntley, Urmita Banerjee, Dae-young Kim, Clif Flynt, and Gunes Koru
Health IT Lab, Department of Information Systems
University of Maryland, Baltimore County
Baltimore, MD
yili.zhang@umbc.edu

I. Introduction

A toolkit developed by Health IT Lab in Tcl/Tk [1]–[3] is a client-server software [4] for data quality [5] improvement for healthcare organizations. A global database is built on a server maintained by a system admin of the organization, considering data in this type of organizations is confidential and should be kept within the organization. Users will do create, read, update, and delete (CRUD) operations to the global database by remote communication package *::comm::comm* [6]. To reduce verbose process of updating a software on client-end and minimize the workload of system admin on server-end, the software requires an auto-updating mechanism to check and update the software automatically without effecting transport between client and server. However, the toolkit as a single Starkit [7] or Starpack [8] is not able to update itself and relaunch itself after updates. Instead, the running process will be intervened and the kit will be corrupted. In this case, a driver program was developed for auto-updating, which will launch main program as a child process, check update for it periodically, and update the if there is there is any updates detected. The updating function was achieved by leveraging update function in Starkit Developer eXtension (SDX) kit [9] and modifying details and fixing bugs to make it adapt to update features on both Linux and Windows system. This mechanism make the toolkit updates automatically fast and efficiently without affecting users' operation.

II. Methods

A. Update Function in SDX Kit

The sync feature in SDX kit is for starkit synchronization through channel, which is utilized in this study for starkit comparison and synchronization. Update command is called in our program to serve the purpose with "-n" option to check if there is a new version of kit on the target URL, or without "-n" option to sync the kit from the target URL. Listing 1 is the description of update function in help of SDX kit.

If update function is executed with "-n" option, it will return "Up to date" if there is no difference between two kits and "number of difference(s)" if there is any difference between two kits and indicates the number of difference. If update function is executed without "-n" option, it will replace the kit with the kit on target URL. Since the function only applies to Linux platform, it is modified to also be adapted to Windows platform. Some commands will be changed accordingly if the platform is detected as Windows, such as the "file mtime" command will cause error on Windows, they are modified to "file exists" instead if the platform is detected as Windows.

B. Driver Program

Then a driver program was developed including self-invoked functions for auto-updating and compiled as a Starpack. Main functionality program will be compiled as a Starkit, which will be upgraded and updated by driver program. The driver program will launch the Starkit as a child process, start to check for updates periodically, and update the kit if there is any new version detected.

Listing 1. Listing help of help function

```
update    Fetch or update a starkit from a Starsync server (via http)

          Usage: update ?-from url? ?-n? starkit

          -from url     Use specified Starsync server instead of default.
          -n            Show differences, but do not make any changes.

          Fetch changes so local starkit matches the one on the server.
          The Starsync mechanism only transfers files which have changed.

          Warning: this adds, modifies, *and* deletes files in the starkit.
```

There are six main functions in driver program:

- Server update: check and update in server mode
- Client update: check and update in client mode
- Update time configuration: read and write time for update period
- Process termination: terminate the main program before upgrading
- Progress bar: shows progress bar for when upgrading
- Server restart: restart server if server is down

1) Updates for server-end: The driver program for server will ping a URL, check if there is any difference of the kit on local and the kit on the target URL. If there is any difference detected, the driver program will wait till there is no transport between client and server, end server program, and update the toolkit on server-end. During updating toolkit on server, client will lose connection, operations of users on client-end will be terminated and they will enter a "server maintain mode". After the toolkit on server-end is synchronized, the driver program will re-launch the toolkit program in server mode. Then client can connect to server again and keep processing. The server program is running as a service, which will alleviate the workload for system admin. The updates for server will happen automatically and need no extra action, it will keep server running and updating on time unless the computer is shut down.

2) Updates for client-end: Since we do not want the client to hold an advanced version of software than server, the client need to check the version of kit with the kit on server. In this case, there is no need to check the kit on client-end with the URL, because if there is any updates detected, the driver program still need to check kit version with the kit on server before upgrade. Thus, the driver program for client only ping server periodically to check the version number of kit. Version information of the toolkit is stored in a Sqlite3 database [10], [11], wrapped with the kit. When server is launched, the kit will be mounted and the version number will be read into a variable on server. When client is launched, version number in the kit on client-end will also be read into a variable. Client program will read the version variable on server and compare it with the version variable on client-end. If version of client is outdated, user will be asked to update immediately, suspend running client program, and upgrade the kit with URL. This mechanism prevents client update earlier than server, which may cause inconsistency and corrupt the program and transport. In this case, user will be notified for updates right after server is updated if the time to check for updates is short.

3) Configuration of Update Time: Considering from security and load of CPU, pinging an URL too frequently can be a issue for some organizations such as Federal government and State agencies, the driver program enables user to configure the time period to check for updates. The configuration is only for server because the driver program for server will ping the target URL periodically while the driver program for client only ping it when there

is a outdated version detected. System admin who maintains server will modify the time in a text file, which will be read by driver program as the interval time between to checks of update. The default time will be set as one second and range for configuration is 0.1 second to 24 hours. In some organizations, pinging a remote server regularly will be detected as a violation operation. In this case, the interval time will be biased by twenty percent of the configured time randomly. That is, the real time between two checks will be a random number between 0.8 times to 1.2 times of configured time. This method evaded unnecessary restricts effectively.

4) Termination of Toolkit: The function to terminate the toolkit is calling Windows or Linux commands to kill a process. The function will be called after a need for updates detected and before the updates happen.

5) Progress Bar: When a upgrading is happening, a progress bar will show on the screen to notify user the progress of upgrading. The function will be called after the update stars, and the progress bar will move forward after one action is done.

6) Restart of Server: As server is running as a service, it will be hard to get noticed if server is down because of unexpected error on time. Thus, a function was developed as a self-invoked function to check if server is wake or not periodically. The function will restart server if server is detected as slept. By executing this function, the server can keep alive.

Figure 1 shows the process of how does driver facilitate the auto-updates.

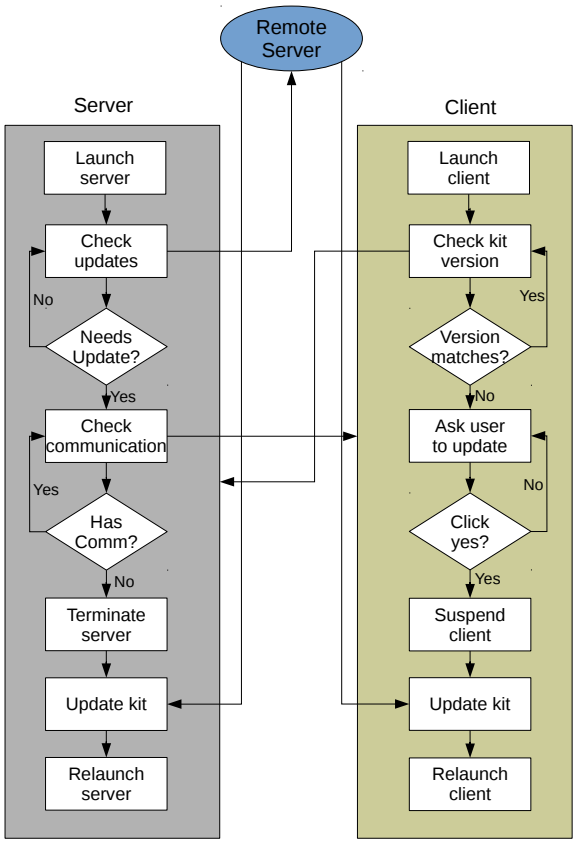

Fig. 1. Process of Auto-Updates for Client-Server Program

C. Termination of Driver Program

Although the toolkit is executed as a child process of driver program, the execution of driver program and toolkit should start and end at the same time. Therefore, ending driver program will end the toolkit, for both server mode and client mode. As a running service, the driver program of server will be terminated by ending the service manually, and server program will be ended at the same time. However, on client end, user only operates on the toolkit which is also the child process and may even not realize about the driver program. Thus, driver program will be terminated when user exit from the toolkit. The process ID of driver program will be passed as an argument to child process, exiting the child process will also kill the driver program by its process ID. In this way, user will exit from toolkit and end driver program at the same time.

D. Wrapping the Software

Same with the driver program – compiling the driver for client mode and server mode in one Starpack – toolkit for both client and server is compiled in one Starkit. A server option will be given by user when executing the Starpack to start update checking in server mode, and start to run the program Starkit as a server. Otherwise, if there is no option given, the driver program will check for updates in client mode and launch program kit as a client. In this case, driver program looks updates for the same Starkit for both client and server.

III. Optimization and Implementation

At the beginning of development, there is no time interval set for update checking, CPU will be overloaded because of driver is pinging ceaselessly, which slow down the performance of main program. Then we increased the default time interval for checking to every one second, which can both improve the performance of main program and update program in a timely manner.

Executing the server is simply by command line or starting the service. The command includes the target URL for update and "–server" option to indicate the software should run in server mode. If server is installed as a service, the option and target URL is already hard coded when software is installed. Executing client is by double clicking on the software icon with the default URL or command line with target URL.

IV. Discussion and Conclusion

The auto-update mechanism enabled user to check for updates automatically, especially on the server-end, no intervene from user is required to finish updating. What's more, this mechanism ensured client updates after server, which prevent the case that the version of client program is advanced than server program. At the end, the mechanism guaranteed both of performance and timely updates, it can also be applied to other client-server software other than the toolkit we developed.

References

[1] Flynt C. Tcl/Tk: A Developer's Guide. Elsevier; 2012.

[2] Welch BB, Jones K, Hobbs J. Practical programming in Tcl and Tk. vol. 1. Prentice Hall Professional; 2003.

[3] Ousterhout JK, Jones K. Tcl and the Tk toolkit. Pearson Education; 2009.

[4] Wikipedia. Client-Server Model Wikipedia;. (Accessed on 09/24/2018). "http://www.webcitation.org/72gG00LNf".

[5] Strong DM, Lee YW, Wang RY. Data quality in context. Communications of the ACM. 1997;40(5):103–110.

[6] Corporation A. Remote communication package;. (Accessed on 09/24/2018). http://www.webcitation.org/72gGJSvXM.

[7] SEH. Tcl Starkit;. (Accessed on 09/24/2018). http://www.webcitation.org/72gGSKD3c.

[8] Pooryorick. Tcl Starpack;. (Accessed on 09/24/2018). http://www.webcitation.org/72gGWmetg.

[9] Pooryorick. Tcl SDX kit;. (Accessed on 09/24/2018). https://wiki.tcl.tk/3411.

[10] Newman C. SQLite (Developer's Library). Sams; 2004.

[11] Owens M, Allen G. SQLite. Springer; 2010.