The Microsoft® Telematics Platform:
A Platform for Smart Telematics Systems

Paul R. Kappes
Edward J. Lansinger
Microsoft Corporation

ABSTRACT

The changing landscape of telematics demands a new approach to development of telematics systems. This paper presents an overview of the Microsoft® telematics platform architecture for general purpose, in-car compute platforms.

INTRODUCTION

Experience with the world of PCs, cell phones, personal digital assistants (PDAs), and other "smart" consumer electronics has changed how consumers expect to interact with technological products. Common themes among these products include rapid access to information, communications, entertainment, friendly user interfaces, and integration with services that go beyond the functionality contained in the device itself. These devices also benefit from a virtuous cycle that keeps the devices and related services fresh and interesting to consumers, which in turn spurs increased adoption and attracts more technology suppliers to the platform. This cycle is enabled by a fundamental openness that allows many third parties to find ways to add value to the basic system. Telematics systems will be responsible for turning automobiles into similarly "smart" devices.

The industry is at an inflection point now with the availability of several key technologies that can finally be integrated into vehicles at cost points that will allow mass-market adoption. Recognizing this, Microsoft developed a telematics platform architecture as the basis for new telematics systems it is developing with OEM and Tier 1 partners. This architecture has already been successfully used for both prototypes as well as production-intent systems. It addresses the primary concerns of cost and reliability while providing the computational power and connectivity required of smart devices. Moreover, it anticipates the need for rapid innovation in telematics solutions that will drive widespread market acceptance. It does this by balancing the requirement for openness with the need for OEMs to maintain system integrity and control brand image.

This paper covers:

• Key enabling technologies for telematics supported by the platform
• Platform hardware architecture
• Platform software
• The platform as an enabler of the virtuous technology cycle
ENABLING TECHNOLOGIES FOR TELEMATICS

Recent advances now make it possible to create telematics systems that match customer expectations for smart technology. Newly advanced technologies include:

Wireless data networks: Long awaited, it is now possible to get a wireless connection to the Internet along most public roads and highways using CDMA- or GPRS-based “2/2.5G” networks. This will enable a wealth of information and services to be delivered to vehicles in motion.

Speech synthesis: Customers won’t accept stilted “robo-speak” from their car anymore (if they ever did), such as the infamous “a door is ajar” utterance from certain cars in the 1980s. In addition to significant advances made in voice synthesis algorithms since those times, a key element in making natural speech a reality has been the drop in memory prices. Sufficient memory (now in the 8 to16MB range) is required for natural-sounding speech due to the need to store speech recorded from voice talent as well as high fidelity model parameters for entirely synthetic speech.

Speech recognition: Recognizing spoken commands with high accuracy is a computationally-intensive process involving real-time filtering, noise reduction, and pattern matching. Microprocessors of sufficient power are now available at cost points that make this possible.

Global Positioning Satellites (GPS): Three major advances have happened in recent years that dramatically improved suitability of GPS for automobile applications. The first was the turning off of Selective Availability, increasing positioning accuracy by a factor of ten, down to about one road-width. The second advance was the development of fast acquisition and re-acquisition techniques so GPS position fixes are available shortly after key-on and immediately after passing under obstructions. The third advance was the development of hybrid GPS/dead reckoning systems that could accurately predict vehicle position during short-term outages such as when traveling through tunnels.

Bluetooth: This short-range wireless technology is positioned to become the major means by which cell phones, PDAs, laptops, and other smart technology devices will be able to communicate directly with vehicles.

Control Area Network (CAN) networks are now replacing the low-speed, proprietary networks that historically have interconnected vehicle electronic systems. Because of the standardized nature of CAN, telematics systems are more easily able to extract useful information directly from the vehicle, like diagnostics.

Operating Systems: The adoption of well-known operating systems (e.g. Windows® Automotive) with a lineage to general-purpose desktop operating systems marks a significant departure from earlier efforts that used home-grown or specialized embedded operating systems. This element is critical to the creation of a virtuous cycle for rapid innovation of telematics systems, in that it greatly expands the available base of software developers that are able to create applications for these telematics systems. Such a change is also necessary because telematics devices combine functionality from previously separate modules (e.g. navigation, trip computer, stereo system) into multi-function single modules, and this additional complexity demands a greater level of sophistication from the operating system to manage the running of multiple applications.

TELEMATICS HARDWARE ARCHITECTURE

Although the platform is designed specifically for automotive applications, the basic architecture built around a 32-bit CPU with memory management is similar to many “smart devices” on the market today. This familiarity is an important component in attracting developers to the platform.
Throughout the development of the architecture, and its realization into prototype and production intent systems, much attention was paid to making it automotive-grade. All parts are available in automotive temperature grade from suppliers that are able to meet automotive qualification standards. PCB fabrication and assembly require no processes that are not already in use for production automotive electronic systems. Its ~1.5 mA power consumption when "off" meets typical automotive requirements for long-term current drain.

The following is a block diagram of the architecture:

### Hardware Architecture

- **SDRAM**: 32-512 MB
- **CPU**: 300-400MHz (e.g. Samsung 24xx)
- **JTAG**: Debug
- **MMC/SD**: USB Device
- **LCD ctrl**: I2S
- **UART UART**: 16 bit
- **A/D**: and GPIOs SPI
- **USB Host**: NAND I/F
- **FPGA**: 200K-400K gates
- **Hardware Architecture**: NAND Flash 32-512 MB
- **Bluetooth**: PCM I/F
- **Microphone**: Stereo CODEC
- **Mic in**: Line out
- **GPS**: CAN
- **Cellphone Module Data/Voice**: SIM socket
- **Serial Debug**: UART
- **USB Flash Drive**: Flash Drive
- **Mono CODEC**: GSM Voice Out
- **GSM Voice In**: I2S I/F
- **Central Processing Unit (CPU)**: This is a 32-bit processor designed for embedded use and available in automotive temperature grade. Prototypes have been based on Samsung 24xx processors in the 300-400 MHz speed range. This provides sufficiently scalable computational power to handle intensive tasks such as speech recognition while servicing network connections to the vehicle bus and the Internet, with the headroom to allow many client applications to run simultaneously.

- **Field Programming Gate Array (FPGA)**: This architecture realizes specialized logic in an FPGA programmable logic device. Historically, an Application-Specific Integrated Circuit (ASIC) would be used for this functionality as a cheaper though much less flexible option (due to the long lead times and high fixed costs involved in ASIC design). However, FPGA costs have come down dramatically in recent years. The reprogrammable nature of the FPGA, particularly in light of the rapidly changing telematics landscape, now generally gives it the edge over the ASIC for products up to moderately high volumes (e.g. hundreds of thousands of units per year).

- **Prototypes** have used Xilinx Spartan III FPGAs, taking advantage of their Digital Signal Processor (DSP) capabilities to offload intensive DSP processing from the main CPU.

Memory: 32 MB Flash memory and 32 MB DRAM are currently viewed as the practical minimum for telematics systems of typical functionality. This is in large part driven by the requirements of speech recognition and speech synthesis. The architecture can easily support as much as 1 GB of combined Flash and DRAM, so the basic architecture can scale up to very sophisticated systems.

Cell phone Module: High-end telematics systems have dedicated cellular modules that allow for data (and optionally voice) connections. (e.g. Siemens AC45)
Bluetooth: Bluetooth is an enabling technology for several scenarios. It provides the means by which a user can use the platform as a hands-free interface to their personal cell phone. But it is also used to share information with smart devices over a data connection, for example to connect to a PDA and extract contact list information. In low-cost systems that have no integrated cell phone module, it relies on an Internet connection through the user’s Bluetooth-equipped cell phone.

Controller Area Network (CAN) bus: The platform supports a CAN interface, which is becoming the worldwide standard for vehicle internetworking. It also supports LIN and K/Line. Although not included in the initial implementation; the underlying architecture could support the addition of other standards such as ISO 9141-2 and J1850. This addition would require some hardware and low level software rework.

Global Positioning System (GPS): A host-based GPS system is used to keep costs down. (e.g. SiRF) This means that the complex arithmetic for computing a position fix is performed by software running on the main CPU. Dead Reckoning is also integrated into the GPS software. Depending on target vehicle requirements, a yaw rate gyro may be added into the system to enhance accuracy, or Dead Reckoning may be based solely on wheel travel information available over CAN.

Audio: The platform has a microphone input to accept voice commands. It supports monophonic output for hands-free phone functionality, and stereo output which allows playback of media files in high fidelity on the vehicle’s audio system. It uses Text to Speech (TTS) technology to speak to the driver. (e.g. Scansoft)

Video: While the underlying architecture can supports both “headless” systems as well as systems with displays; initial implementations are headless. In a typical headless system, the user interacts primarily through voice. If a general-purpose display is available in the target vehicle, it can send messages over CAN to show visual output.

General-Purpose Input/Output: Analog and digital inputs as well as digital outputs are provided for interfacing to buttons, lights, etc.
MICROSOFT's TELEMATICS PLATFORM SOFTWARE

The platform software architecture aims to satisfy the following requirements:

- Reliable, consistent operation
- Seamless coordination of access to common resources for multiple applications
- Increased security for applications and data
- Rapid software development

The platform software comprises 4 major layers: The Windows CE operating system, the Microsoft .NET Compact Framework, the Automotive Framework Classes, and applications. The following diagram depicts those layers.
Software Stack

Windows CE operating system: This is a real-time embedded operating system based on the well-known Win32 API. This operating system was chosen in order to meet small RAM and ROM footprint requirements (keeping cost down). Included in the OS is the Board Support Package (BSP). The BSP’s low level software glues the OS to the platform’s hardware.

.NET Compact Framework: Typically, user-visible functionality comes from applications that run within the “managed code” runtime environment provided by the .NET Compact Framework. Managed code, a technique originally developed for desktop PC applications, improves system reliability in several ways. For one, the environment has checks to catch common programming mistakes, which helps to identify bugs earlier in the development process as well as provide graceful error handling. Managed code also provides code access security features that enforce security policies, allowing (for example) OEM software to execute with full privileges while third party software runs in a “sandbox” with limited access to system resources. This keeps an errant or malicious application from affecting the operation of the system as a whole.

The managed code runtime environment is called the Common Language Runtime (CLR). The CLR is the execution engine for .NET Compact Framework applications. Typically 3rd party applications execute in the CLR’s managed execution environment. The CLR provides a number of services, including:
- Code management (loading and execution)
- Application memory isolation
• Verification of type safety
• Access to metadata (enhanced type information)
• Managing memory for managed objects
• Enforcement of code access security
• Exception handling, including cross-language exceptions
• Support for developer services (debugging, and so on)

The .NET Compact Framework Classes form the building blocks of managed code applications. They abstract operating system services allowing developers to add functionality to programs in a consistent manner regardless of programming language or operating environment. While the .NET compact framework class libraries are a subset of the full Microsoft .NET Framework class library found on the desktop; it does contain all of the basic classes.

Automotive Framework: Special purpose software components, or “classes”, were created in order to make it easier for applications to access telematics functionality. Examples include classes for GPS and CAN messages. These classes are an add-on to the .NET Compact Framework. The automotive classes enable applications, written in managed code, to interact with the Automotive Middleware which provides access to underlying functionality built to support the device.

The automotive framework classes simplify application development by encapsulating the core application services required by all applications and complying with an application model that enables developers to design a speech user interface (SUI) for their application using eXtensible Markup Language (XML).

The automotive framework classes can be used by any application that needs to implement the functionality provided in the automotive framework. For example, if an application needs to leverage the SUI capabilities of the platform to provide voice announcements, it can instantiate the speech user interface classes.

The following are a few examples of class that the automotive framework provides:
• Display Class — the Display Class enables an application to write fixed and/or dynamic messages to the vehicle display. The Display Class abstracts multiple display types, character maps, and message maps. The Display Class supports the ability to send display-specific messages as well.
• Application Model — The Application Model is a class framework for building automotive applications. It includes a skin parser that enables separation of speech UI presentation from application logic.
• Speech User Interface Classes — the speech user interface classes provide application written in managed code with a set of speech controls to manage speech recognition and text-to-speech events. These classes abstract the underlying Microsoft Speech API (SAPI). They also provide the device with a mechanism to manage speech interactions from multiple applications via a priority-based queue.

Applications: The application layer provides user appreciable functionality; the real value of the device. The platform seeks to simplify the construction, deployment, and running of applications. To do this, the platform provides a SDK to enable Tier 1 and their 3rd party suppliers to develop managed code application.

Developers use the SDK in conjunction with Microsoft Visual Studio® to construct platform applications. A single .exe file contains an application. Each application executes in its own process space. The SDK makes available to developers class libraries that include the automotive framework classes (built specifically for The platform) and the .NET Compact
Framework. The application, the automotive framework classes, and the base class library code execute within the managed environment of the common language runtime of the Windows Automotive OS.

Developers write application logic in either Visual C#® or Visual Basic® .NET. They use Visual Studio to edit, build, test and debug the application. At build time, Visual Studio compiles the source files into IL and stores the IL in the .exe.

Developers store application resources in one or more skin files. Skin files may contain text–to–speech (TTS) string resources, grammars, speech control properties, and display elements. Developers define the contents of these skin files using the eXtensible Markup Language (XML) syntax. At runtime the automotive framework loads and parses these skin files.

Depending on application deployment requirements, developers may include one or more applications into a single application. Let me explain. The .exe file is the unit of deployment. a developer can choose to combine multiple applications as sub-applications into a single .exe. If the developer chooses to do this, then all sub-applications must be deployed together. For example: there are two applications that rely on the same XML Web services. The developer may choose to package these applications into a single .exe as sub-applications. Because each process instance has some memory overhead, one benefit that comes from this packaging is memory conservation.

A FEW SPECIFIC SOFTWARE COMPONENTS

A few major elements included in the software stack are: device management, server connectivity, speech API, and “Skinable” user interface.

Device Management: PC software is now upgradeable over an Internet connection, so software must be, too, in order to meet customer expectations. OEMs naturally have concerns about what software is allowed to be loaded onto the telematics systems they produce. The Device Management component, including Over The Air (OTA) updates, helps to provide a more secure method to download new software only from authorized parties. This saves considerable cost over requiring the vehicle owner to bring the car in to the dealer for reprogramming.

Server Connectivity: A key strategy behind the platform is allowing a lot of functionality to reside "off-board", on servers connected to the Internet and communicating wirelessly through the platform to the user. Upgrading or adding functionality at the server avoids having to “touch” each device in the vehicle fleet, simplifying rollout, reducing cost, and ensuring a consistent experience. It therefore includes support for a number of open standards for server connections (e.g. TCP/IP, HTTP, SOAP, and XML) that allow interconnection with many types of back end systems.

Speech Service: A multifunction system like our platform runs multiple applications, each of which may need to converse with the user through Speech Synthesis and Speech Recognition. To avoid cacophony, all speech goes through the Speech Service, which manages the process of holding multiple conversations with the user. The Speech Service makes certain that questions and responses come through in an orderly manner, and can interrupt when high priority messages need to get through.

“Skinable” User Interface: For systems with graphic displays, user interface elements are common. A “Skinable” user interface enables developers to easily customized apps for specific look and feel consistent with the target vehicle brand image. While these concepts apply to display based user interface; because the initial instances of the platform UI are primarily speech oriented, this discussion focuses on applications that have a speech user interface.
A skin is the part of an application that defines the speech user interaction for the application. The application's skin contains information about speech control properties, text–to–speech string resources, along with any other information that determines the speech UI presentation of an application.

Skins enable developers to separate speech behavior from the application logic. By maintaining the speech UI presentation in a separate location, developers can easily update or replace the application's speech functionality without re–structuring the underlying code. Development teams can create uniform speech definition for a group of applications by distributing the same skin file with each application.

Skin markup is the XML code that enables developers to separate speech UI presentation from application logic. Skin files are written in skin markup. Skin markup is a subset of XML specifically designed for the platform.

Skin markup has a set of rules to map classes to elements; attributes to properties as well as events; and map XML namespaces to common language runtime namespaces. For example, skin elements map directly to automotive framework classes.

Skin markup provides a convenient way to set speech control properties and group speech resources. It can also assist development teams with localization when teams use skin markup as a central repository for resources belonging to all applications.

Microsoft’s Telematics Platform: ENABLING A VIRTUOUS CYCLE FOR TELEMATICS

Microsoft believes, based on experience in PCs and consumer electronics, that telematics will not take off without enabling the virtuous cycle evident in other technologies. Briefly stated, a technological platform that is attractive to developers drives new applications, which attract new customers, which attracts more developers, and so on. The end result is a popular technology that customers want and can afford.

Because of the complexity of telematics applications and services (we are only just beginning to scratch the surface of the possibilities), successful OEMs will be the ones that provide a platform that allows third parties to innovate. No one company is large enough to provide a complete solution. The risk is not in providing a good platform that has a well defined set of features, but rather it is in not seeing the unanticipated uses of the platform. Because telematics is poised for rapid growth over the next decade due to the confluence of the relevant enabling technologies, it is also poised for rapid obsolescence cycles if the original equipment telematics systems are closed and static. Customers want the ability to keep up with the times without major reinvestment, much as they can with the other familiar technologies in their lives.

Adopting a platform like the Microsoft telematics platform, one that has been designed to be familiar and open to developers, is a first step in creating telematics systems that age gracefully. The platform addresses this problem with a familiar programming model, support for well-known standards, and “Over The Air” upgradeability. At the same time, OEMs need to maintain brand image and usability standards, so completely open systems where users can load whatever they want are not generally viable. The platform supports OEM concerns with the ability to control the user interface and the ability to set strict controls on what software is downloadable to the platform.
Starting with this framework, OEMs may also need to make changes to their approach to telematics system development. For example:

- Telematics systems must not be viewed as fixed-function devices, but rather as devices whose software and thus functionality can change over time. This runs contrary to experience with most other vehicle modules (e.g. control systems) and represents a cultural and structural obstacle for the industry.
- Development of telematics systems needs, to an extent, to be decoupled from the mainline automobile development cycle. This is because the telematics industry can change dramatically during the three to five years it takes to bring a new vehicle to market.
- Choices regarding platform performance (e.g. CPU speed and available memory) must be made in light of the need to support new functionality in the future and the high cost of upgrading hardware.
- OEMs must be careful of the roadblocks they place in the way of partners that wish to create innovative new solutions. If developing new software and services requires complex business relationships to be formed, this will have the effect of driving innovation away from the platform to other platforms.

CONCLUSION

The industry is on the cusp of a period of rapid growth and innovation in telematics. Consumers will relate well to telematics if it matches expectations developed through experience with other technologies, such as PCs. Industry players need to look beyond the straightforward question of what functionality to implement and instead must see the need for a flexible platform that enables organic growth of capabilities within certain controls. The Microsoft telematics platform is one example of an architecture that is aligned with this vision.