A method for building desktop software automated update systems

D. Sitnikov[1], A. Sitnikov[2] & O. Ryabov[3]
[1]*Kharkov State Academy of Culture, Ukraine*
[2]*Kharkov National University of Radio Electronics, Ukraine*
[3]*National Institute of Advanced Industrial Science and Technology, Japan*

Abstract

One of the problems in modern software development is the problem of introducing changes and fixes to the deployed software products in an automated fashion thus not requiring the end user to download the redundant product modules that had not been changed and performing manually the uninstall/install/check sequence. In this paper a method for building an automated update system has been suggested. The automated update system guarantees that the end user receives the latest changes and fixes to the product thus maintaining smooth experience from the software product. In this paper one of the methods for implementation of an automatic software update system for desktop applications is considered.

1 Introduction

In June 1998 Microsoft released a new operating system in which a new module responsible for notifying the user on the possibility of downloading some system add-ons such as new desktop themes, device driver updates, games and additional elements like NetMeeting was included. In the beginning Windows Update was accessible only from browsers and contained mainly operating system updates and new software in its repository. However crucial updates for software and the operating system kernel were later published through the same system. Microsoft connects the successful sales of Windows 98 in particular with inclusion of the module Windows Update [1].

WIT Transactions on Information and Communication Technologies, Vol 45, © 2013 WIT Press
www.witpress.com, ISSN 1743-3517 (on-line)
doi:10.2495/DATA130101

Really the process of updating modern software not only leads to improving user experience but also has unprecedented importance for safety and security of the system work as a whole. For instance, in case of appearing a new exploit for software the producer can quickly correct errors in the application and minimize the resulting damage.

Modern Agile methodologies for developing software such as eXtreme Programming, Lean, Scrum and others suggest an approach that differs from the traditional one. This approach requires the end user participation in the development process. In the projects using the Agile methodology user reaction is very important and, thus, the role of an automatic update system for such projects can hardly be overestimated.

Automated software updating plays an important part not only in desktop applications but also in Web applications.

2 Description of existing systems

As examples of applications using automated update systems one can consider the browsers Mozilla Firefox, Google Chrome, Internet Explorer, mail client Mozilla Thunderbird, Microsoft Office package, the developer environment Java Development Kit, Java Runtime Environment and many others. Thus at present using AUS (Automatic Update Systems) can be considered as a standard de facto.

In the applications market at present there is a trend towards the consolidation of sources for software distribution in specialized “markets”. As examples one can consider Apple iTunes that can work on the mobile platform iOS and on the desktop platform MacOS as well. Also Google Android and Windows Phone have a similar system. In Microsoft Windows 8 the application Microsoft Market is built in by default. All these systems imply by default the possibility of updating software being distributed.

3 Requirements to Software AUS

We think that a modern software AUS should meet the following requirements:

- Safety
- Sustainability if case of unexpected errors
- Scalability
- Modularity
- Extendibility
- Integration into development process

4 The development process and AUS

In a simple way the process of developing a modern software application includes the following stages:

1. Application requirements analysis.
2. Creating specifications for modules being developed/changed.
3. Writing a program code.
4. Generating a test version of an application.
5. Testing assembly from step 4.
6. Correcting errors and if necessary returning to step 3.
7. Generating a final assembly.
8. Distribution of the assembly from step 7 to the end user environment.
9. Gathering and consolidation of information on discovered errors and user feedback; returning to step 1 or step 3.

In this paper we consider only steps 7 and 8.

As a whole AUS consists of two main parts: client module that directly updates an application and server application that stores packages with updates and the current application version.

5 A description of our approach

In our opinion the development process should include "seamless" integration of software AUS. In other words update packages creation should be performed without direct involvement of the user.

Below consider the main stages for process of creating an update package for an application.

1. A responsible person launches the process of generating the final application assembly on the CI server.
2. Obtaining information on the last successful application assembly.
3. Determining the necessity of assembling a client automated update system module.
4. Downloading the last application revision from the version control system
5. Determining application modules that have been changed.
6. Calculating the assembly version number.
7. Composing a list of changes based on information from the version control system.
8. Modification of auxiliary files storing version information for modules discovered in step 5.
9. Assembling and signing all application modules with a company certificate private key.
10. Creating and signing an update package with company certificate private key.
11. Creating a file with the help of which the client module determines the necessity of updating the application (further it is called "indication file").
12. Creating an installation package.
13. Loading the resulting files to the update server.

Let us consider some elements of this process in more detail.

6 Launching the process of generating a final application

We think that for errorless assembly of a final application the company should use the methodology of so called continuous integration (CI). Thus in the process of generating the resulting application the human influence can be minimized. It can be seen especially well in the process of developing software consisting of a set of heterogeneous modules. Normally the application assembly process is described in the form of so called assembly plan that has some coded name for facilitating its identification on the stage of log analysis.

At present there exist a variety of commercial and free solutions. The following products can be considered as special examples: Atlassian Bamboo and JetBrains TeamCity that allows configuring and controlling the process of assembly in a special visual editor. In spite of the fact that when mentioning CI a particular software product is meant, in principle the company can use this methodology entirely with the help of standard tools provided along with the development environment and operating system.

The company's responsible person making decisions on release of a new application version uses the CI server interface for launching the application assembly plan and supervises the process by using a log viewing system. In case of assembly errors a message with a detailed description of the error and link to the full log of assembly process is sent to the e-mail addresses of team persons responsible for application development. In case of a successfully completed process the responsible person synchronizes artifacts of the assembly (installation packages, update packages, indication file) with the production server from which applications can be automatically updated.

7 AUS server module

The general structure of the server module looks as follows:

1. Catalogue latest. This file contains all application modules in the original form.
2. Files *.zdef. Such files contain scenarios for updating each preceding version. For example file 2115-2111.zdef stores a scenario of application updates from version 2.1.1.1 to versions 2.1.1.5. Files are packed in the Zip format.
3. relnotes.txt. This file contains messages sent when files are committed in the version control system.
4. update.info. This is an indication file for the client module AUS. It looks as follows:

```
<UpdateInfo>
  <Self Hash="SHA256 HASH"/>
  <ProductVersion="2.1.1.10" MainExecutable="App.exe"/>
</UpdateInfo>
```

The above element Self contains SHA256 hash from a client module file AUS. This hash allows the client module of AUS to determine if there is a need in updating itself. The element Product stores information on the actual version of the application and its main executable file.

5. Updater.exe. The last actual version of the client module.

8 Update scenario format

In the process of its work the CI server generates separate update scenarios for each preceding application version. For example, if the first application version was 2.1.1.1 then in case of generating applications with version 2.1.1.5 the following scenarios will be created:

2.1.1.1 -> 2.1.1.5
2.1.1.2 -> 2.1.1.5
2.1.1.3 -> 2.1.1.5
2.1.1.4 -> 2.1.1.5

Thus the minimization of company server outgoing traffic is performed since during the update process not all application modules are downloaded but only changed parts of some modules. A typical scenario file looks for example as follows:

```
<?xml version="1.0"?>
<UpdateDefinition>
  <Scenario>
    <Pre />
    <Post />
    <Actions>
   <UpdateFile HashBefore="SHA256 Hash" HashAfter="SHA256 Hash" Path="path to file">VCDIFF in Base64</UpdateFile>
    ...
    ...
    </Actions>
  </Scenario>
  <Signature ...
</UpdateDefinition>
```

The changes to the particular software module is introduced with the help of format of delta encoding described in RFC 3284 called VCDIFF [2]. The binary delta file is then encoded using BASE64 scheme that allows to use it directly in the XML file.

Various actions built in the client module can be indicated as Action, such as registry operations, launching programs etc.

The Signature element is a scenario digital signature and added after all other elements at a corresponding stage.

9 AUS Client module

This module is designated for tracking new application version appearance and direct changing application modules according to a corresponding update package scenario.

A client module is launched in separate thread at application start-up and passes the following steps in the process of its working:

1. Self-Maintenance. At this stage the actuality of the client module itself is checked. Thus self-updating mechanism is implemented. In case a new version is accessible it is loaded and the corresponding file is replaced. The updated version fulfills all further operations.
2. Check Updater Hash. This step allows determining whether or not the last version of the client module AUS has been damaged during the download process. If the hashes are not equal it is not possible to update the application.
3. Download Update Definition. During this step the corresponding update scenario is downloaded.
4. Check Definition Signature. The client module is to confirm the validity of the scenario signature with the public company key which is distributed along with it. In the case of check failure it is not possible to continue the process of updating.
5. Check File Hashes. At this stage the unpacking of diff-files from the update scenario and checking their hashes are fulfilled. In case of check failure the update process should be stopped.
6. Perform Pre Actions. If the scenario contains any preliminary actions that should be performed before updating application modules such actions are fulfilled at this stage.
7. Update Core Libraries. Here direct updating application files with the help of scenario diff-files is performed.
8. Perform Post Actions. In case the scenario contains actions that should be performed after updating application modules such actions are fulfilled at this stage.
9. Clean-Up. Temporary files are deleted.
10. Start The Program Up. The main application file defined in the indication file is started.

10 Processing update errors

The most common error types are as follows:

1. Security errors.
2. Network connection errors.
3. Data integrity errors.

The first two types do not suppose any special processing except for indicating errors to the user and asking the user to try launching the update later

or call support. In the case the file hash does not equal the expected one the client module requests the latest version of the corresponding file on the server (catalogue “latest”) and replaces the original file completely (not only the changed part).

11 Conclusions and future work

The suggested method for developing an automated software update system for desktop applications allows the business to concentrate its efforts on continually improving the software quality and provide smooth and error-prone user experience to the end users.

One of the major improvements for the suggested method would be the inclusion of the mechanism for dealing with security certificates expiration. Additionally, there is some room for improvement within the update format that can be replaced with pure binary one to reduce packet size.

The system based on the described method is currently being used in one of the leading driver updates software products Xionix DriverHound [3].

References

[1] Strong Holiday Sales Make Windows 98 Best-Selling Software of 1998 http://www.microsoft.com/en-us/news/press/1999/feb99/holislpr.aspx
[2] The VCDIFF Generic Differencing and Compression Data Format (http://tools.ietf.org/html/rfc3284.html)
[3] Xionix DriverHound http://xionix.com/products/driverhound/