A Remote System Update Mechanization (RSUM)

A Thesis Presented to

The Faculty of the Computer Science Program

California State University Channel Islands

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science in Computer Science

by

Evan Glick

November 2010

COMPUTER SCIENCE PROGRAM APPROVALS

Approved: _____ Date: _____
Professor William Wolfe, PhD., Computer Science Program Chair
Thesis Advisor and Evaluation Committee Lead

Approved: _____ Date: _____
Professor Andrzej Bieszczad, PhD., Master's Degree Program Director
Evaluation Committee Member

Approved: _____ Date: _____
Professor Peter Smith, PhD., Undergraduate Advisor
Evaluation Committee Member

UNIVERSITY APPROVAL

Approved: _____ Date: _____
Professor Gary A. Berg, PhD., Interim Associate Provost

Remote System Update Mechanization

by
Evan Glick

Computer Science Program
California State University Channel Islands

Abstract

Distribution of product updates is an integral part of system maintenance and enhancement that is routinely provided by equipment developers as part of their extended product support to customers. Such refinements may affect hardware, software or both, and, though some processes are currently being used to provide similar capability, their functionality is limited in situations where destination systems are isolated or otherwise secured. In general, these utilities are not capable of concealing the actions performed for security-constrained updates.

This paper addresses the development of a remote system update mechanization capable of efficiently performing, with minimal user involvement, not only software updates, but also firmware and reprogrammable electronic device settings modifications for a specific proprietary signal analysis system. By leveraging existing resident functionality, these features are implemented to achieve the desired distributed updates in an easy to use but obfuscated manner.

Acknowledgements

I would like to express appreciation to my employer (SystemWare, Incorporated) for their support and encouragement in carrying out this development, and, in particular, to my colleagues Mike Levesque and Dale Schroeder for their equipment-based critical comments and suggestions. Additionally, I am grateful to Professor William Wolfe for his insights and patient guidance in helping me shape this report and its technical content, and to Professors Andrzej Bieszczad and Peter Smith for their participation in its evaluation.

TABLE OF CONTENTS

1 Introduction

1.1 Purpose of the project

This project focuses on the implementation of enhancements to a proprietary signal analysis suite, enabling the distribution of product updates between three different machine types: Unit As (UA - highest level in hierarchy), Unit Bs (UB - middle tier), and Sensors (lowest level). A typical system configuration consists of a single UA, and approximately one UB for every ten Sensors installed. One example system configuration could consist of one UA and two UBs each with ten Sensors attached for a total of twenty Sensors, all managed by the user at the UA. Within this context, only the UA and some of the UBs will be directly accessible for user interaction. Since the majority of the machines in a complete system will be Sensors, the goal is to achieve the capability to perform updates on potentially hundreds of physically inaccessible machines simultaneously.

Due to the nature of how the machines interact, it is crucial that corresponding revisions of software and firmware are always utilized [23]. One of the essential features of this development is to provide users with the flexibility to either apply officially released updates, or to create their own update packages to apply to all or selected machines (UBs or Sensors) within their distributed system. User-created packages may address software updates, firmware updates [13], general updates, and/or any electrically erasable programmable read-only memory (EEPROM) [12] settings that need to be applied.

Since most systems will reside in inaccessible locations, the update process will by necessity be configured to execute all actions automatically upon user initiation, and then return to a standby state at completion. In many cases, the only other option available for system setup would be by means of Microsoft's Remote Desktop™ application, and would result in a tedious update procedure for configurations with numerous distributed machines. Thus, creation of this product enhancement makes the update process significantly more manageable throughout the equipment life-cycle.

Initial considerations regarding alternative strategies for achieving the required update functionality included research into the availability of methods already available as products or described in the open literature. This study resulted in the finding that, while similar functionality is available across different platforms (System Center Configuration Manager™, Apple Remote Manager™, etc.), [1-9] none of these options were appropriate for this application since the majority of these systems will reside on dedicated and/or isolated networks. The requirement to conceal the update methods for system configuration changes was also a deciding factor behind the development of this mechanism.

Project design was influenced specifically by the methodologies and frameworks discussed in the open literature to minimize node downtime in the application of system updates [18,19,22-23]. Consideration for system and communication security [10], as well as the actual upgrade process led to the development of the RSUM in its current form.

1.2 Intelligent Agents

The design and eventual development of the RSUM was motivated by the desire to enhance the user experience when performing system maintenance on Systemware proprietary systems. Implementation of an Intelligent Agent [14,27-28] to assist the user by acting on their behalf in performing the specified system updates effectively minimizes the complexity of the actions required to perform this task. This potentially allows for non-expert users to perform the update procedures, as the desires of the user are defined in the creation of update packages. When deployed, these packages spawn the Agent to perform the operations.

1.3 Other Considerations

To ensure proper functionality, all system elements are required to be connected to the same local area network (LAN), and their corresponding internet protocol address (IP / PORT) information known to the UA / UB user intending to utilize this capability. Since this information is already requisite for normal system operation, it should be made available during installation.

2 Overview

Distribution of product updates poses a variety of challenges to system developers, especially in situations where the destination equipment is isolated from external intervention, and may be hosted with any one of several evolved configurations/versions. The challenges faced when updating distributed systems have been emphasized in the open literature, especially when attempting to minimize system down-time [18,19,22-23]. To achieve the overall objective of creating such an update mechanism, several essential elements must be implemented. Each of these specific features is outlined in the steps below, and will be described in greater detail in the design portion of the document.

2.1 Package Creation / Editing

Whenever updates are to be applied to one or more remote machines, each instance requires that appropriate adjustments are made to achieve destination-specific "packages" for distribution within the protected environment. These packages must be properly assembled and specifically tailored before they can be sent and/or applied to any remote machine. In order to facilitate creation of appropriate packages, the interface common to both the site's system manager and manufacturer's representatives will be created in a way that facilitates proper selection of the various available update options. Additional features, such as the ability to modify existing packages, or to view package contents before their application to remote systems, will also be implemented to maximize system functionality.

2.2 Remote Target Selection & Package Transfer

Once a package has been created, the site's system administrator will to be required to specify which machines (specific or all) upon which the update package is to be applied. Depending on the specific system configuration, there may be instances where only specific machines require an update. Since this capability will be implemented within an application, existing resident functionality specific to transferring files will be leveraged to ensure that package delivery to the specified machines succeeded without corruption or modification. Failure to securely deliver update packages could undermine system integrity, as well as purpose [10].

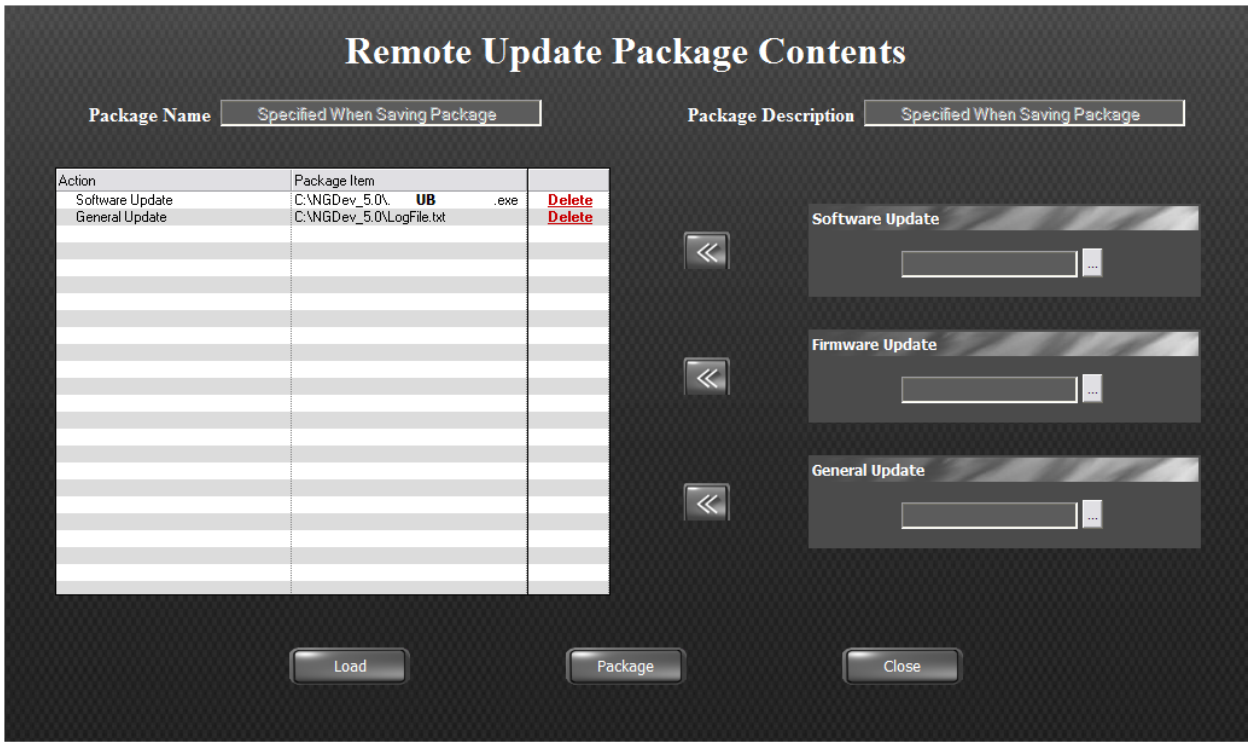
2.3 Agent Creation & Executing Instruction Set Contents

Upon successful update package transfer to a remote machine (i.e., UB or Sensor), a unique message confirming its arrival and containing its name will be issued. Processing this message will result in decompression of the associated package and extraction of the Update Agent. This Agent will then be initiated to execute the instructions as outlined in the update package while maintaining communications with the higher-level machine (i.e., UB or UA) as appropriate. Subsequently, the Agent will configure the system back into an idle, standby state ready for regular use upon completion.

3 Design Details

3.1 Product Update Create/Load GUI - Customer Implementation

The elements that the Customer user wishes to update on remote machines must first be selected before their eventual deployment targets can be specified and the contents of the Product Update Package prepared. A display similar to the one shown below will be implemented in HTML/ Javascript allowing a user to select Software, Firmware, and General selections to package. Located within each portion, the ability for a user to browse for the corresponding file type will be present. The ability to Load in previously created packages and view the contents, package name, and package description will also be implemented. The interfaces demonstrating how these attributes are set will be described below.

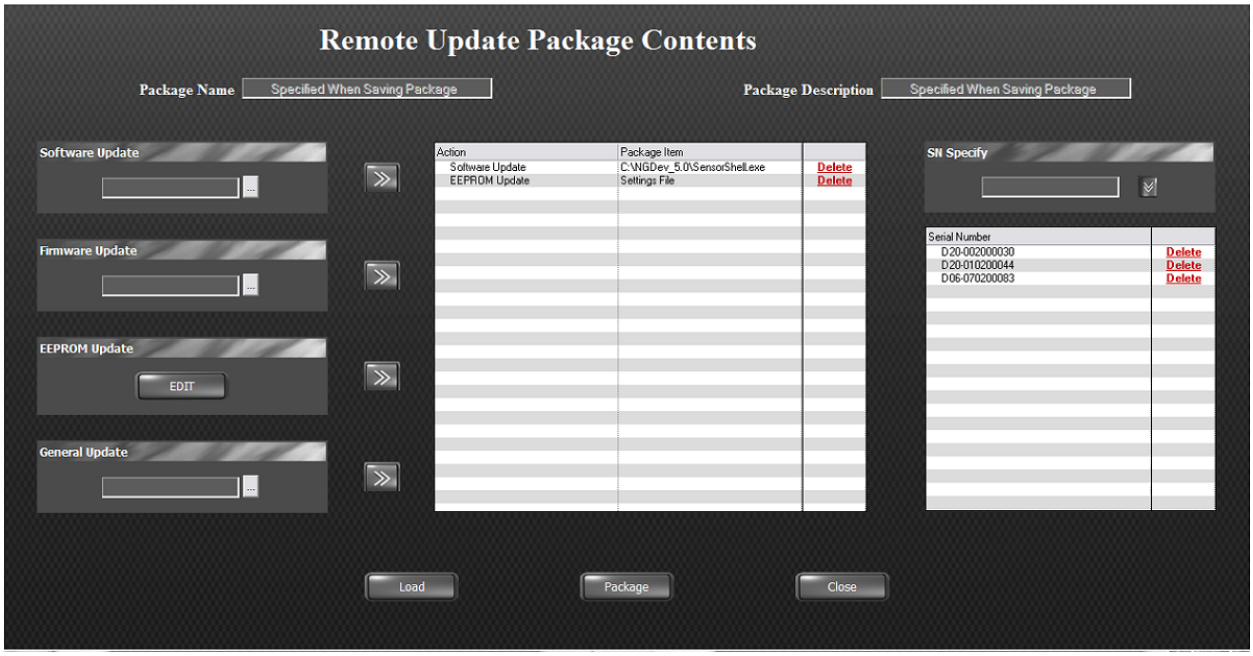


The order in which each element is selected and added to the Remote Update Package Contents will dictate the sequence that the update is performed on the remote system. As future elements are added to the General Update portion of this feature, this may allow for operations currently not anticipated.

3.2 Product Update Create/Load GUI - Representative Implementation

The elements that the Representative user wishes to update on remote machines must first be selected before their eventual deployment targets can be specified and the contents of the Product Update Package prepared. A display similar to the one shown below will be implemented in HTML/Javascript allowing a representative to select Software, Firmware, EEPROM, and General selections to package, as well as determine if the Package may only be applied to specific units based upon their Serial Number. Located within the interface is the ability for a representative to browse for each appropriate file type corresponding with its update action. The ability to Load in previously created packages and view the contents, package name, and package description will also be implemented. The interfaces demonstrating how these attributes are set will be described below.

This particular interface will only be accessible to a User when the HW Key Dongle is detected on the system.



The order in which each element is selected and added to the Remote Update Package Contents will dictate the sequence that the update is performed on the remote system. As future elements are added to the General Update portion of this feature, this may allow for operations currently not anticipated.

3.3 Product Update GUI Attributes

The processes associated with each of the elements contained within the GUIs shown in **Step 3.1** and **Step 3.2** may be categorized as either an update option, package option, or directly related to the serial number of the unit being updated. While the controls are shown, the actions observed when activated have not been defined. Outlined below are each of the attributes and functions to be implemented in order to achieve the described functionality.

Update Options

Software Update - Allow user to browse for *.exe files to specify as new software update.

Firmware Update - Allow user to browse for *.bit files to specify as new firmware update. *(Sensor ONLY)*

EEPROM Update - Present dialog for the user to input the specified settings to be established on the remote system *(Sensor ONLY - Will Require HW KEY Dongle - Not Visible Without).*

General Update - Allow user to browse for *.* files to specify to send in update package.

Package Options

Close - Close the Remote Update Package Dialog / do NOT save selections

Load - Load a previously created package *(Populates Tree Grid - Specified Settings)*

Package - Create directory/folder/instruction-set & zip

Package Desc. - Note indicating purpose / contents of package

Package Name - Allow the user to specify the name the package (.zip)

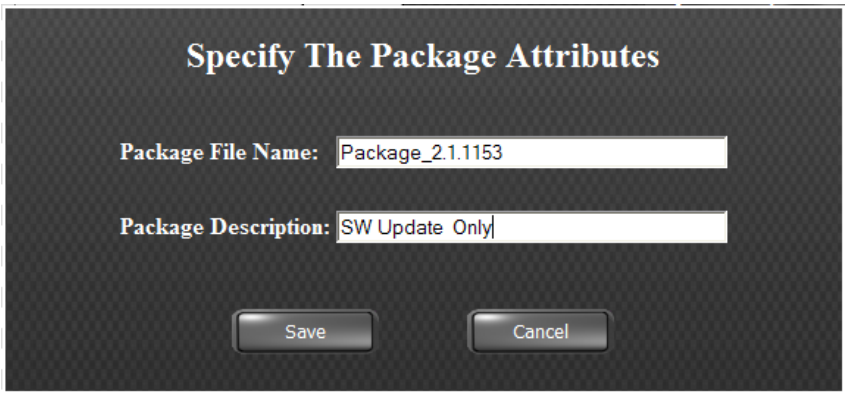
SN Specify

Serial Number - Values input by the Representative user will dictate which machines the resulting package may be applied to. *(Rep. ONLY - Will Require HW KEY Dongle - Not Visible Without).*

Once the contents of the update package have been selected by the user, the files to be transferred to the remote system(s) will need to be prepared along with the instructions of how to perform the update operation. This will be discussed in the following steps.

3.4 Product Update Package GUI

Once elements have been selected and added to the package interface, the user will be able to select the "Package" button on the Remote Update Package Contents dialog. Performing this action will finalize the properties and content of the package, as well as allow the user to specify overall package attributes. The entries specified by the user will ideally make the package content clear, as this is the information that will be readily available when selecting a package to load or deploy. Selecting this button will result in a display similar to the following.

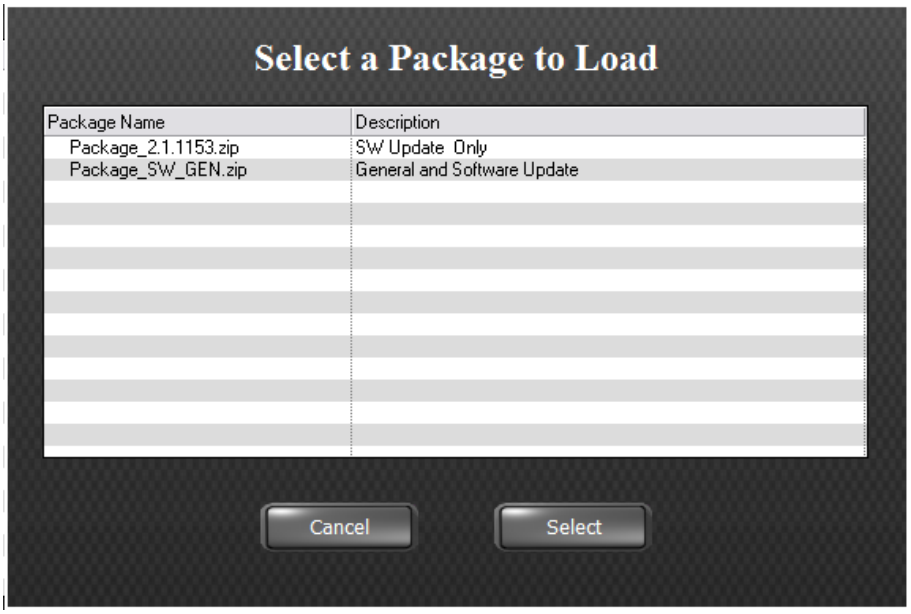


The entries specified by the user within this dialog will set the name of the ZIP package, as well as the package description which will be set as an attribute of the ZIP folder. Setting the package description as an attribute of the ZIP folder will allow for package descriptions to be read without unzipping the package and parsing the XML Instruction-Set.

Entering values into the fields and selecting the "Save" button, will return the user to the original Remote Update Package Contents dialog with all package information updated for review. At this point the package will be created; however, it may be loaded and edited at a later date as described in the following sections.

3.5 Product Update Load Package GUI

There may be instances where a user wishes to load in a previously created package to review its contents, or to make modifications for re-use. Packages available for selection may have been created by the user, or may have been provided as an official system upgrade. In order to provide the desired user experience, a dialog will be created that displays the attributes specified in the Product Update Package GUI described in **Step 3.4**. Selecting the "Load" button from the Remote Update Package Contents dialog will result in a display similar to the following.

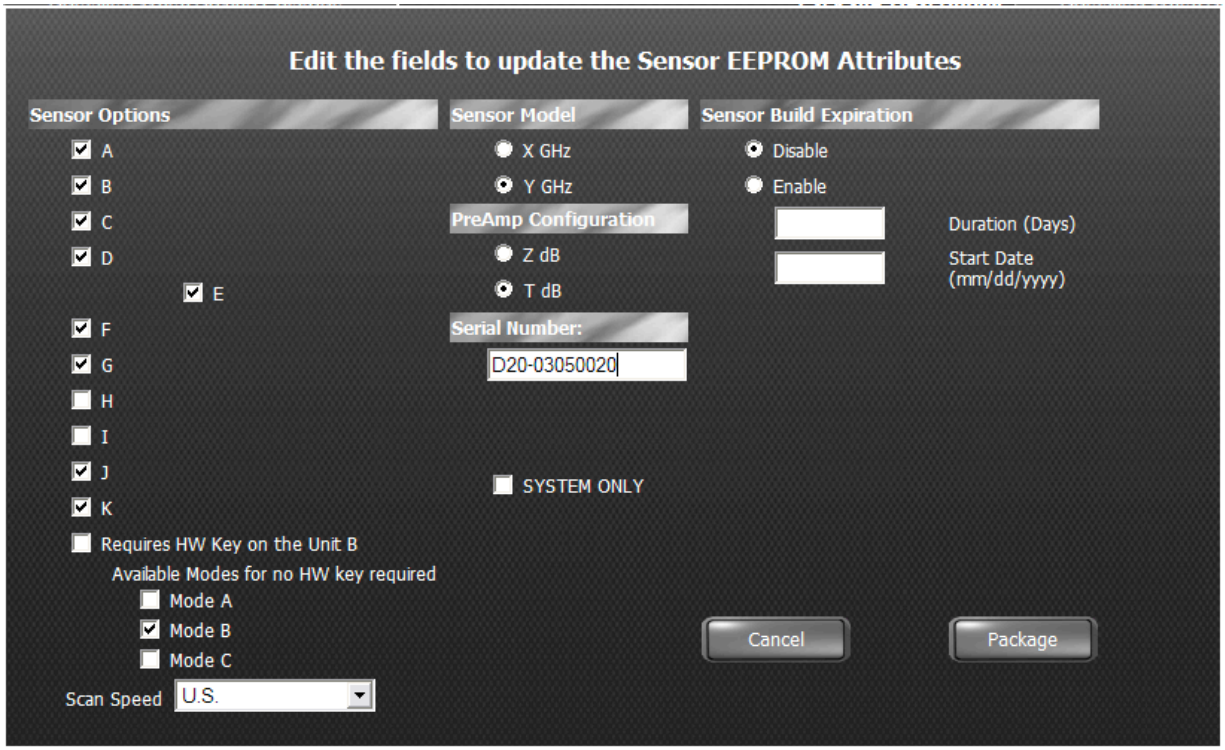


From within this dialog a user will be able to select any previously saved package (located within the default C:\Sensor\UserSetupData\Local\Package directory). When a package is selected, the Remote Update Package Contents dialog will populate with the appropriate entries corresponding to the contents of the XML Instruction Set of the loaded package. If the package is modified after loading, the "Package" button will change to "Re-Package" and all attributes will repopulate corresponding to their current values.

In the case where a package is created containing either EEPROM settings, or specified S/Ns of systems which the package only applies, the package will only successfully load if the HW Key is detected on the system. If no HW Key is present and the package is selected, a message similar to the following will be displayed: "This package is not intended to have its contents altered."

3.6 Product Update EEPROM GUI (REP-ONLY)

The ability to specify EEPROM settings to be applied via an update package will only be made available to a Representative user who possesses the required HW Key dongle. In order to provide the desired capability, an interface will need to be created which allows for the attributes to be specified. This interface will be activated by selecting the "EDIT" button contained within the EEPROM Update portion of the Remote Update Package Contents dialog, and will appear similar to the following.



Selecting "Package" will create an entry within the Remote Update Package Contents dialog. Clicking the "EEPROM" button subsequent times will allow the User to review and or modify the settings specified.

The settings specified from within this dialog will be directly written to the XML Instruction-Set outlined in **Step 3.7** below. These values are stored as member variables of the class until the Package is finalized (Zipped & Password protected).

3.7 Package XML Instruction Set

Once the contents of the update package to be sent to the remote system(s) has been specified by the user in the previous step, the contents of the package may be prepared along with an Instruction-Set indicating how the update is to be performed (sequence defined by order specified in **Step 3.1 / 3.2**). The instruction set will be constructed via existing XML software classes present within the application to create an XML file. The tags and attributes established within this XML file will indicate which update operations are to be performed and which files are to be utilized for each purpose as necessary. Portions of the file format are specified below.

XML Instruction Set Format:

```
<PackageContents>
        <NumActions>4</NumActions>
        <PackageSource>UB-123</PackageSource>
        <Description>Description</Description>
        <Action0>
                <ItemName>Software Update</ ItemName >
                <ItemFile>install.exe</ItemFile>
        </Action0>
        <Action1>
                <ItemName>Firmware Update</ ItemName >
                <ItemFile>Firmware.bit</ItemFile>
        </Action1>
        <Action2>
                <ItemName>EEPROM Update</ ItemName >
                <ItemFile>Settings File</ItemFile>
        </Action2>
        <Action3>
                <ItemName>General Update</ ItemName >
                <ItemFile>File.*</ItemFile>
        </Action3>
</PackageContents>
```

3.8 File Manipulation & Package Creation

The XML instruction set file will be zipped along with any additional update files specified in **Step 3.1 / 3.2** to create the Update Package to be sent to the remote machines(s). If the folder does not already exist, a "Package" Directory will be created under **C:\Sensor\UserSetupData \Local\Package** within which the specified files will be copied and zipped packages will be created. Each zipped package will also be password protected during the zipping process in order to prevent any potential interference. Available secure zip and unzip classes will be utilized to perform this operation.

When packages are created, all specified files will be copied to a zip directory where the result is password protected. When packages are unzipped for loading or to execute their contents, the files are unzipped to the local package directory with the exception of the XML instruction set which is unzipped directly into memory in order to prevent interception/manipulation.

Along with the update files and XML instruction file, an updated version of the SensorAgent will also be packaged into the update to be sent. Sending the SensorAgent down as part of the package helps minimize the possibility of a user manipulating the update operations being performed on their system, and also ensures that any modifications made to the update mechanism are performed on the remote system [23]. The details of the SensorAgent will be specified in **Step 3.11** in this document.

Step 3.9 will outline the interface to select the target systems being updated.

3.9 Remote Target Selection

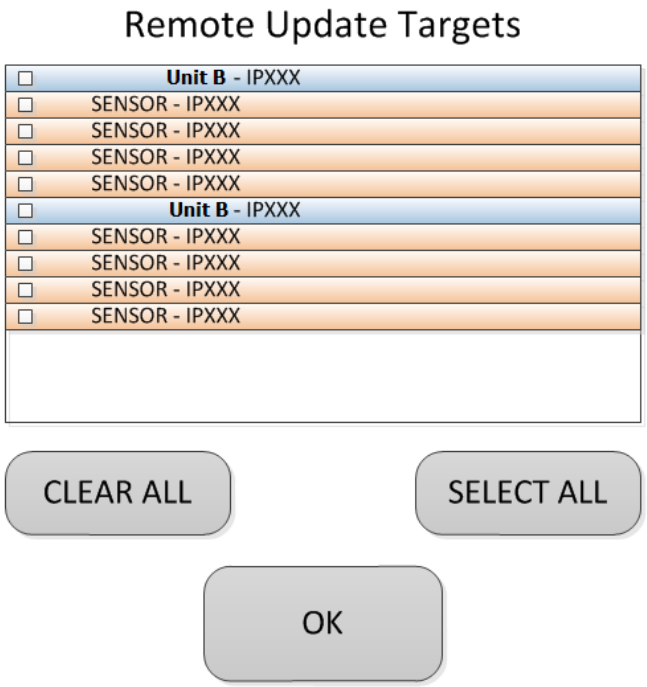
With the update selections made and the update package created, the systems which are to be updated must be selected by the user. Since there are two different methods that may be utilized to perform this update, the details of each will be described below.

Update Performed via Unit B Configuration Settings

A dialog will be presented containing a list of all connected systems (UBs & Sensors) within a grid. In each row the option to select each system will be available in the form of a checkbox. Within this dialog there will also be "Select All" and "Clear" options for the cases where a User wants to perform a universal update or clear their selections and start over.

Update Performed via Sensor Shell View Window (Utilities)

Since the intended update target is known in this scenario, no remote target selection is necessary. The update will continue to **Step 3.10** described below.



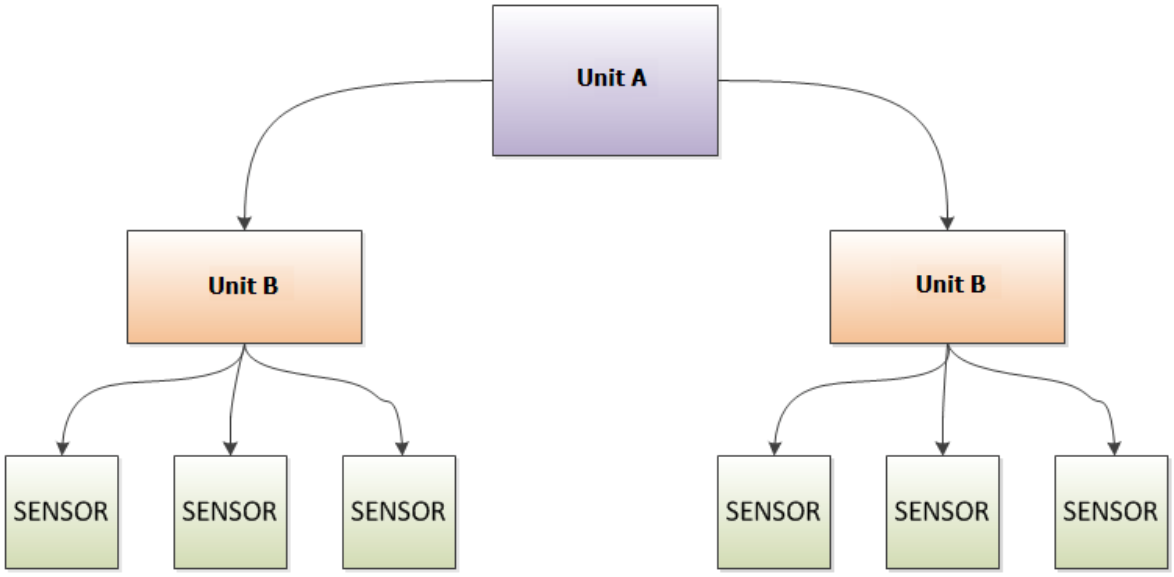
The entire package will be sent to all systems specified via this Remote Target Selection dialog (only one package). During the execution of the instructions the machine type will be determined and unrelated updates will not be performed.

ONLY currently connected Sensors / UBs will be displayed within the Remote Update Targets Dialog in order to prevent collision issues with packages failing to deploy.

3.10 Transfer The Update Package

At this point the Update Package has been created, and its intended destination system(s) have been identified. In order to transfer the packaged .zip file to the remote systems, existing file transfer functionality will be utilized.

In order to minimize the bandwidth being consumed in the transfer process, the Update Package will be sent initially to each top node selected in the Remote Target Selection (UB). From each of these nodes, the package will then be transferred to the selected lower level nodes (Sensors) [18-19,22,25-27]. This is demonstrated in the following image.



Status messages will be sent back to the UA / UB indicating when each Update Package has been successfully transferred to its intended destination. The status of each system will also be updated in the UA / UB node view to indicate that an update is being applied to the system in the form of an animated icon.

Once the Update Package has arrived on the remote system, and the UA / UB has received confirmation, a message will be sent to the remote system which will initiate the SensorUpdate Agent. The creation and capabilities of this agent will be covered in **Step 3.11** below.

3.11 Create SensorAgent Agent

A remote system (UB / Sensor) will receive a specific message once an Update Package has been successfully transmitted and is available for unpacking and processing. This message will contain the command to both extract/unzip the contents of the Update Package, as well as execute an SensorUpdate Agent which was passed down as part of the Update Package [14,27-28]. This agent will be created as a separate project specifically to perform the update operations contained within the XML instruction set described in **Step 3.7**.

The contents of the XML Instruction Set will dictate the actions then performed by the SensorUpdate Agent. Each of the potential update instructions that may be performed by the SensorUpdate Agent will be outlined in the following steps.

3.12 SensorAgent: Performing Software Update

In the case where a Software Update was selected by the user in **Step 3.1/3.2**, an updated version of the Systemware software has been transferred to the remote system as part of the Update Package. In order to successfully perform the software update, the following steps will be taken. (This update may be performed on both an UB or Sensor node type.)

- Terminate the UB/Sensor Software (Gracefully)
- Uninstall the existing UB/Sensor Software (Silent - No Prompts)
- Install the updated UB/Sensor Software (Silent - No Prompts)
- Place a copy of SensorUpdate Agent in the System Startup (Folder or Registry)
- Add an entry in the Log File (Software Update Performed)
- Reboot the System

Once the system restarts, the SensorAgent will continue with the update process where it left off.

3.13 SensorAgent: Performing Firmware Update

In the case where a Firmware Update was selected by the user in **Step 3.1/3.2**, an updated version of the Systemware firmware [13] has been transferred to the remote machine as part of the Update Package. In order to successfully perform the firmware update, the following steps will be taken. (This update may only be performed on the Sensor node type.)

- Verify the node type is that of a Sensor. If it is an UB node type, do not perform the Firmware Update
- Terminate the Sensor Software (Gracefully)
- Load/Parse the Bit Image
    - Perform the Firmware Burn
- Close the connection to the FPGA
- Send a Message to the UA/UB indicating that a Firmware Burn has occurred and that a Wake on LAN packet will need to be sent to this system in 2 minutes.
- Place a copy of SensorAgent in the System Startup (Folder or Registry)
- Add an entry in the Log File (Firmware Update Performed)
- Shut Down the System

Once the system receives the Wake on LAN Packet and boots up, the SensorAgent will continue with the update process where it left off.

3.14 SensorAgent: Performing EEPROM Update

In the case where an EEPROM Update was selected by the Representative user in **Step 3.2**, the specified EEPROM [12] settings were transferred to the remote system as part of the contents of the XML Instruction File contained within the Update Package. In order to successfully perform the EEPROM settings update, the following steps will be taken. *(Sensor Node Type Only)*

- Verify the node type is that of a Sensor. If it is an UB node type, do NOT perform the EEPROM Update.
- Terminate the Sensor Software (Gracefully)
- Read the desired settings from the XML File
- Verify if a S/N Restriction is placed on this EEPROM Update, that the S/N of this Sensor matches those allowed for this Update Package
    - If the S/N is allowable, **APPLY** the EEPROM Settings (Within SensorAgent)
    - If the S/N is NOT allowable, **CANCEL** the EEPROM Update
- Add an entry in the Log File (EEPROM Settings Updated)

3.15 SensorAgent: Performing General Update

In the case where a General Update was selected by the user in **Step 3.1/3.2**, an additional file or specific command was passed down to the remote machine via the update package / XML instruction set. Since an updated copy of the SensorAgent will have been transferred to the remote system as part of the Update Package as well, the intended logic required to perform the update should be processed by the SensorAgent without incident.

In the future it is possible that additional features may be added to this category. By allowing the user to specify the sequence of events in the XML instruction set, and by passing down the SensorAgent on every update, this mechanism should be flexible enough to handle any future updates desired [23].

3.16 SensorAgent: System Configuration to Idle

Once the SensorAgent resident on the remote system has completed applying all updates specified by the user in **Step 3.1/3.2**, all necessary clean-up will be performed. The following steps will be taken to ensure the system is placed back into an idle state where it may be commanded by an UA/UB machine and resume normal operation.

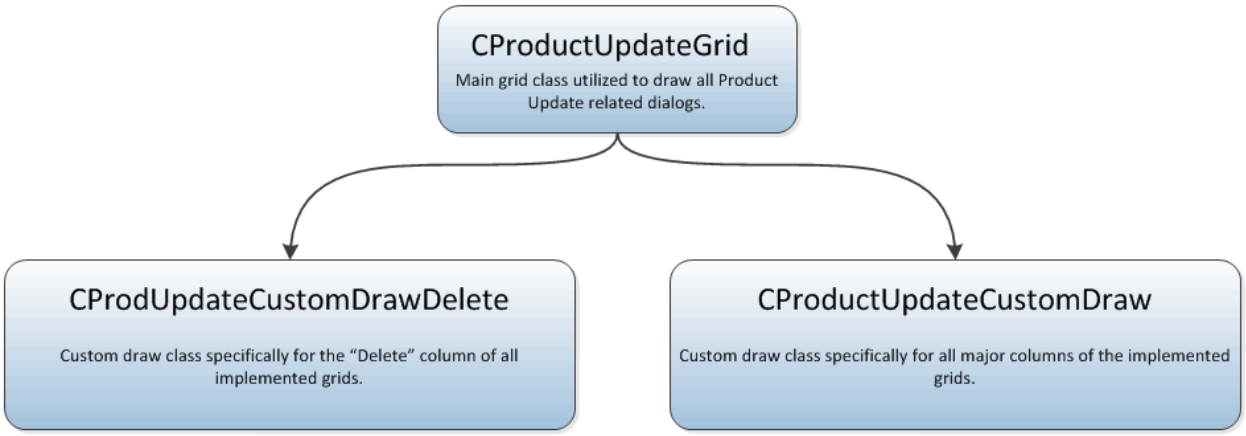
- The Update Package will be deleted
- Send a message to the UA/UB machine that originally sent the Update Package
    - Indicate the Update was Successful / Failed
- All changes made regarding SensorAgent in the System Startup Folder / Registry will be undone
- Add an entry in the Log File (Update Completed)
    - Transfer the Log File to the MRM/RM which initiated the Update
- Send a message to the UA/UB machine to initiate a connection to the remote system in 2 minutes.
- Terminate the SensorAgent
- Reboot the System

When the system reboots, the UB/Sensor software will be automatically launched. The UA/UB machine will attempt and succeed to connect to the remote system. At this point all updates should have been performed and the system should be available to resume normal operation.

4 Implementation Details

4.1 Product Update Create/Load GUI - Customer Implementation

In order to create the designed Customer Implementation of the Product Update Create/Load GUI, a CDHtml Dialog class **ProductUpdateDlg** was created. Along with the necessary HTML controls and their associated member variables and event functions, a third party grid control interface (Dapfor) was also added to the dialog. The implementation of the grid control required the creation of a **ProductUpdateGrid** CDataObject class as well as the creation of two additional CCustomDraw classes in order to manipulate the contents of the grid. These two classes are **ProductUpdateCustomDraw** and **ProductUpdateCustomDrawDelete**. This is depicted in the following diagram.



In an effort to minimize the amount of duplicated code between the Customer and Representative versions of the Product Update Create/Load GUI, a **ProductUpdateObj** object was also created which contains the majority of the logic to perform the actions associated with creating a Remote System Update Package. Details describing the **ProductUpdateObj** object will be provided in **Section 4.3** below.

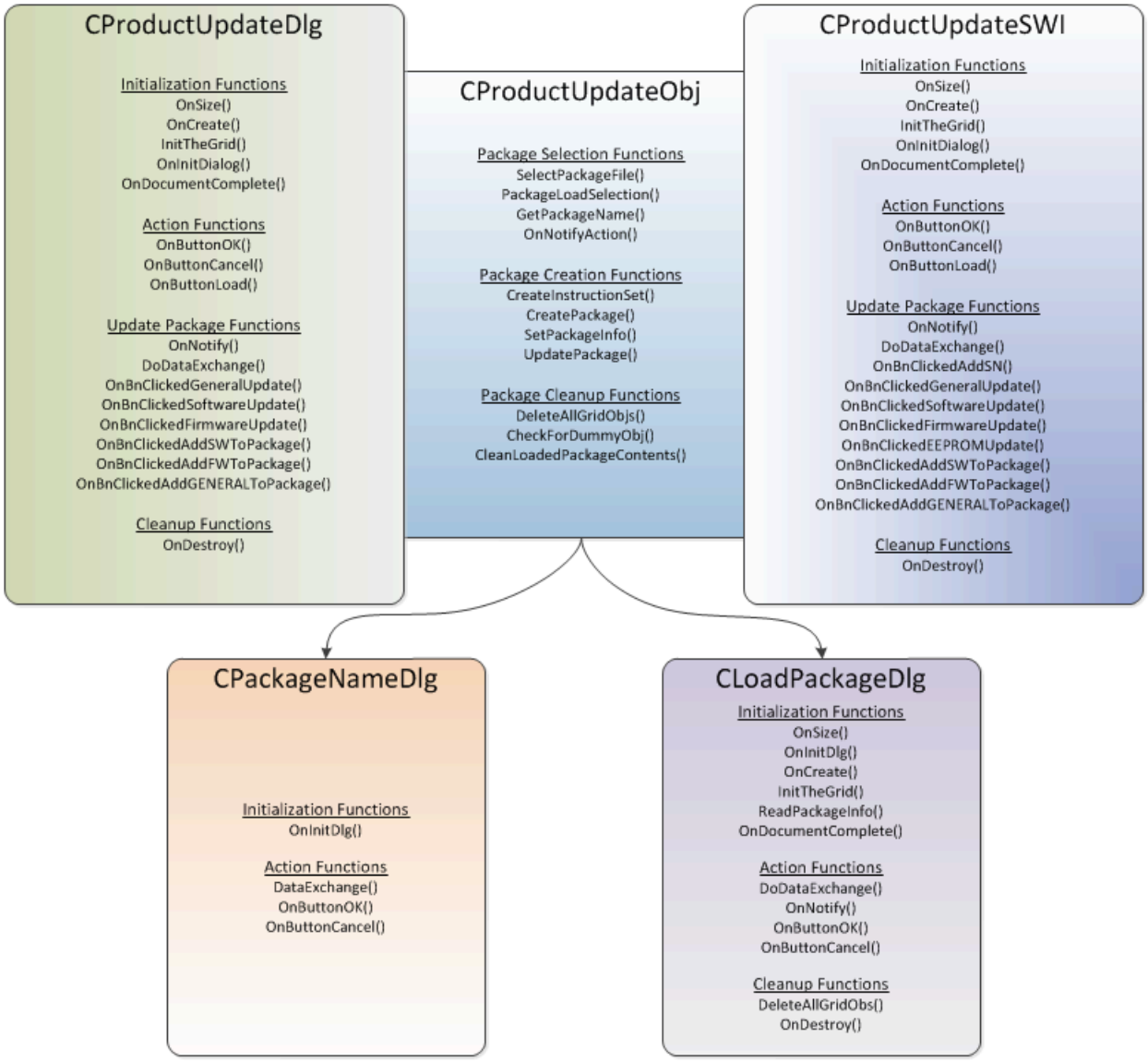
4.2 Product Update Create/Load GUI - Representative Implementation

In order to create the designed Representative Implementation of the Product Update Create/ Load GUI, a CDHtml Dialog class **ProductUpdateSWI** was created. Along with the necessary HTML controls, their associated member variables and event functions, a third party grid control interface (Dapfor) was also added to the dialog which also utilized the previously created **ProductUpdateGrid** CDataObject class as well as a **ProdUpdateCustomDrawDelete** and **ProductUpdateCustomDraw** CCustomDraw classes in order to manipulate the contents of the grid.

Similar to the Customer Implementation of the interface, the **ProductUpdateObj** object was utilized to perform the actions associated with creating a System Update Package. Since the Representative version of this interface has additional functionality associated with EEPROM option settings, as well as the ability to tie packages to specific serial numbers, some additional functionality and references to a separate **EEPROM_ConfigDlg** do exist within this dialog class. Details describing the **ProductUpdateObj** object will be provided in **Section 4.3** below. Additional details describing the **EEPROM_ConfigDlg** will be provided in **Section 4.6**.

4.3 ProductUpdateObj Class & Class Relations

As mentioned in **Section 4.1** and **4.2**, the **ProductUpdateObj** was created in order to minimize duplicated code which would have been needed in the implementation of both the Customer as well as Representative Product Update Create/Load GUI. The following diagram more clearly depicts the function relationships implemented in the development of this portion of the project.



The noticeable differences between the **ProductUpdateDlg** and **ProductUpdateSWI** CDHtml dialog classes shown in the diagram are purely the added functionality not intended for customer use.

The **PackageNameDlg** and **LoadPackageDlg** CDHtml dialog classes shown attached to the **ProductUpdateObj** object will be described in the following **Section 4.4** and **4.5**.

4.4 Product Update Package GUI

In order to create the designed Product Update Package GUI, a CDHtml Dialog class **PackageNameDlg** was created. This simple dialog purely contains the necessary HTML controls, their associated member variables and event functions to allow the Package Name and Description to be entered and saved.

As shown in **Section 4.3**, this class is only instantiated from within the **ProductUpdateObj** object when it is called via either the Customer or Representative version of the Product Update Create/ Load GUI.

4.5 Product Update Load Package GUI

In order to create the designed Product Update Load Package GUI, a CDHtml Dialog class **LoadPackageDlg** was created. Along with the necessary HTML controls, their associated member variables, and event functions, a third party grid control interface (Dapfor) was also added to the dialog. This grid control also utilizes the previously created **ProductUpdateGrid** CDataObject class as well as both the **ProdUpdateCustomDrawDelete** and **ProductUpdateCustomDraw** CCustomDraw class in order to manipulate the contents of the grid.

As shown in **Section 4.3**, this class is only instantiated from within the **ProductUpdateObj** object when it is called via either the Customer or Representative version of the Product Update Create/ Load GUI.

4.6 Product Update EEPROM GUI (REP-ONLY)

In order to create the designed Product Update EEPROM GUI, a CDHtml Dialog class **EEPROM_ConfigDlg** was created. This simple dialog purely contains the necessary HTML controls, their associated member variables, and event functions to allow the various attributes which are stored in the EEPROM of the sensor hardware to be established.

As outlined in the design portion of the document, all of the specified attributes are stored in memory until the package is created and password protected in order to prevent manipulation other than directly through the EEPROM GUI. Access to this dialog, as well as the Representative Implementation of the Create/Load GUI requires the presence of the HW Key dongle. Code to check for the presence and attributes of an attached dongle were made available and were not developed as part of this project.

4.7 Package XML Instruction Set

Performed within **ProductUpdateObj**, the XML Instruction Set is created and formatted once the contents of the update package have been specified by the user. The sequence of entries within the XML file are dictated by the entry order of selections into the Product Update Create/Load GUI, and the format of the file is described in **Section 3.7**. One large string is constructed containing all of the Instruction Set information formatted into proper XML format before it is written to the InstructionSet.xml file located within a directory named according to the Package Name specified by the user.

4.8 File Manipulation & Package Creation

Once the user selects to save a Package and the appropriate information has been entered and recorded (**PackageNameDlg**), a directory is created based upon the specified Package Name. Any files which had been specified by the user to be added to the package are then copied and placed within this directory along with the created XML Instruction-Set. Finally a copy of the SensorAgent is placed within the directory before it is zipped and password protected to prevent tampering. All of these actions are performed via standard APIs.

4.9 Remote Target Selection

In order to create the designed Remote Target Selection dialog, a CDHtml Dialog class **ProductUpdateRemoteSelect** was created. Along with the necessary HTML controls, their associated member variables and event functions, a third party grid control interface (Dapfor) was also added to the dialog which also utilized the previously created **ProductUpdateGrid** CDataObject class as well as a **ProdUpdateCustomDrawDelete** and **ProductUpdateCustomDraw** CCustomDraw classes in order to manipulate the contents of the grid. Once the user makes their selections indicating the package destination(s), a function call is invoked to transfer the package as specified.

4.10 Transfer The Update Package

The implementation of transferring the update package was performed via the existing file transfer capabilities present in the application. In the case where the machine being used to deploy the update is an UA, the package is sequentially sent to each UB specified by the user in the Remote Target Selection interface. Once the package is received at each UB, the package is then deployed to each specified Sensor attached to that UB before the update is applied to all machines at approximately the same time.

4.11 Create SensorAgent Agent

In order to create the SensorAgent, a new project was started within which the necessary classes and DLLs were added in order to provide support for communicating with the hardware elements of the Sensor machines, as well as to provide the capability to securely communicate between the different machine types. The ProductUpdateObj was also added in order to utilize the previously implemented code to unzip the update package, parse the XML Instruction-Set, interpret the instructions to be executed, as well as perform necessary file clean up once the actions are performed.

4.12 SensorAgent: Performing Software Update

In the case where a Software Update was selected by the user in **Step 3.1/3.2**, an updated version of the Systemware software has been transferred to the remote system as part of the Update Package. The SensorAgent is aware of the name and location of the installation executable as a result of parsing the XML Instruction-Set. The local software running on the machine is terminated, and an uninstall is performed via command line function calls. Similarly, once the uninstall is performed, the new install is applied via the same command line capability.

4.13 SensorAgent: Performing Firmware Update

In the case where a Firmware Update was selected by the user in **Step 3.1/3.2**, an updated version of the Systemware firmware has been transferred to the remote system as part of the Update Package. The SensorAgent is aware of the name and location of the firmware bit image as a result of parsing the XML Instruction-Set. The local software running on the machine is terminated, and the firmware burn is performed using third party drivers to command the FPGA. Once this has completed, the system will send a message to the higher level UB to send a Wake on LAN packet in two minutes. Command line functionality will then be used to shut the machine down, to then await being powered back on by the Wake on LAN packet. (A power cycle is required to make the firmware image take hold.)

4.14 SensorAgent: Performing EEPROM Update

In the case where an EEPROM Update was selected by the Representative user in **Step 3.2**, the specified EEPROM settings were transferred to the remote system as part of the contents of the XML Instruction-Set contained within the Update Package. The SensorAgent is able to determine if it is running on an UB or Sensor node, and if it is running on a Sensor the main application will gracefully be terminated. Once closed, the SensorAgent will utilize third party drivers to set the applicable EEPROM settings as specified in the XML Instruction-Set. Once this has been performed, either the next instruction in the update package will be performed, or the system will be configured to an idle useable state.

4.15 SensorAgent: Performing General Update

In the case where a General Update was selected by the user in **Step 3.1/3.2**, an additional file or specific command was passed down to the remote machine via the update package / XML instruction set. Since an updated copy of the SensorAgent will have been transferred to the remote system as part of the Update Package as well, the intended logic required to perform the update should be processed by the SensorAgent without incident. Currently, the SensorAgent is configured to gracefully exit the main application. Once closed, the SensorAgent will utilize command line functionality to attempt to execute the General Update file. Once this has been performed, either the next instruction in the update package will be performed, or the system will be configured to an idle useable state.

4.16 SensorAgent: System Configuration to Idle

Once all of the contents of the update package have been applied, the SensorAgent will configure the machine back to an idle useable state. Using the original capability of the CProductUpdateObj class, the update package along with any unzipped files will all be destroyed. Existing routines to perform secure messaging with the higher level (UA/UB) machine will be used to send status messages indicating the success or failure of the application of the update package. Similarly, all additional file system changes to allow for potential continuation of the SensorAgent after a reboot or power cycle will be undone to restore the machine back to its original state (with the exception of the updates performed). The SensorAgent will then be terminated and destroyed via command line functionality after a command has been sent to reboot the machine in 30 seconds.

When the system reboots, the UB/Sensor software will be automatically launched. The UA/UB machine will attempt and succeed to connect to the remote system. At this point all updates should have been performed and the system should be available to resume normal operation.

5 Performance Test Methods

Functional verification of the RSUM emphasizes testing performed to characterize proper system behavior under conditions affected by its application and utilization. Within these, several tests were performed to emulate end-users' and equipment manufacturer's representatives use in the field. Additionally, prior to incorporation into the final release, any additional candidate software modules are required pass preliminary acceptance testing. Each such test scenario is addressed in the following discussion.

5.1 Package Creation

Packages were implemented using both the End-User and Representative Package Creation dialogs, and subsequently re-loaded back into the Remote Update Package Contents dialog to verify that the contents specified were identical to the results displayed when originally loaded. Multiple packages were loaded in succession, each with different options to verify that no issues would occur. All options available were exercised and various package combinations were created and archived for deployment test. This test utilized all created interfaces with the exception of the Remote Target Selection GUI which will be utilized in the following exercises.

5.2 Localized Package Deployment

Initially, a representative package containing both a software and firmware update, and based on a local configuration containing one UB and several Sensors, was deployed to several Sensors using the packages created in **Step 5.1**. Following this deployment, acknowledgement messages were received/displayed, indicating that package delivery succeeded. Subsequently, follow-up messages confirmed the successful application of both software and firmware versions. Finally, the systems were automatically power cycled and reconfigured to a stand-by state. Once complete, the all versions of software and firmware were separately verified as being correctly updated to the version specified in the deployed package.

5.3 Large Scale Package Deployment

Using a process similar to that described in **Step 5.2**, the packages created and tested in **Step 5.1** were utilized to verify the functionality of updates applied from an UA to two different UBs (one physically located in Camarillo, and an additional UB located on the East Coast), each of which populated with several attached Sensors. As before, the package deployed for this test contained instructions for both software and EEPROM updates. The resulting displayed messages confirmed that package delivery was successful both at the UB-level and at the Sensor level for each machine involved. Next, the software update was automatically executed resulting in a success update message at the UA-level, followed by an EEPROM update success message being displayed at the UA-level for the corresponding Sensors, and all of the systems being automatically restarted and re-connected in a stand-by state. Following this exercise, the distributed software and EEPROM settings were individually verified and shown to have been correctly updated to the version specified in the deployed package.

6 Test Results

6.1 Mechanism Performance

Measurements were taken recording the time elapsed when performing updates to Sensors manually, as well as through the developed mechanism. The following charts describe the benefits observed as a result of automating the process, as well as the time saved since actions are performed in parallel rather than sequentially by an individual.

Emphasis is given to the Software Update and Firmware Update processes since EEPROM updates could not be remotely applied prior to this development, and the time required to perform General Updates will likely be inconsistent.

The times listed in the tables below for performing the Software Update are based upon the assumption that the user applying the updates manually maintains focus and does not become distracted during the process. The assumption that there are no observed variations in machine performance, and that updates are performed on Unit B's simultaneously, causes no discernible difference in update times to be listed for the mechanized application of the software updates.

Software Update - Manual Application			
	One Sensor	**Ten Sensors**	**One Hundred Sensors**
Uninstall	43 Seconds	7 Minutes, 10 Seconds	1 Hour, 11 Minutes, 40 Seconds
Install	15 Seconds	2 Minutes, 30 Seconds	25 Minutes
Remote Desktop	20 Seconds	3 Minutes, 20 Seconds	33 Minutes, 20 Seconds
Folder Navigation	15 Seconds	2 Minutes, 30 Seconds	25 Minutes
Restart	5 Seconds	50 Seconds	8 Minutes, 20 Seconds
Total	1 Minute, 38 Seconds	16 Minutes, 20 Seconds	2 Hours, 43 Minutes, 20 Seconds

Software Update - Mechanized Application			
	One Sensor	**Ten Sensors**	**One Hundred Sensors**
Uninstall	43 Seconds	43 Seconds	43 Seconds
Install	15 Seconds	15 Seconds	15 Seconds
Remote Desktop	N/A	N/A	N/A
Folder Navigation	N/A	N/A	N/A
Restart	5 Seconds	5 Seconds	5 Seconds
Total	1 Minute, 3 Seconds	1 Minute, 3 Seconds	1 Minute, 3 Seconds

The times listed in the tables below for performing the Firmware Update are based upon the assumption that the user applying the updates manually maintains focus and does not become distracted during the process. The assumption that there are no observed variations in machine performance, and that updates are performed on Unit B's simultaneously, causes no discernible difference in update times to be listed for the mechanized application of the firmware updates.

Firmware Update - Manual Application			
	One Sensor	**Ten Sensors**	**One Hundred Sensors**
Remote Desktop	20 Seconds	3 Minutes, 20 Seconds	33 Minutes, 20 Seconds
Folder Navigation	15 Seconds	2 Minutes, 30 Seconds	25 Minutes
Launch FW Application	10 Seconds	1 Minute, 40 Seconds	16 Minutes, 40 Seconds
Select .Bit File	10 Seconds	1 Minute, 40 Seconds	16 Minutes, 40 Seconds
Burn FW	3 Minutes	30 Minutes	5 Hours
Shutdown	5 Seconds	50 Seconds	8 Minutes, 20 Seconds
Wake on LAN	10 Seconds	1 Minute, 40 Seconds	16 Minutes, 40 Seconds
Total	4 Minutes, 10 Seconds	41 Minutes, 40 Seconds	6 Hours, 56 Minutes, 40 Seconds

Firmware Update - Mechanized Application			
	One Sensor	**Ten Sensors**	**One Hundred Sensors**
Remote Desktop	N/A	N/A	N/A
Folder Navigation	N/A	N/A	N/A
Launch FW Application	N/A	N/A	N/A
Select .Bit File	N/A	N/A	N/A
Burn FW	3 Minutes	3 Minutes	3 Minutes
Shutdown	5 Seconds	5 Seconds	5 Seconds
Wake on LAN	10 Seconds	10 Seconds	10 Seconds
Total	3 Minutes, 15 Seconds	3 Minutes, 15 Seconds	3 Minutes, 15 Seconds

Configurations which contain more Sensors requiring Software and or Firmware updates will observe more significant time savings when utilizing the RSUM to apply the updates. Beyond the time elapsed, the entire user experience will be greatly improved through the elimination of steps and potential for error.

7 Concluding Remarks

7.1 Development Summary

This report documents the requirements definition, design, implementation and performance verification of a custom, remote update tool for a proprietary signal analysis instrumentation product-set. Because of the sensitive nature of its purpose and intended application, details in that regard have been intentionally excluded.

Extensive testing and performance characterization of a variety of typically encountered equipment configurations has disclosed no anomalous behavior when functionality exercised with the system enhancements described in this report. Moreover, RSUM has been shown to provide a remarkably more efficient, concealed method for simultaneously updating multiple system elements with the latest revisions of software, firmware, EEPROM settings, and/or general updates than was previously possible. While future refinement may be introduced to allow additional functionality, the overall goals of this project have been successfully accomplished.

8 Anticipated Future Enhancements

Virtually every complex tool developed benefits from enhancements (i.e., updates) made following its initial deployment. Some of these fall into the category of system utilization, command and control (e.g., scheduling), while others are associated with diagnostic features required for investigating various types of performance anomalies (e.g., debuggers). The need for others (as shown by experience) will not become apparent until many systems have been fielded and user feedback examined. Such features already considered relevant for next-release implementation are described below.

8.1 Scheduled Package Deliveries

Time-coordinated distribution and installation of product updates is an essential capability for optimum system-level performance management by the end-item users' and/or the original equipment manufacturer's technical representative. Accordingly, user-defined scheduling of the transmission and/or application of system updates to off-peak hours is a commonly provided feature [18,22,25-26]. Thus, by affording system managers this capability, equipment functional loading can be optimized, balancing the demands of system administration with equipment utilization. Future enhancement plans include development and incorporation of such a scheduler feature.

8.2 Additional Logging / Status Updates

The burgeoning complexity of system installation packages, and their associated overall upgrade processes, increases the necessity for clear and thorough status logging to facilitate determination and resolution of any issues encountered during product update deployment, installation and debugging. Considered an important value-added feature for investigating unexpected problems encountered during the system update process, implementation of additional logging and status updates is also intended in next release plans for this project.

9 References

[1] System Center Configuration Manager: Overview http://www.microsoft.com/systemcenter/en/us/configuration-manager/cm-overview.aspx Last Viewed August 20, 2010

[2] Apple Remote Desktop http://www.apple.com/remotedesktop/ Last Viewed August 20, 2010

[3] Introduction to Software Update Channels http://msdn.microsoft.com/en-us/library/aa740931%28VS.85%29.aspx Last Viewed August 22, 2010

[4] Update Center - Software Update UI Requirements, Design and Implementation http://wiki.updatecenter.java.net/Wiki.jsp?page=SoftwareUpdate Last Viewed August 24, 2010

[5] Wen-Kang Wei; Kuo-Feng Ssu; Jiau, H.C. Implementation of Nonstop Software Update for Client-Server Applications. Department of Electrical Engineering, Nat. Cheng Kung University., Tainan, Taiwan, 2003

[6] Windows Update http://en.wikipedia.org/wiki/Windows_Update Last Viewed September 2, 2010

[7] Automatic Updates - ubuntu https://wiki.ubuntu.com/AutomaticUpdates Last Viewed September 2, 2010

[8] Patch (computing) http://en.wikipedia.org/wiki/Patch_%28computing%29 Last Viewed September 2, 2010

[9] Apple Software Update http://en.wikipedia.org/wiki/Apple_Software_Update Last Viewed September 2, 2010

[10] D. White. Limiting Vulnerability Exposure Through Effective Patch Management, Rhodes University, 2006

[11] Linux Kernel Patch Format http://linux.yyz.us/patch-format.html Last Viewed September 2, 2010

[12] EEPROM http://en.wikipedia.org/wiki/EEPROM Last Viewed September 2, 2010

[13] Firmware http://en.wikipedia.org/wiki/Firmware Last Viewed September 2, 2010

[14] Software Agent http://en.wikipedia.org/wiki/Software_agent Last Viewed November 2010

[15] Belief-Desire-Intention Software Model http://en.wikipedia.org/wiki/Belief-Desire-Intention_software_model Last Viewed November 2010

[16] Using XML Access to Deploy Portlet Services http://www.ibm.com/developerworks/websphere/library/techarticles/0303_buczak/ buczak.html Last Viewed September 2, 2010

[17] D. Metler. Secure software updates for mobile devices. Master's Thesis, Department of Information Technology, University of Zurich, July 2002

[18] S. Ajmani, B. Liskov, and L. Shrira. Scheduling and Simulation: How to Upgrade Distributed Systems. In HotOS IX, May 2003.

[19] I. Neamtiu, M. Hicks, G. Stoyle, and M. Oriol. Practical Dynamic Software Updating For C. In Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI), June 2006.

[20] B. C. Popescu, B. Crispo, and A. S. Tanenbaum. Secure Data Replication Over Untrusted Hosts. In HotOS IX, May 2003.

[21] A. Bellissimo, J. Burgess, and K. Fu. Secure Software Updates: Disappointments and New Challenges. Department of Computer Science, University of Massachusetts Amherst, 2006

[22] C. R. Hofmeister and J. M. Purtilo. A framework for dynamic reconfiguration of distributed programs. Technical Report CS-TR-3119, University of Maryland, College Park, 1993.

[23] J. Kramer, and J. Magee. The Evolving Philosophers Problem: Dynamic change management. *IEEE Transactions on Software Engineering*, 16(11), 1990.

[24] T. Ritzau, and J. Andersson. Dynamic Deployment of Java Applications. In Java for Embedded Systems Workshop, 2000.

[25] S. Ajmani. Automatic Software Upgrades for Distributed Systems, PhD Thesis, MIT, 2004

[26] S. Ajmani, B. Liskov, and L. Shrira. Modular Software Upgrades for Distributed Systems, In European Conference on Object-Oriented Programming (ECOOP), July 2006.

[27] P. Ames. Agents that Reduce Work and Information Overload, CACM, 7, 31-40, 1994

[28] C. Guilfoyle, and E. Warner. Intelligent Agents: the New Revolution in Software, Ovum, 1994

[29] Systemware Inc., http://www.caci.com/systemware/ Last Viewed November 2010