

ABSTRACT

In today's world, the car is not just a machine used for regular transport – it has become a gadget. And, as with any gadget, it has both software and hardware. Hardware is solid and stable, and is designed to work for years. Software is flexible, unreliable and full of features. The typical lifecycle of software is much shorter compared to hardware. And, people want new software features.

But, after the car leaves the gates of the factory, the manufacturer cannot update the software. Any defects, issues, inconveniences, and non-implemented features remain with the vehicle over its entire life. This is a problem which Luxoft solves with remote reflash.

In this whitepaper, Luxoft describes its approach to remote software updates, which can be performed in the factory, dealer's garage or in the field; anywhere the car has Internet access via Ethernet, WiFi or cellular link. Luxoft's approach to software upgrade is safe, resource effective and platform independent. It can be used to upgrade the traditional QNX, GreenHills, RTXC or other RTOS-based firmware bundles, as well as the more modern Linux/Android or Windows Auto images.

Luxoft has developed a more advanced solution, upgrading the necessary application without touching the OS or other functionalities. A detailed whitepaper will be published upon completion of the proper R&D project.

INTRODUCTION



The last ten years have brought outstanding changes to the traditional telematics concepts. Starting from remote car access and automation, it migrated through communication and safety to infotainment and multimedia. And when discussing in-vehicle telematics, we mostly talk about PC or PDA-like functionality adopted for use in very specific car driving conditions.

These conceptual changes impact many things, changing the architecture and design approaches for typical telematic applications.

At the start of the in-car infotainment epoch, most of the software was created like typical embedded firmware with a monolithic structure, targeted addressing and direct peripherals control. But while this design is very convenient for creating stable and fast control applications, it is not suitable for user-oriented programs on the edge of modern technological

achievements. Additionally, it significantly slows down the entire development cycle as it requires the long laboratory-based procedure for software update whether defect issues, applying enhancements or just adding end user functionalities.

The design of modern infotainment systems has travelled a long way from this classic concept. It assumes small firmware (historically named 'boot loader', but practically being a kind of BIOS for

automotive software), multi-purpose operating system with limited real-time capabilities (but having advanced UI support), and a number of user-targeted applications that provide functionality mostly oriented to the driver's and passengers' convenience rather than to control functions support. This evolution was not smooth, and is not finished yet, so it is not surprising that there are some aspects remaining. The most valuable remaining aspect is the software upgrade.

Software upgrades were not considered a priority in the automotive software world for many years. While automotive systems were considered monolithic embedded devices, it was assumed that hardware and software lifetimes were the same. In the worst case scenario, software must be re-flashed in the factory or in the dealer's garage, but only on an exceptional basis in case of critical errors or safety problems. Now the user wants to see many features, which cannot be pre-developed, nor can they be installed during system design. Therefore, modern car infotainment systems are more like regular computers with several hundred million lines of code, pluggable modules and installable applications. This kind of system cannot be developed as 100% bug free software. And, its software content becomes obsolete much faster than the hardware or the car itself. This brings the task of software upgrades back to the top of the priority list for the automotive segment.

CLASSIFICATION

At this point in the evolution of automotive telematics systems, there are five major scenarios of the software update to be covered:

- Upgrade in the development laboratory. This type of upgrade produces minimum restrictions as it is done in a controlled environment, and by trained staff. However, we must take into account that typical development of automotive systems is distributed worldwide, and hardware might be far away from the software development centers. Also, shipment of the target equipment and required tools requires significant investment which produces additional costs. This is the traditional approach which has kept its value for epochs, but is not enough for the current state of art.
- Upgrade at the factory. This is required when the version of software is changed, or multiple application versions are used with the same target hardware. This situation is very typical for car manufacturers and tier one suppliers, who produce their systems for multiple markets, languages and vehicle classes. This type of update needs much more care as factory staff is less trained and qualified than engineering teams. Additionally, it must be oriented for mass sequential re-flash of many units in a short time period. This method is obvious and unavoidable, however it is usually more related to the manufacturing process than to the software features itself.
- In-garage upgrade. This scenario is important for controllable software replacement when fixing defects or applying enhancements. This must be fully automated to avoid problems for dealer staff which are not controlled from the software vendor's side; however, it is still being performed in a partially controlled environment with relatively unlimited time conditions, and without irremovable concurrent activities. Actually, many vendors apply relatively stable, and widelydistributed methods

of, in-garage upgrades using CD/DVD (for radio units and head unit systems) or using debug interface (usually UART or CAN bus).

- In the field upgrades. This kind of upgrade might be performed by the end user, or perhaps by the test driver during the vehicle equipment field test. This type of upgrade is performed in a practically uncontrollable environment with non-qualified staff, and potentially in conditions which cannot be fully managed (for example, unstable data transfer channels, parallel background activities, etc.). In the past, field upgrades were considered exotic and unsafe, however when network capabilities provide relatively fast and reliable data delivery, the field upgrade becomes an obvious alternative to an in-garage upgrade.
- Separate user-initiated application downloads. This is a very specific type of software upgrade, which depends on the entire architecture and capabilities of the HMI design. At this point in automotive technologies, there is no common standard for this type of software update, and this task cannot be solved together with the previously listed methods of software updates. Many companies, like Daimler and BMW, are now considering, or even working on, head unit systems which have the capability to download and install applications in runtime as a way to implement a kind of vehicle-oriented application store. However, these activities are in the R&D phase still, and their future is not obvious due to the absence of a standard infrastructure-compatible automotive software platform, which might be used for open systems development by third party vendors.

In this whitepaper we do not discuss any methods of the user application downloads due to the very specific nature of this task. Luxoft's technology provides a common method to solve the first four tasks and focuses on the most complex (the field re-flash), assuming other problems are particular cases of this one.

PROBLEMS



The issue of controllable and automated software upgrades is not specific to the automotive industry. The industry of PC, PDA, and smartphones software produces a number of methods, which can be reused for car devices as well. However, the in-vehicle environment has its own specific characteristics, which cannot be simply ignored.

First of all, the car environment is very sensitive. If a PC or smartphone software is corrupted during an upgrade, the device can be delivered to the service center and repaired for a relatively low cost. In the worst case scenario, it must be entirely replaced with a new one. This

definitely creates the additional costs, reputation issues and inconvenience, but rarely safety problems. In the case of vehicle software, this can impact the driver's safety, the ability even to travel to the service center, and definite costs, as a car is a very expensive device in the scale of the computer world.

The second problem is connectivity. PCs and smartphones are enabled with a set of peripheral connections, and additional equipment can be connected for software upgrades. At this time, the current vehicle head unit is not equipped with open interfaces that are allowed for usage in the field. It may have NAD for data communication over the air, Bluetooth for limited access to the user's telephony and... typically, that's it. It usually has access to CD/DVD, SD-cards and other media, but writing software to these media drives appears to be a complex and inconvenient procedure.

Finally, the vehicle is a device for everybody. When a user buys a PC, he is usually ready to manage the software, being a native part of the computer. But, the car driver views the car as four wheeled road transport, not a computer device, and is not typically willing to deal with complicated and unclear procedures to manage a software upgrade of his multimedia system. Therefore, this process must be as transparent for the user as it possible.

In addition to these major problems, there are some technical ones, which appear minor, but are very valuable in practice. In particular, car multimedia is not an open system by design. It has very limited resources and cannot be expanded with additional persistent storage to keep the downloaded software, or provide a faster data channel to deliver it.

MANDATORY REQUIREMENTS

Summarizing the problems above, we can say that the technology of the automotive-graded software update must meet some mandatory requirements:

- Safety. It should not impact the user, nor require his attention while he is driving the car.
- Security. Downloadable software must be compatible with hardware, data and settings already existing in the system, and must be protected from uncontrollable invasion.
- Reliability. The process should be ended successfully even if it is interrupted. The process should always produce the correct behavior of the upgradable system.
- Speed. The entire upgrade must be as quick as possible.
- Controllability. The upgrade process should be expected and approved by the user to avoid negative reactions from his side.

SUBTASKS



These requirements generate a set of subtasks that the automotive software upgrade is broken into.

1. Which part of the software is to be upgraded, and how is the entire system to be organized to follow the requirements and restrictions listed above?
2. Which data transfer channel is to be used for software delivery?
3. How is the new software to be applied instead of the older one?
4. How is the entire process to be triggered and who shall manage its initiation?
5. What is the method to ensure that the entire procedure meets the requirements?

The Luxoft's Remote Reflash Technology solves all these subtasks, and answers the appropriate questions. Surely, it is just one of the possible approaches, but it is optimal for many tasks, assuming a software upgrade in the various conditions listed above.

DATA TRANSFER CHANNEL



TCP/IP is the only data transfer channel that seems to cover all the required software upgrade scenarios. Luxoft's approach assumes that HTTP over TCP port 80 is used to download software upgrade descriptions and patches. This provides an opportunity to use existing Web infrastructure to support the Reflash on the server side.

In practice, TCP/IP is available in all contexts where the upgrade procedure is to be performed. For laboratory and factory conditions, it is typically used in the form of a corporate Intranet. For the garage, it is a regular Internet connection via landline. For field scenarios, NAD is used for TCP/IP data transfer. For devices having no NAD onboard, Bluetooth™ may be used to get TCP/IP over consumer electronics using DUN or PAN profiles.

It is assumed that the Boot Loader supports a reliable method for getting a TCP/IP connection with the server with the given DNS name, and getting the updates catalogue. Then, it selects the freshest of the compatible upgrade sets using compatibility information in the catalogue, and downloads the upgrade data and applies it to the Main Software and File System.

MAIN SOFTWARE UPGRADE

Main Software is typically large, and persistent storage may not have enough memory to save the second copy of its image. From the other side, it is typical that the changes in main software are rather small, especially in the case of fixing defects or changing just some functions without changing others. That's why Luxoft's approach is to store and deliver upgrades in patch format. Each patch has a pointer to the memory space to be changed (practically, executable files) and data to be applied instead of the entire original content.

Patches are grouped into packets of four kilobytes in length, which can be easily kept in Boot Loader memory. Each packet is signed to prevent change by an unauthorized user. After the packet is downloaded from the server, the Boot Loader unpacks it to the separate patches, and applies every patch from the packet. When all patches are applied, Boot Loader runs with the next packet.

Prior to starting the entire process, the application image is invalidated by an invalidation flag to avoid launching the application until all patches are applied.

After all packets are loaded, and patches applied successfully, Boot Loader clears the invalidation flag to mark the Main Software as valid. When this is done, the updated version of the software can be launched in the normal manner (via system reboot).

FILE SYSTEM UPGRADE

The method of File System upgrade is similar to the main software upgrade, but the patches point out which files/database elements are to be replaced, and provide the replacement rules. These rules take into account the original content of the data elements as these files may contain unique information which should not be replaced with the common content downloaded from the server. The File System upgrade is done in the same series as the Main Software upgrade (using packets), and the application integrity is protected with the same invalidation flag to prohibit launching the Main Software until all related data are aligned with the newly delivered executables.

TRIGGERING THE PROCESS



Luxoft's approach assumes three separate methods to start the re-flash process. The first is "soft-initiated" to be applied for in-the-field upgrades. The Main Software triggers the re-flash at the user's demand, sets the invalidation flag, and performs a system reboot. It enables the user with a safe and controllable method to start the software upgrade when, and where, it is convenient for him.

The “Hard” method is the special DIO input triggering re-flash process used in the factory. When the system detects the trigger signal, the Boot Loader unconditionally starts the software upgrade process until it is completed, and the Reflash DIO input is cleared by factory equipment.

The “Safety proving” method assumes that the Boot Loader checks the invalidation flag, and Main Software consistency at startup, and runs the software update if the software is invalid or incomplete. This method guarantees application download will continue until the software is completely loaded and verified.

ADVANTAGES OF THE PROCESS

Luxoft’s approach to software re-flash covers almost all typical upgrade scenarios, from the development laboratory to the field. It is applicable for modules with limited resources, and is reliable and fully automated (except the launch). This approach doesn’t require trained staff, and is applicable to various types of software images. The Patching system minimizes data traffic and time to load.

DISADVANTAGES OF THE PROCESS

Luxoft’s approach to software upgrade assumes the Main Software remains invalid while the update is ongoing. In the case of an interruption, the update is started again from the beginning.

SUMMARY

Upgrading software remotely doesn't initially seem to be rocket science. However, now we have incredibly advanced technologies that are developing automotive systems rapidly and effectively. This is important. Suppliers need to be provided with the proper hardware. Firmware needs to be kept fresh and updated. Manufacturing hardware must be accessed at all times. Sold units need maintenance.

All these tasks make product management much more complicated than necessary. Luxoft does not want to revolutionize the field. We do not pursue blue sky development. We do not need to be the first or the most innovative. We want to solve real problems, your problems.

We will give you a tailored approach to speed up automotive systems development. This is not rocket science and not a game changer, but it works and it’s helpful. That's not bad, isn't it?

Luxoft

You steer, we accelerate.

Michael Minkevich,
VP Technology Services

Tel.: +7(495) 967-80-30 ext. 4427

E-mail: MMinkevich@luxoft.com