Learn the discipline, pursue the art, and contribute ideas at www.ArchitectureJournal.net

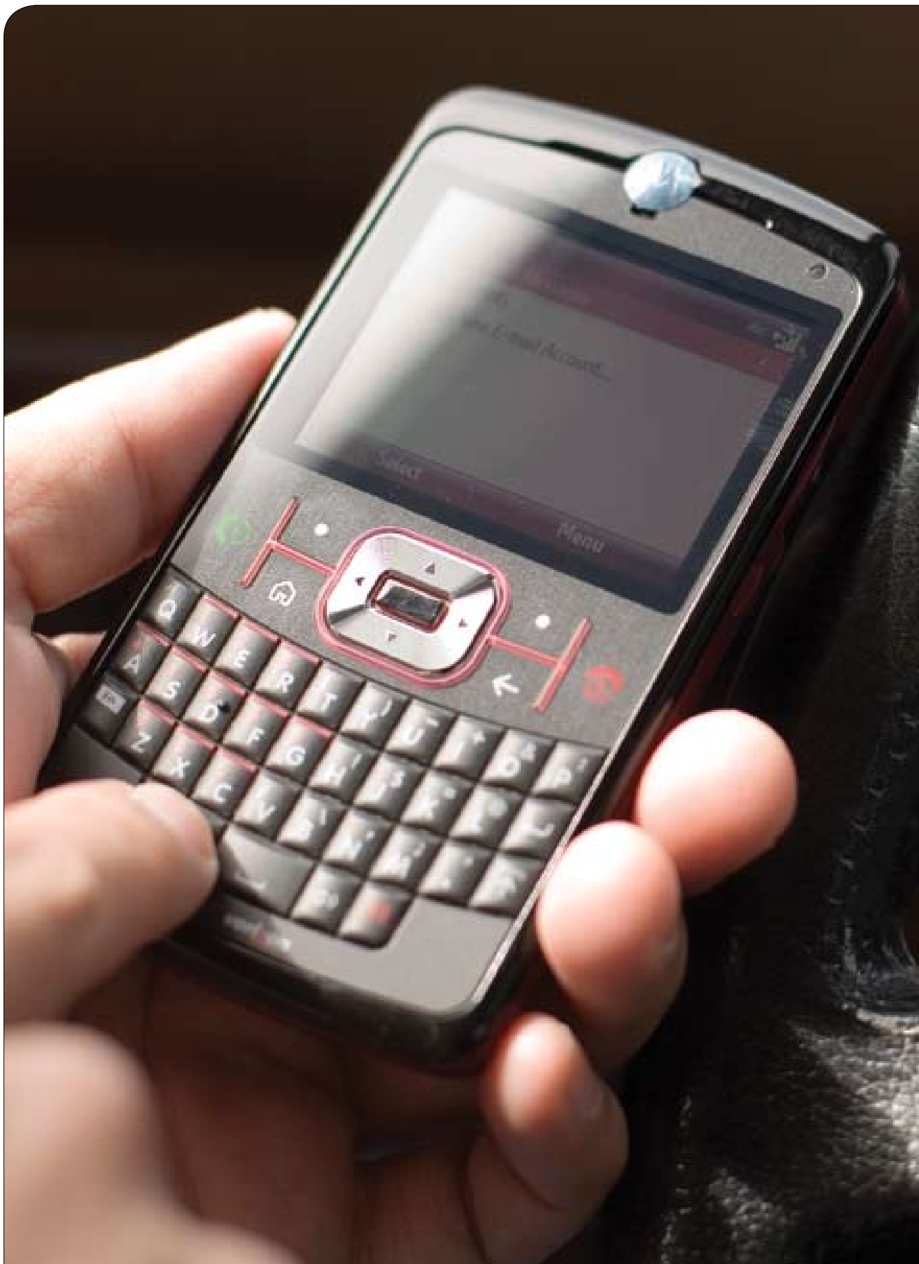
Resources you can build on.

THE ARCHITECTURE JOURNAL™

Input for Better Outcomes

Journal 14

Mobile Architecture



Microsoft®

Contents

THE ARCHITECTURE JOURNAL™
Input for Better Outcomes

Journal 14

Foreword

Dear Architect,

Every day I find the pervasiveness of mobile phones and devices just amazing, especially when I look at the projected growth rate for the next few years. With this growth and the rapid advances in mobile technology, I realize that there is a very good chance that my children will grow up never knowing what landline, rotary or pulse dialing really means! With any technology, software plays an important role in complementing this phenomenal growth of hardware, and this is the focus of this issue of *The Architecture Journal*.

To lead off this issue, Atanu Banerjee covers many considerations and aspects of applications on mobile devices today. Following this, Kulathumani Hariharan, an architect at Tata Consultancy Services, shares best practices, tips, and recommendations that may be pertinent if you are considering taking a line-of-business application to the mobile platform.

We are then joined by Christoph Schittko, Darryl Hogan, and Jon Box as they introduce us to a scenario of a connected consumer experience in automotive devices. We explore what the future of software in the automobile may look like and some of the architectural perspectives that support this. Closely related to this article, we are very pleased to have our first external architect profile in *The Architecture Journal*. Faisal Waris is an architectural consultant, working at Ford Motor Company. We ask him about some of his thoughts on architecture, especially as they relate to mobile development.

Following Faisal, Rodney Guzman of InterKnowlogy shares some of his thoughts on mobile data architecture. Rodney explores some of the data challenges with occasionally connected applications and offers some ideas and concepts to help address data conflict resolution. Taking a deeper dive into mobile development, we are then joined by Munjal Budhabhatti from ThoughtWorks, who covers the importance of test-driven development and continuous integration, common engineering practices for many organizations, and discusses how these can be implemented for mobile applications.

We wrap up this issue with a trip to Hungary with András Velvárt and Peter Smulovics to look at how Monicomp, an organization that installs, maintains, and repairs point-of-service systems is using an ultra-mobile PC application for their support technicians on the road.

That brings this issue to close. I hope that some of the articles and authors help inspire mobile application development in your organization. We'll be returning in the new year with Journal 15 on the "Role of an Architect," where we'll be taking a closer look at the people in our profession, and putting the work that we do under the microscope!

Simon

Simon Guest

Founder
Arvindra Sehmi
Microsoft Corporation

Editor-in-Chief
Simon Guest
Microsoft Corporation

Microsoft Editorial Board
Gianpaolo Carraro
John deVadoss
John Evdemon
Neil Hutson
Eugenio Pace
Javed Sikander
Philip Teale

Publisher
Lisa Slouffman
Microsoft Corporation

Design, Print, and Distribution
CMP Technology – Contract Publishing
Chris Harding, Managing Director
Angela Duarte, Publication Manager
Lisa Broscritto, Project Manager
Kellie Ferris, Director of Advertising
Jimmy Pizzo, Production Director

Architectural Considerations for a World of Devices

by Atanu Banerjee

Summary
The number of mobile device users is rapidly increasing, but the promise of solutions that leverage networks of connected devices has remained largely unrealized. Some of the pieces needed to build rich, connected experiences were not available until recently. This article explores some of the economic, social, and technology trends that are driving the adoption of mobile devices; describes the different kinds of user experiences that are now becoming possible; and presents an overview of architectural concerns associated with such mobility solutions, at the levels of hardware, software, connectivity, and services capabilities.

New Opportunities With Devices

After 10 years of hype, mobility solutions are finally taking off. You could ask, "Why now? What has changed? Are there any new opportunities to consider? Should I consider mobile devices in my solutions?" It turns out, economic, social, and technology trends are accelerating the move to devices. There is a broad spectrum of devices with different form factors, running different kinds of applications, as shown in Figure 1, and associated with different sets of trends.

Economic Trends

An important driver of adoption for cellular phones has been emerging markets. For example, *BusinessWeek* reports that Nigeria had 500,000 telephone lines in 2001, but now has more than 30 million cell phone subscribers. It is currently estimated that there will be 5 billion cell phone users in the world by 2015.

Adoption of cell phones drives adoption of services for cell phones. In Asia, many services leverage high-end devices to deliver rich, interactive media. These require higher end smartphones and pocket PCs shown in Figure 1. However, many people in emerging markets cannot afford such devices, so services that target lower end volume handsets are also being rolled out. In Kenya, Safaricom rolled out an SMS-based service for mobile payments in March 2007, called M-Pesa, that has been widely adopted. Unsurprisingly, access to improved communications can also be hugely beneficial to local economies. Dr. Robert Jensen at Brown University conducted a study of Indian fishermen who started using mobile phones to find the best coastal marketplaces for their catch. While the fishermen saw profits increase 8 percent, consumer prices actually dropped by 4 percent because less fish was being wasted.

Today in Helsinki, Finland, 57 percent of public transport single tickets are paid by mobile phone. In Croatia, over half of all parking is paid by mobile phone. Twenty percent of London's congestion charge is paid by mobile phone. (See Resources: Mobile Phones As Mass Media.)

Social Trends

There are now over twice as many mobile phones worldwide as there are personal computers. The wireless industry used the opening of its largest trade show in March to outline opportunities for a "three-screen" world (PC, TV, mobile), in which mobile devices become major avenues for TV shows, music, games, and advertising. For many younger consumers, it might even be argued that the mobile device is the most important of those three screens.

Accompanying the growth in devices, the evolution of the Internet is leading to new usage patterns. Today's solutions are differentiated from older ones by their global reach and scale, which lead to new channels for user participation. For example, sporting goods company Nike sells the Nike+, a small sensor that fits into a runner's shoes and tracks his progress on an iPod that the runner also carries. When the Apple iPod is connected to a PC, details of the runner's runs get posted to the Nike+ Web site, a social networking site for runners, so

Figure 1: Range of mobile devices, and the kinds of applications that run on them

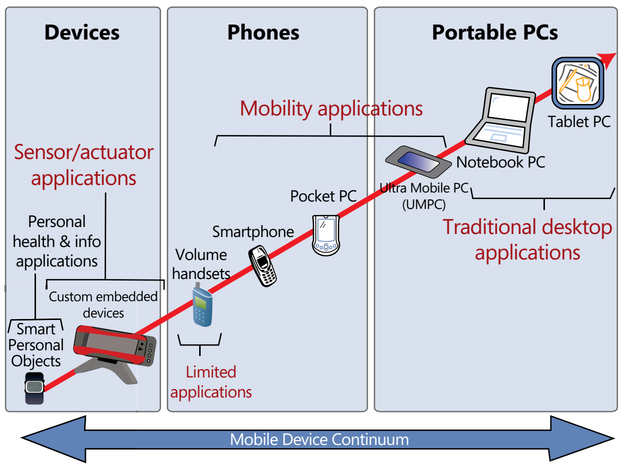
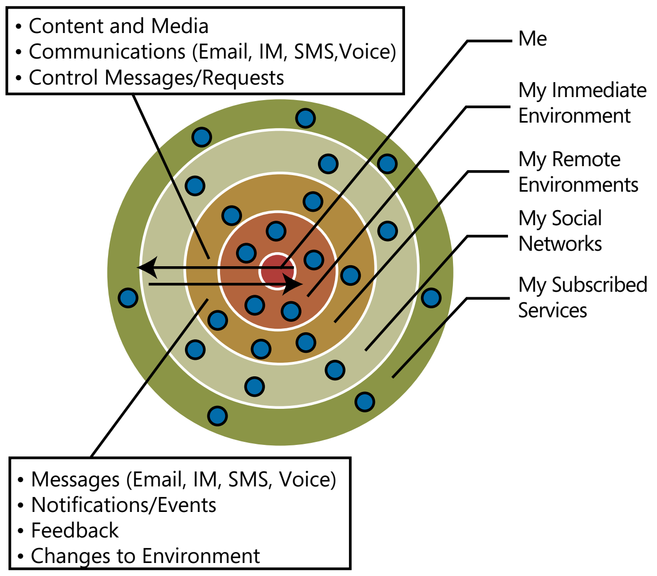


Figure 2: User experiences in a world of connected devices span multiple physical and virtual spaces. Each of the blue dots represents a physical environment (a room, the home, workspace), a social environment (friends, family, colleagues), a virtual environment (profile pages, virtual world, online game) or a subscribed online service.



that runners around the world can form groups to track each other's routes and progress.

The Internet is also changing the way that content gets created. Blogging is making content publishing less impersonal as readers are closer to authors. Content is also becoming interactive and social with online gaming, chat, and the advent of communities around user-generated content. This trend will accelerate with the proliferation of mobile devices that make it easy to capture content on a device to edit content directly on the device, attach context to the newly created content and then upload the content to storage either on a PC or in the cloud. In some P2P scenarios, it is also becoming possible to share content directly from the device itself.

This transformation of content pipelines makes it more likely that content creation will be triggered by external events rather than on fixed schedules. The handiness of devices makes it increasingly likely that a means of recording an event will be at hand when it happens, leading to the spontaneous creation of new content that might not have been captured otherwise. Not surprisingly, the amount of content being generated and stored has exploded. The upsurge in spontaneous citizen reporting has resulted, on more than one occasion, in footage from a mobile device leading to dramatic public reaction.

Technology Trends

The first trend that is reshaping the industry is that of convergence. Today people use a wide variety of devices—smartphones, PDAs, laptops, personal media players, cameras and camcorders. It is expected that these technologies will converge into more powerful, general-purpose personal computing devices that can be used for a wide variety of business and consumer-oriented tasks. Convergence in networks will mean seamless handling of both voice and data over the same protocols.

Convergence leads to the second trend: Devices are getting smarter. A new generation of smartphones is becoming increasingly aware of the user's environments and local context, through sensors (such as GPS or accelerometer) and better software on the device. This context might be used to tag content (for example, tagging a photograph with time/location metadata), to tailor application behavior (no application alerts when the user is on a phone call, for example), or to control the user's local environment (such as settings in a car self-adjusting based on the driver's identity which has been retrieved from a device on the driver's person). Networks will also extend to cover devices and agents distributed around the body over protocols such as Bluetooth — which is the idea of Personal Area Networks (PANs). All this will lead to new architectures where devices are much more than just information displays — they will become first class application platforms in their own right. Not only that, but there will likely be some scenarios (such as PANs) where some devices act as servers for other devices (in client-server architectures), or as super-peers / index servers (in P2P architectures).

This is important as mobile device applications need to provide a user experience that is very different from that of a desktop. The key characteristic of mobile users is that they are engaged in some other primary activity. A device-based application should not force its users to make accommodations, but instead fit into people's lives and lifestyles by being context-aware, nonobtrusive, and ready to provide value rapidly at short notice.

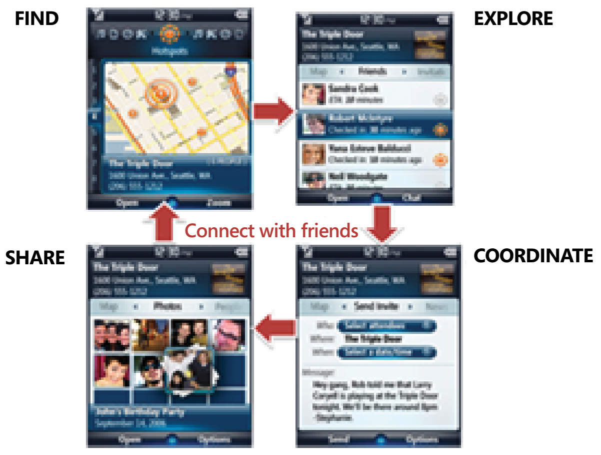
The third trend is the mobile Internet, a collection of Web sites and services specifically targeted at mobile devices and available over Internet protocols. Growth of the mobile Web will accelerate consumption of Internet-based applications and services from mobile devices, which today is constrained by device and mobile access plan limitations.

The combination of these three trends will result in a move toward pervasive computing. As devices proliferate and become smarter, more computing power will be embedded at the edges of the network. As devices become better at handling user context, they will become increasingly unobtrusive. As devices become better networked, and as the mobile Internet evolves, users will have available a rich set of services that can make use of this personal context. The borders between human environments and computing devices will gradually blur, and users will get the sense of being assisted by their immediate environments. Large numbers of embedded computing devices will force new solution architectures to handle emergent challenges around user experience, device management, security, content management, and so forth. Network access needs to become universally available. Although we are clearly not at the point of pervasive computing yet, we are moving in that direction with the growth of embedded devices and smartphones, the spread of wireless connectivity across our environments (from work places to living spaces to some cars), and the broad availability of Internet services to be accessed by devices.

What Will User Experience Look Like?

As devices become more common, software will need to span a mesh of Web-connected devices and embrace the increasing pervasiveness of the Internet, a core pattern of Web 2.0 described by Tim O'Reilly as "software above the level of a single device." Devices are used in multiple physical and virtual spaces (Figure 2, above).

Figure 3: Social networking on devices



Experiences Related to Me
The devices in these scenarios are typically used for communications (phone, email), gathering of content (mobile search), consumption of content (personal media players), and for health monitoring (heart rate monitor). The application scope in these scenarios centers on information gathered about you, created for you, or consumed directly by you. Relevant information includes credentials (Live ID), contacts, messages, presence information, and personal content (audio, video). The types of devices that are important include cell phones, smartphones, PDAs, Ultra-Mobile PCs (UMPCs), laptops, and health monitors. The connectivity needed is for Bluetooth Personal Area Networks (PAN), Health sensors, and so on.

Experiences Related to My Local Environment
As described earlier, the proliferation of smart devices will lead to spaces where the boundaries between a person's immediate environment and computing devices in that environment are blurred. This will be achieved through networked devices with built-in sensors (GPS, accelerometer, ambient light sensor), that understand user context, and are unobtrusive in their actions, so that users get the sense of being assisted by their environment.

An example of such an environment is the Microsoft Auto solution, which connects a user's devices (such as mobile phones and portable media players) into a single in-car system that can be operated with the driver's voice or buttons on the steering wheel. Ford Motor Company will roll out a solution called Ford Sync in 2008, which will enable next generation mobile user experiences: for example, users entering a car while talking on a mobile phone and can press a button on the steering wheel to have the phone connect to Sync without interrupting the call. Another case of extending an automotive environment with devices is that of OnStar, which provides security and roadside assistance: within the car, a communications device is connected to the radio, a GPS antenna, and a microphone via an on-board network (or bus).

Conference rooms are being extended with devices as well. Microsoft RoundTable is a combination video conferencing camera and microphone that uses sound and motion detection to automatically shift focus to the current speaker. Eliminating the need for speakers to move to face a fixed camera when they start to speak is in line with the idea of devices becoming less obtrusive in their actions.

In some cases, devices may need to share information with other devices also in a user's vicinity, as well as with Internet-based services, raising issues around discovery, handshaking, shared understanding of the user's identity and context, and so on.

Experiences Related to My Remote Environments
Remote environments are similar to local environments, in that they are spaces where devices gather information and take actions. However remote environments are not in the immediate vicinity of the user of the device. In other words, scenarios and experiences that relate to a users remote environments allow that person to connect to, monitor, and work with devices at other locations. Reasons to do this would be to monitor or even control the environment at remote locations as a safeguard against criminal activity or for other reasons. For example, people might be interested in remotely monitoring their homes or workplaces (such as a data center), or even loved ones (children or elderly parents). A simple example of a device that does this is a baby monitor. Businesses might want to monitor remote locations; there are a host of logistics-related scenarios using RFID devices in this category as well (for example, to ensure the electronic pedigree of pharmaceuticals as they are transported through a supply chain, in order to eliminate counterfeit drugs).

Experiences Related to My Social Networks
Figure 3 shows that mobile devices fit into social networks in the same way personal computers do. Users use their devices to search for and find people and their content, to coordinate with their friends and relatives, and to share content with others.

However, the reach and scale of devices is much broader than personal computers, and social interactions are more spontaneous (when the user is on the go, her camera phone is always at hand). Applications and services need to accommodate such scenarios, where the user's attention span is sharply reduced.

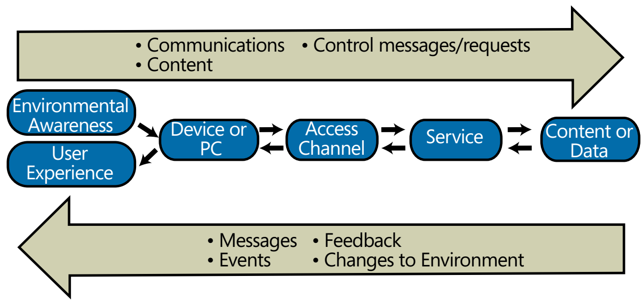
What architectures are needed to build end-to-end solutions that support devices?

Challenges in Delivering Rich Experiences on Devices
There are many differences between devices and personal computers, and it would be a mistake to consider devices as just smaller versions of PCs. In order to deliver rich experiences onto devices, solution architects need to consider a number of constraining factors, much more so than when delivering applications to a personal computer. These constraints include hardware capabilities of the devices themselves, device operating systems and application runtimes, development tools, connectivity choices, and also available services running on the Web. Some of the challenges in building experiences for devices are:

1. Unlike PCs, people consider devices to be accessories: Users carry devices on their person and often view them as expressions of their lifestyle or even personal self-image. Coolness factor, great design, and user experience are critical.
   Implications: Rich device design and presentation capabilities are needed—both for the device itself as well as for the applications running on the device.

2. Devices have limited resources. As devices (and their batteries) become smaller and lighter, screen sizes and layouts become more restrictive, and the "power budget" to support rich applications shrinks. More complex devices tend to have shorter battery lives than simple bare-bones phones. Memory available on a device is limited as well, although this is mitigated by improvements in storage technology. Devices support for Wi-Fi often comes at the price of shorter battery life as well.
   Implications: Energy efficiency is critical. Device operating systems should have fine-grained control over hardware utilization. In some cases, application processing should be offloaded from the device to avoid excessive resource consumption. This leads to a tradeoff with the first two items in this list.
3. Devices are not standardized. Unlike PCs, the form factors and hardware/software profiles of devices are much less standardized.
   Implications: A developer of applications for mobile devices needs to target lowest common denominator for screen size, shape, and orientation in order to deploy to a wide variety of handsets. This leads to an implicit tradeoff with the need for rich user experience; solution architects will have to optimize both software and hardware together in order to get the best overall experience. Application developers for embedded devices have a different set of challenges in this area – lack of standardization in hardware make it hard to presume the set of resources that will be available to the application.
4. Devices need to support offline scenarios and occasionally slow connections. For many reasons, devices are not always connected (connections may not be cost-effective during international travel, for example) or access speeds are not fast enough to support decision-making at the point of need (using online maps to make routing decisions while traveling).
   Implications: Devices need to be more than thin client displays; they need to be application platforms in their own right. A key requirement for this capability is local processing and storage, with synchronization with PCs or Internet-services.
5. Connectivity is not standardized. Although several standards exist for network protocols, there are several ways for a user to access information and services on the Internet from their mobile devices. Depending upon the capabilities of their device, and the service plan that they have with their network operator, a user might want to access information and services on the mobile internet via voice (through voice recognition), messaging (SMS, email), or through Internet protocols (WiFi, tethered connection, or appropriate data plan). In emerging markets services will probably need to be delivered over SMS or voice, as users are more likely to have volume handsets with limited capabilities. For rich media experiences, it is likely that a fast Internet protocol will be required.
   Implications: Tailor access to the kind of experience being delivered and the market to which it is being delivered. It is possible that services will need to be accessible to devices from a number of different end points, each supporting a different address and method of access.

Figure 4: A conceptual framework to apply Software + Services to a mesh of devices



Applying a Software + Services Approach to Devices

The last issue of *The Architecture Journal* covered the emerging paradigm of Software + Services. In the context of a mobile application, the goal is to combine the best of the Web with the best aspects of devices—but subject to the constraints just described. As shown in Figure 4, solution architects need to design for a specific type of user experience, and pick an appropriate device based on minimum device capabilities needed (hardware and software on the device) and connectivity options and services available to users of the device.

Hardware Form Factors of the Device

Just as there is a wide range of mobility-based scenarios, there is a wide range of device form factors to support these scenarios as shown in Figure 1. The choice of device depends on how it is going to be used.

For example, in the consumer personal device space, the spectrum of available hardware ranges broadly from WM (Windows Mobile)-based devices on one end, laptops on the other, and UMPC devices in the middle. WM devices are usually either smartphones or pocket PCs (typically under 5-inch screen size), and UMPCs are usually portable digital companions (typically 5.6- to 7-inch screen sizes).

Sensors to Receive Inputs From the Immediate Environment

Devices can receive inputs from a variety of different sensor elements listed in this section. Some of these sensors will provide channels for users to interact with the software running on the device. Other sensors on the device will provide applications with a view of the user's current context at any time so that software can adjust itself accordingly.

1. *Touch Technologies.* Some devices use touch technologies to improve user experience. Newer devices use "capacitive touch," which does not require pressure to register touch (unlike the "resistive touch" found on older devices, which often required a stylus). Devices with "capacitive touch" are easier to use, more accurate, and more responsive. The older touch screens were often not very clear in sunlight, which made it harder to see rich media, but the newer touch screens are typically brighter, as their surface isn't covered with the thin film required for "resistive touch." Another advance is multitouch (the ability to handle input from more than one finger at the same time), which lets users resize a window by pinching or expanding two fingers on the screen.
   A common user objection to touch screens is their lack of tactile response. However some handset manufacturers are adding the tactile-feedback technology found in game controllers (for example, to give a slight vibration when a touch screen's virtual keyboard is tapped). This will be similar to the response that users are accustomed to getting from traditional mechanical keyboards.
2. *GPS*. Many mobility solutions depend upon knowledge of the user's location. A common technique for a device to determine its location

is GPS (or Global Positioning System) which does so based on line of sight from three or more satellites (which means that GPS cannot be used indoors). Some location-based services on the Web will automatically use GPS information on the device (when available) to provide information filtered by the users location (Live Search).

3. *Accelerometer.* Some devices have accelerometers built in. A basic usage of this is to automatically detect the orientation of the device, that is, landscape or portrait mode. More advanced uses of the accelerometer include gesture recognition, media control, or game control. In the future, it is quite conceivable that accelerometers on devices could lead to sophisticated control scenarios similar to those of the remote control of the Nintendo Wii, which is built around an accelerometer.
4. *Health monitoring sensors.* Examples are sensors for heart rate or blood sugars.
5. *RFID.* RFID readers, scanners, and printers are a range of devices that use RFID technologies in primarily enterprise scenarios such as logistics and supply chain management. Devices with built-in RFID sensors are meant to replace an older generation of devices that use bar code scanning.
6. *Other Sensors.* Other sensors found on devices might include ambient light sensors (to control screen brightness and preserve battery life) or proximity sensors (to turn off display when the device is being used as a phone).

Software Running On the Device

Software running on the device falls into the following categories:

1. Device Operating System
2. Application Platform—both the application runtime and design tools
3. Mobile Browser—this is emerging as an application platform in its own right for consumer devices
4. Applications.

The operating system should be chosen based on how the device is to be used, as there is an implicit tradeoff between managing limited device resources and the richness of applications running on the device. Devices have different needs; let us look at the software stacks for each type of device represented in Figure 1.

Software Stack for Embedded Devices. Windows Embedded CE is a hard, real-time, 32-bit, memory-protected operating system kernel that can support a wide range of processor architectures (ARM, MIPS, x86, or SH4). It comes as a set of about 700 components, from which a subset can be packaged into custom images. For example a kernel only image can be assembled that boots with an approximately 300 KB footprint, but it is also possible to add other technologies into the image—such as Web server, browser, media player, networking support, .Net Compact Framework—all of which increase the size of the OS image. Devices built with Windows Embedded CE might be headless, or might have some form of display. Also devices can either be open (that is, exposing application APIs) or closed (without a third-party developer story).

Windows CE is available to the general embedded system development community to build their own devices. It is also used within Microsoft to build the Windows Mobile and Microsoft Auto solutions. Windows Mobile is used to power smartphones and PDAs, while Microsoft Auto is a platform for the auto industry to build advanced in-vehicle solutions.

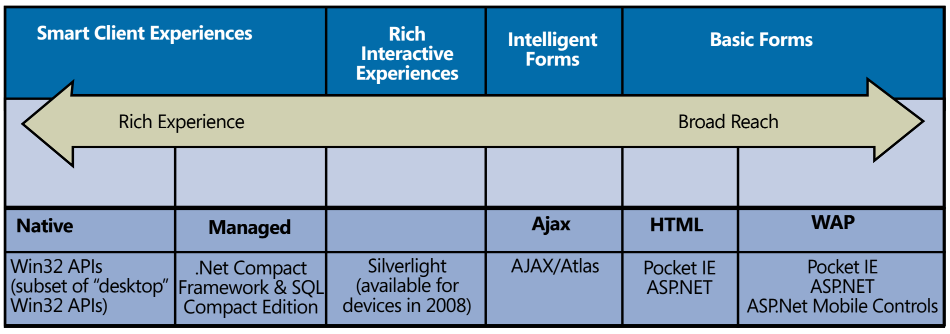
Software Stack for Smartphones and PDAs. Windows Mobile chooses its own set of operating system components from Windows Embedded CE, with a custom shell, device-specific technologies (connection manager), and some applications (Office Mobile). Windows Mobile OEMs often add their own specific applications and services to the image (screen plug-ins, applications like VoIP, games), but do not customize the set of components in the base WM image. The result is a consistent set of APIs that are offered across all Windows Mobile devices: In theory, applications written for one Windows Mobile device should work across all Windows Mobile devices. In reality, mobile devices vary greatly in their hardware capabilities (connectivity options, screen size, resolution, orientation), making it difficult to build an application that works well across all devices, even when the underlying APIs are the same. Figure 5 shows the range of development options for building application interfaces on a Windows Mobile smartphone.

Software Stack for Ultra-Mobile PCs. UMPC devices get the full fidelity software stack—the Windows Vista operating system, the .Net Framework as the runtime for managed applications, and IE7 as the browser. Existing PC applications do not need to be rewritten to run on an UMPC—although they might be extended to support touch and ink (these capabilities are now built right into Windows Vista). However, all this comes at the cost of battery life (often just 4-6 hours), as UMPCs do not manage device resources at the granular level the way Windows Mobile does.

Access Channels to Support Devices

Mobile devices such as cell phones are used primarily for communications—mostly voice calling today—but also some other forms of messaging, such as email, instant messaging (IM), and SMS. Beyond these basic communication services, it is expected that devices will connect to a much richer set of application services in the future. While there is a relatively large market for such services, it cannot be assumed

Figure 5: Development options for building application interfaces on a Windows Mobile smartphone

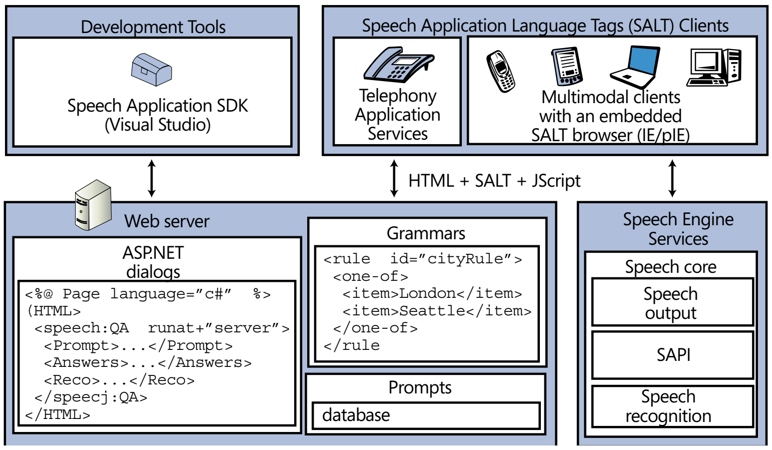

Smart Client Experiences
Rich Interactive Experiences
Intelligent Forms
Basic Forms
Rich Experience
Broad Reach
Native
Managed
Ajax
HTML
WAP
Win32 APIs (subset of "desktop" Win32 APIs)
.Net Compact Framework & SQL Compact Edition
Silverlight (available for devices in 2008)
AJAX/Atlas
Pocket IE ASP.NET
Pocket IE ASP.NET ASP.Net Mobile Controls

that users of those services will always have advanced Internet-capable devices that either have a carrier data plan or are WiFi-enabled. This leads to service delivery to devices over other channels as well (such as SMS or voice, as in the previously discussed SMS-based services for mobile payments in Kenya).

Some examples of network channels over which applications and services may be delivered are:

1. Voice
   a. Voice Recognition: These services are accessed through a phone call, with voice recognition software running on the other end. Figure 6 shows how Microsoft Office Communications Server can be used to deliver speech-enabled applications that can either be accessed through telephony application services, or through alternate channels.
   As an example of such a mobile application, consider Live Search on Windows Mobile devices, which can be accessed through a voice interface in the U.S. Voice recognition software converts the users speech into the search query string; results are then displayed as usual.
2. Messaging
   a. SMS: Some basic services are now being offered over SMS, such as stock updates, alerts, and, in some countries now, Internet search. For example, Microsoft Research did a project in India, where they built an SMS-enabled solution for a sugarcane cooperative. Farmers can use their phones to get information (such as market price information) by sending in requests as SMS messages. The responses are also sent back to them through SMS. Microsoft Research has made the toolkit used for this project available as a shared solution on CodePlex. (See Resources: SMS Server Toolkit.)
3. Internet
   a. WiFi: This works well for devices equipped for WiFi, when in the vicinity of an accessible wireless hotspot.
   b. Mobile data plan: This typically is a premium service offered by mobile network operators, to provide Internet access over the operator's own cellular network.
4. Connection to other devices using P2P (Peer-to-Peer) technologies
   a. Some devices can exchange information with another device directly, without having that information pass through a central server. Architectural considerations, such as discovery and handshaking, can be accomplished in two ways:
      i. A central index server brokers the connection: For example consider the XBox 360 and the XBox Live. When a user logs into his XBox 360, he can join a group of up to 16 gamers that play within a single session. Although the Live service tracks who is online and brokers the initial connection into the group, all further messaging between XBox 360 consoles is direct through P2P technologies and not through the server.
      ii. Without a central index server: Discovery of nearby devices, and the handshaking between them, happens directly, without going through a central server. For example, the Zune music player can share currently playing music with up to three other Zunes in the vicinity via P2P technologies.

Figure 6: Using Speech technologies in Microsoft Office Communications Server to deliver voice enabled services over the internet


Development Tools
Speech Application SDK (Visual Studio)
Speech Application Language Tags (SALT) Clients
Telephony Application Services
Multimodal clients with an embedded SALT browser (IE/pIE)
HTML + SALT + JScript
Web server
ASP.NET dialogs
Grammars
Prompts
database
Speech Engine Services
Speech core
Speech output
SAPI
Speech recognition

```
<%@ Page language="c#"  %>
(HTML>
 <speech:QA  runat+"server">
  <Prompt>...</Prompt>
  <Answers>...</Answers>
  <Reco>...</Reco>
 </speecj:QA>
</HTML>
```

```
<rule  id="cityRule">
 <one-of>
  <item>London</item>
  <item>Seattle</item>
 </one-of>
</rule
```

Services to Support Devices

Beyond basic communication services, it is expected that in the future, devices will connect to a rich set of services on the Internet. These services will likely be architected in three tiers:

1. Application and solution services. These offer support that is specific to a set of scenarios, such as health or CRM.
2. Attached or third-party services. These services offered by other providers are attached onto the application services; for example, a mobility solution for healthcare providers using services for email, update, or collaboration provided by other parties.
3. Utility/infrastructure/building block services.

Services in the second and third tiers represent common, horizontal capabilities which cut across many different application services. Some of these are described below in the context of mobility solutions. In the future, these services would typically be provided by a platform provider, such as Windows Live Platform, or even by a mobile network operator.

Device Management and Security

Attack vectors for devices are similar to that of networked personal computers, except that devices are much more likely to get lost or stolen, and it is often harder to secure them physically given their mobility (as opposed to a computer sitting on a desk or in a data center). So there are three primary areas in which to consider security issues on devices. The first is around securing the device itself. The second is around securing the network—that is, ensuring message confidentiality and integrity. A number of security issues here can be addressed in layers of the networking stack (for example, radio modulation techniques to provide wireless signal transmission security, IPSec, and so forth). The third area of security is securing the applications that run on the device—or run on the Web but are accessed through devices—and that is described in the section on identity and access management.

In some cases management of devices is like that of personal computers. For example, devices need to be upgraded with patches (firmware and software), media, or applications. Communications to devices need to be secured, and in some cases metered, and paid for

(for commercial downloads). However, management of devices does differ in some crucial respects because devices are often much harder to secure: Mobile devices are easy to misplace, and in many cases, access to the devices cannot be restricted.

Management services need to be provided to devices that connect into a network, to answer the following kinds of questions.

- Network administrators: "How many devices are on my network right now? How much bandwidth? How much time? What types of devices are there?"
- Helpdesk: "What is the history for your device? Has your device been updated? What are the details of your device?"
- Security administrators: "What devices don't have my security update? What if I enforce this policy? How many devices are in compliance?"
- User: "I just lost my device and I'd like to safeguard private information on it. Can you wipe it remotely?"

Note that simply locking down a device isn't enough if it has a 2 GB storage card filled with sensitive information when it gets lost. One way that Windows Mobile 6 handles this problem, is by encrypting storage card data so that it can only be read by the device that wrote to it.

Identity and Access Management

Applications running on different networked devices need a way to share credentials among themselves, as well as with back-end services that they connect to. As scenarios for devices become more sophisticated, this will require universally recognized credentials (for example, the identity of the user, and in some scenarios, the identity of the originating device as well). Today a cellular phone is identified by a phone number. Although smartphones do allow the user to connect to back-end services online, those services typically require the user to authenticate himself in multiple additional ways—which have nothing to do with the phone number, such as email address or other Web-based credentials. As devices proliferate in the future, as will the services to support them, a single universal identifier (conceptually similar to Live ID today, perhaps) might solve the authentication problem. However, other new complications will arise—how does identity get chained across devices and Web-based services? Where do boundaries of trust get established?

Although much has been done to secure networks and devices, a different set of technologies are needed to broker trust among applications running on those devices (and services that they connect to)—technologies for federated identity. These technologies help the user manage multiple digital identities and control how much personal information is shared with other devices and services. Each of these identities would be built around a set of claims which are expressions of trust from a certifying party — one or more identity services in the cloud acting as brokers of trust, issuing claims (or expressions of trust) embedded in security tokens.

Rendezvous and Presence

Mobile presence services make users' mobile context available to their social networks. Context data, such as location, device idle time, device profile (ring volume, vibrate), and calendar information, would be available either from other mobile devices, or through a gadget/widget/badge embedded on the user's blog or Web page. It might be displayed as a list (by augmenting a contacts list), or might be displayed on a map. In short, mobile presence should let a user's social networks know when they are reachable and when they are not, and what their preferred mode of communication is at that moment. This information should not be specific to a user's mobile carrier, as it is unlikely that all members of a user's social networks are customers of the same mobile carrier. Ideally, this presence information would connect into a unified communications backbone that combines different forms of messaging.

Location-Based Services

Unlike personal computers, in the future it is likely that most consumer devices will know their exact location, possibly through a position determination technology such as GPS. This opens up the possibility for a wide variety of cloud services which can make use of that information to present to the user content appropriate for that location. The growth of online Geographic Information Systems (GIS) with published interfaces for storing data and associated spatial metadata is enabling this trend. Some of these systems are also marked up with extra user-generated content tagged by location. A location-based service would make use of the user's geographic coordinates as an index or a filter into a GIS, to retrieve the right information. Such services might include local search, navigation, emergency services, tracking of children/pets/objects of value, multi-player mobile games, locating people in your social networks, logistics/transportation management, and so on.

Mobile Search and Advertising Services

In some ways, Web search from a mobile device is not very different from Web search from a personal computer. An analysis of Google search logs presented in "Deciphering Trends In Mobile Search" shows that despite the limitations in input techniques, the average number of words in a search query did not change much across mobile phones, PDAs, and personal computers. (See Resources: Deciphering Trends in Mobile Search.)

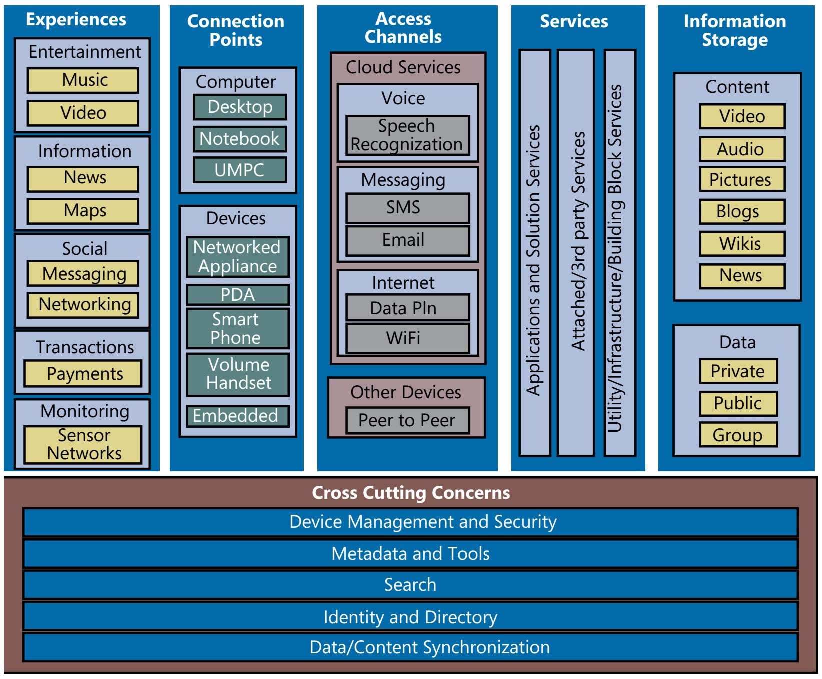
However, in other ways mobile search has the potential to be more dynamic than search from a personal computer. Devices are in the position to know more about the user's current context (location). Search engines today process queries against an index that has been built up by crawling the Web. The potential for mobile search is that the extra context available to the device could also be used by the search engine when formulating its response. For example, search results might be filtered by the current location, or list of sponsored links might include a mobile coupon to a local business.

As an example of this, Live Search for Windows Mobile now includes voice input (beta), gas prices, and hours of operation for businesses. The service can also use GPS data on GPS-enabled phones to provide location-aware local search.

Storage, Content Delivery and Content Management Services

As mentioned in the section on social trends, the proliferation of devices is leading to an explosion in the amount of content that is being generated and that needs to be stored off the device. This leads to the need for storage services to back up devices, for content delivery networks to move the data, and also for content

Figure 7: A summary view of the development options for building mobility solutions



management services to organize newly acquired content so that it becomes discoverable.

Organization of content implies a taxonomy—which might be explicitly defined, but is more likely to be emergent based on tagging of content with a snapshot of the context of the user at the time that the content was created. This context might be location, time, an event, and so forth—anything that the device was aware of and automatically recorded.

Alerting Services

As the amount of content available online keeps increasing, there is a need for users to filter the information signals that they receive. One way that users can do this is by subscribing to a particular alerting service like Windows Live Alerts. These alerts can be received on mobile devices as SMS messages. A user can also embed a gadget for reading alerts on their own Web sites.

Synchronization Services

As users move to a world of devices, content is increasingly being left scattered across personal computers at home, at work, in online services, and now on mobile phones. An important part of end-to-end mobility solutions will include synchronization services, which solve the problem of synchronizing any content, over any protocol, and onto any device or personal computer. These synchronization services would need to be able to handle subtle issues around scenarios involving caching, offline usage, sharing, and roaming. One way to build such services is to use the Microsoft Sync Framework, which lets application developers easily add synchronization capabilities to an application or service. This is enabled through a provider model that can be extended to support common scenarios such as syncing relational databases, file systems, lists, devices, PIM, music, video, and so on.

Summary

A summary view of all the development options available to an architect of mobility solutions is shown in Figure 7, with examples of specific experiences, connection points, and access channels. There are several cross-cutting concerns for devices, services, and access technologies, such as how to manage identity and trust across all these different layers.

The approach for an architect of mobility solutions should be to balance the need to achieve broad reach and scale for his target audience against the need to deliver rich experiences that users can connect with. Ideal solutions will combine the best of the Web with the best aspects of devices.

Resources

Deciphering Trends In Mobile Search
http://www.maryamkamvar.com/publications/KamvarBalujaComputerMagazine.pdf

"Mobile Phones As Mass Media: Models For Content Distribution," by Alan Moore
http://www.masternewmedia.org/media/mobile-phones/mobile-phones-as-mass-media-white-paper-part-2-20070711.htm

"Safaricom: On a Tear in Africa," by Jack Ewing, August 27, 2007, *BusinessWeek*
http://www.businessweek.com/globalbiz/content/aug2007/gb20070827_543072.htm

SMS Server Toolkit
http://research.microsoft.com/research/downloads/details/9190f48f-6e3d-4ee8-b4a9-b346db76be1d/details.aspx

"Upwardly Mobile in Africa," by Jack Ewing, September 13, 2007, *BusinessWeek*
http://www.businessweek.com/globalbiz/content/sep2007/gb20070913_705733.htm

About the Author

Atanu Banerjee is a member of the Platform Architecture Team at Microsoft, where he works on architecture for next generation solutions. He joined Microsoft from i2 Technologies, where he worked in various roles for more than seven years, including chief architect for a supply chain management product line, development manager, product architect, team lead, and software developer. During that time, he wrote a lot of code, designed new solutions, and worked with some large manufacturing customers. Prior to i2, Atanu worked in the advanced control systems group at Aspen Technologies, designing and implementing model predictive control systems for the process industry. Atanu received his Ph.D. from Georgia Tech in 1996. He resides in Redmond, Wash. with his wife and six-year-old son.

Best Practices: Extending Enterprise Applications to Mobile Devices

by Kulathumani Hariharan

Summary
Extending enterprise applications to mobile devices is increasingly becoming a priority for organizations optimizing their workforce. To achieve the desired result of a robust, scalable, secure, and responsive mobile solution with multiple device platform support, many components need to work together. The challenge is to seamlessly extend various flavors of enterprise applications, many based on a variety of technologies and platforms, on to mobile devices. This article outlines the components required to extend a generic enterprise application on to mobile devices, covers some best practices and recommendations, and describes a case study based on a real-world implementation.

Mobile Solution Overview

When extending enterprise applications to mobile devices, many solutions require a three-tier approach: the enterprise application itself, mobile middleware, and the mobile client application.

Enterprise Application. There are, of course, many flavors of enterprise applications that can be extended on to mobile devices, such as Customer Relation Management (CRM), Enterprise Resource Planning (ERP), and Business Intelligence (BI).

Mobile Middleware. As most enterprise applications don't have a direct way of working with devices, mobile middleware (as it will be called in this article) plays a crucial role. Some of the important features of this tier include security, data synchronization, device management, and the necessary support for multiple devices.

Mobile Client Application. The mobile client application is, of course, the software that will run on the device. There are many considerations at this tier, including data availability, communication with middleware, local resource utilization, and local data storage. In addition, many business factors need to be considered. For example, who are the target users? How critical is it to have the latest data? Are there restrictions for storing data on the device? What provisions are there in case of no network connectivity?

When selecting the platform for the device, we see three main options:

- *Online Applications (also known as a thin client).* This is client software, normally a browser, used when connectivity can be guaranteed. Without a connection, the mobile application does not work.
- *Offline Applications (also known as a thick client).* This is client software installed locally to the device that holds all required data for the duration of most operations, and synchronizes at the end of each day or a preconfigured period of time.
- *Occasionally Connected Applications (also known as a smart client).* This is client software installed locally, similar to the offline model, but where the application can update and refresh data at any point in time. The frequency of the data refresh depends on the criticality of the application.

Using the above three tiers as reference, let's now explore what this means for a product-based Sales Force Automation / Field Force Automation (SFA/FFA).

Extending a Product-based SFA/FFA Application on to a Mobile Device

A product-based SFA/FFA is typically part of a CRM or ERP application. It's common that this type of application does not have an existing solution for mobile devices. The application's server-side front end is typically a Web-based or a rich client application, supported by a relational database with a large data store catering to the whole organization. There tend to be certain restrictions on access to this database, including the following challenges:

- Changes to schema to support mobile extension — there is often little that can be done (or should be done) to change the schema to support mobile applications.
- Data access directly from the database, and update into the tables from the mobile device — often there are several layers of communication to go through and it is not possible to access the database directly from the device.
- Understanding the schema of the data store — schemas for these types of applications are designed to be extended, and as a result can be large and unwieldy.
- Designing a staging area with a schema structure similar to the back end for data to flow to the mobile devices — creating a replica environment for development and staging can be a challenge.

Solution Introduction

When extending a product-based SFA/FFA application on to mobile devices, the challenges mentioned in the previous section need to be effectively addressed. The architecture needs to consider components that work in tandem to address these challenges.

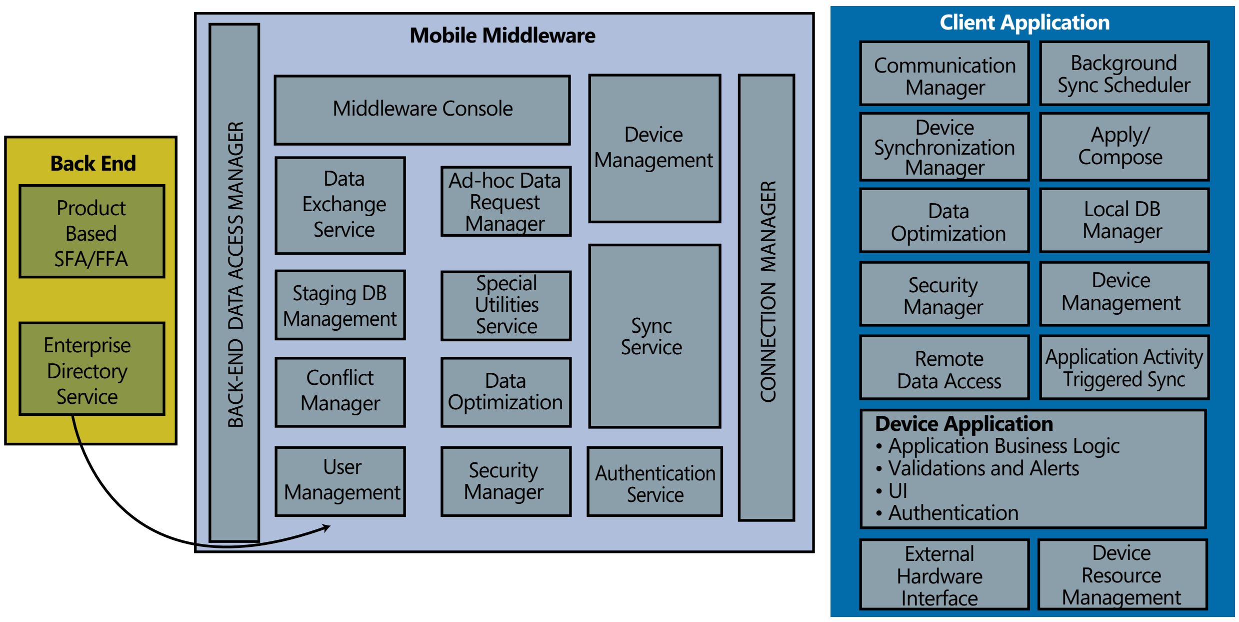
Figure 1 shows a proposed model for the smart client application; Table 1 lists the components for both the middleware and client application with a brief description of its role as part of the solution. Note that the component list is an optional superset and specific implementations may not have all the components.

Best Practices

From experience, we have found the following to be best practice when creating applications based on the model outlined by Figure 1 and Table 1.

1. Use Database Stored Procedures to write wrapper code for faster data access.
2. For ad-hoc data, the data should be populated using database views for faster output to the device.
3. The staging database infrastructure could be part of the main database server for faster response to mobile devices (the benefit is dependent on the number of users and the server load at any point of time).
4. While extending data from the back end to the staging database, include only those columns and fields that are necessary on the mobile device as the same is to be extended on to the device. This will help in adhering to size constraints on the device.
5. The staging database should only have data for a limited period (two months, for example) with regular scheduled archives; constraining the size of the database will reduce seek time.
6. Use the record version number to easily track records for delta updates during synchronization.
7. Use mapping tables in the staging database to track record version to facilitate conflict resolution; for example, to impose a conflict rule, overriding a transaction record with a server-side change even when multiple changes are done on the client end. (A mapping table is a table in the staging database which contains the primary key of the back-end database table and the primary key of the record on the device database.)
8. The Data Exchange Service should be a recurring process and should be configurable in the middleware console to handle continuous changes on the back-end system and staging database (triggered from client), creating an asynchronous method of working.
9. Maintain only necessary user details on the middleware and link to the enterprise directory service for authentication and other user data. This will reduce out-of-sync issues for user information between the enterprise directory and the middleware.
10. Do not store passwords in the staging database; instead, query the enterprise directory service during authentication. This eliminates out-of-sync issues caused due to non-update of the server-side password in the middleware.
11. During synchronization, the client application should first check for application updates by sending its current version and downloading the latest version if applicable; this is an optimized mechanism for application version management.
12. Store the user device profile (device platforms and OS versions) in the user database and push version updates to the device accordingly, sending different builds to different users.
13. Maintain three tables: in-queue, out-queue, and user-wise out-queue for synchronization management, simplifying queue management and optimizing the synchronization process.
14. The communication manager can be made to try alternative types of connectivity when the primary method is not available, so as to use the most efficient available network connectivity option. For example, when wireless LAN is not available, the application tries General Packet Radio Service (GPRS) network; if GPRS is not available, the client does not synchronize.
15. The background sync interval should take into consideration the number of users and the number of concurrent users the server can support. These considerations will assist in reducing the load on the server supporting the maximum number of mobile users.

Figure 1: Components for extending a product-based SFA/FFA on to mobile device—Smart Client Approach



16. The Device Synchronization Manager just needs to send the username, device application version, sync interval time, and the delta updates during synchronization. To reduce the number of concurrent synchronizations, the middleware should return whether an update is available and the next time for synchronization if no update is required.
17. The record state column should be maintained at record level for faster composing of data during synchronization.
18. The applications should be designed in such a way that when the battery power is low, background thread priority should be set to low, reducing CPU usage and extending battery life.
19. Develop the emulator (if not readily available) for the device type. This reduces efforts during development and testing phases. For example, use Microsoft platform builder to develop a Win CE emulator that can support features required by the application.
20. During application development, also develop simulation

Table 1: Middleware and Client Application Components

Middleware

Component	Description
Back-end Data Access Manager	• Consists of wrapper code for calling the back-end APIs to insert, update, and delete data from the back end.
Ad-hoc Data Request Manager	• Preconfigured methods that return ad-hoc real-time data to the mobile device.
Staging Database Management	• Manages data in the staging database • Stores the replica of all transaction tables with mobile users' specific columns and data. • Handles data archiving • Handles in-queue and out-queue • Cleans up database space.
Conflict Manager	• Manages data conflicts while synchronizing with the back-end database • Monitors for conflicts like Server Wins vs. Client Wins, sharing of records by multiple users, updating a record on device when the record has been deleted on the back end and so on.
Data Exchange Service	• Composes changes from the back end for a particular user to be sent to the mobile device • Applies changes from the mobile device to the back end using the Back-end Data Access Manager • Runs as a recursive service, running regularly after a period of time (configurable) • Sends composed data to the out-queue • Picks the data from the in-queue for applying to the back end.
Middleware Console	• The user interface for configuring middleware • Used to configure modules such as user management, data subsetting, synchronization management, device management, and so on.
Special Utilities Service	• Contains business logic to do specific activities that are not a core part of the middleware, but part of the mobile solution—for example, a location-based service that gets location updates captured from the device and uses a geographic information system to map latitude and longitude and display as reports • Optional, driven by business requirements.
User Management	• Manages the mobile users using the mobile devices • User list can be linked with the enterprise directory services for using the same authentication on mobile devices.
Device Management	• Manages the devices from the server • Sends new application updates to the mobile devices • Views application logs • Explores device-related issues • Manages enforcing enterprise security policies like erasing of data on devices that have not synchronized for certain number of days.
Data Optimization	• Optimizes the way data is sent to the mobile device • Compresses data and chooses the best method for sending data based on the connection speed or type of connectivity • Uncompresses data coming from the mobile device.
Security Manager	• Encrypts data being sent to the mobile device • Decrypts the data coming from the mobile device.
Synchronization Service	• Core component of the middleware • Incoming data from the mobile device is received into the in-queue and the outgoing data is pushed to the mobile device from out-queue • Data exchange service composes changes for a particular user into the out-queue and applies the in-queue changes from the mobile device onto the back end • Background synchronization from the device; the service maintains a user-wise queue and checks for new records to be sent to the mobile device.
Authentication Service	• Authenticates the mobile user during login process and synchronization.
Connection Manager	• Handles multiple connections at the same time from mobile users up to a maximum number of concurrent users.

application(s) to test the input data and output data through peripherals attached to the device.
21. Dummy data obtained from the device manufacturer or created using device manuals will be crucial for simulation application.
22. When designing the modules for the peripherals, place hardware-specific and application-specific functions in their respective libraries so that changes to the peripherals can be made without affecting the application library.
23. Where the application consists of multiple screens having common UI parts and functionality, design a base form that contains the common elements.
24. Use frames instead of multiple forms wherever possible for faster user interface response.
25. Messages (such as error messages and alerts) should be configured in the middleware and should flow to the devices. Other configuration files should also be configurable on middleware and

Client Application *(cont'd)*

Component	Description
Communication Manager	• Establishes connection to the network.
Device Synchronization Manager	• Authenticates with the authentication service • Sends and receives data from the synchronization service in the middleware • Downloads application updates and device management commands.
Data Optimization	• Optimizes the way data is sent to the middleware • Compresses data and chooses the best method for sending based on the connection speed and type of connectivity • Uncompresses incoming data from the middleware.
Security Manager	• Encrypts data being sent to the middleware • Decrypts the data coming from the middleware.
Apply/Compose	• Applies the incoming data • Composes changes to be sent to the back end.
Scheduler for Background Synchronization	• Configurable component schedules background synchronization from mobile device • Sends data to the server at a pre-configured interval without user intervention (automatically).
Remote Data Access	• Calls the methods defined in the 'Ad-hoc Data Request Manager' to have real-time data on the mobile device; if connectivity is not available then data existing in the device is used.
Local Database Manager	• Manages data in the device database • Applies and composes data • Cleans up temporary data from the database • Manages record state • Manages device configuration details.
Device Management	• Executes commands from middleware • Applies application updates • Locks out application if the user enters wrong password for a specific number of times • Send logs to middleware.
Application Activity Triggered Synch	• Triggers synchronization on completion of a complete business flow, ensuring that the mobile client back end are in sync.
Application Business Logic	• Lynchpin of the whole mobile solution (actual application used by the mobile users).
Validations and Alerts	• Validates the user input with some business rules • In case of non-compliance, displays alerts on the user interface accordingly.
External Hardware Interface	• Interacts with external hardware interfaces attached with the device • Necessary for data flow to external applications, such as Barcode Scanner.
Device Resource Management	• Accesses the resource state and displays alerts if user attention is required.
User Interface	• Face of the mobile solution. • Information entered by the user through the UI is validated and processed by the business logic.
Authentication	• Authenticates user with the middleware if there is network connectivity, otherwise authenticates with credentials available locally.

then pushed to the device application for use.

26. Database-specific queries should not be hard-coded; instead, the queries should be fetched from the middleware via a configuration file.

Technical Case Study: Extending a CRM Application on to Mobile Devices for a Territory Management System

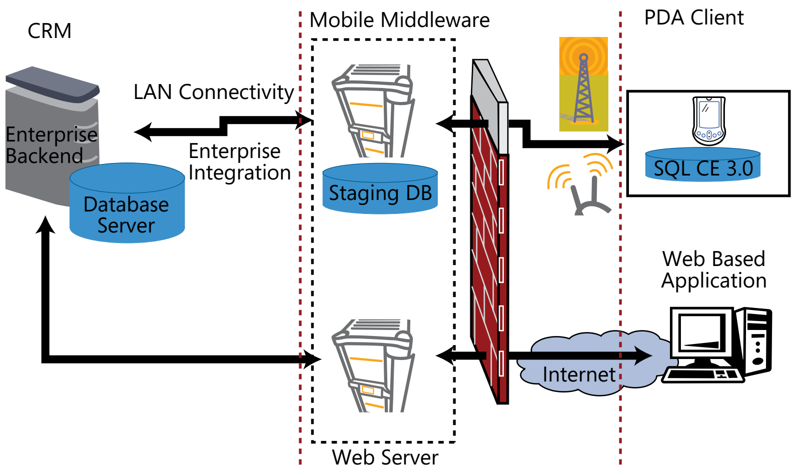
A top pharmaceutical company with pan-India presence wanted to improve the efficiency of its Sales Representatives working in the field. The sales representatives of the company meet physicians in hospitals and clinics to promote the company drugs, distribute samples and promotional materials and, at the end of the day, record all details through a Web-based CRM application.

The proposed handheld-based solution had the following goals:

- Improve the efficiency and effectiveness of sales representatives
- Improve productivity by 10 percent
- Reduce manual process expenses like stationary and telephone
- Provide a quick and easy user interface for successful user adoption
- Upload latest data from device on to the server so that managers can track the sales representative's work
- Download latest product inventory on the device for getting orders from chemists (only if there is connectivity).

This handheld-based solution was designed, developed, and implemented throughout India where the pharmaceutical company is located.

Figure 2: Deployment diagram of territory management system on handheld device



Problem Definition

Sales representatives of the pharmaceutical company make about 10 field calls a day to meet physicians to promote drugs, take orders, and distribute samples. A sales call typically lasts 2-10 minutes since the physicians have full schedules. In this limited time, the sales representative has to discuss the drugs, get feedback from the physician, and distribute samples, journals, and promotional materials. The information collected is captured on paper by the sales representative who, at end of the day, enters all call-related information on to the Web-based CRM application. The sales representative's supervisor views the data and approves the day's work; the management team can also analyze the data and view reports.

This manual process of capturing data has several problems. Data has to be entered by sales representatives on paper and then reentered at end-of-day into the CRM system via the Web. This process leads to data errors and discrepancies.

The **key drawbacks** of the existing manual process were:

- Inefficient data collection process
- Capturing information on paper is time-consuming
- Delays in getting information from the field

Figure 3: Solution architecture for territory management system on handheld device

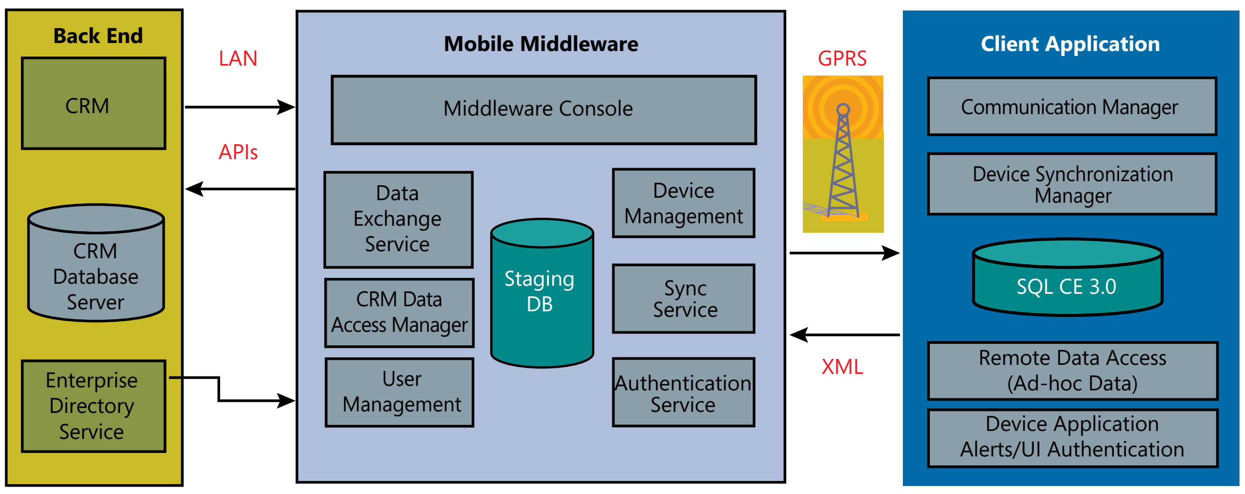


Table 2: Overview of Middleware and Client Application Components in the Solution Architecture for the Territory Management System Case Study

Middleware

Component	Description
CRM Data Access Manager	• Performs updates on CRM database (data captured by sales representative while on the field)
Data Exchange Service	• Composes the changes (out–queue) from CRM back-end system based on the data sub-setting for every user (like doctor/chemist appointment schedule) • Runs in an interval of two minutes after the completion of the previous process • Applies data from the staging database (in-queue) to CRM back-end system using the CRM Data Access Manager (data captured by sales representative in the field).
User Management	• Contains list of users linked with enterprise directory service • Contains user specific information like device information, last synchronization date time, device lockout status and so on.
Middleware Console	• User interface for the middleware • User interface for user management, device management, viewing synchronization logs and data exchange service logs.
Device Management	• Place new application builds in the pre-defined shared location in the middleware • Create builds to cater to different types of devices. (Note the corresponding component in the client application is taken care by Device Synchronization Manager.)
Sync Service	• Place the incoming records in the in-queue • Push the outgoing records from the out-queue • Manage user wise queue like an index table to check if a particular user has any records for download.
Authentication Service	• Authenticates the user by connecting to the enterprise directory service.

Client Application

Component	Description
Communication Manager	• Connects to the middleware for authentication and data synchronization • Tries to connect with middleware first via Microsoft ActiveSync, if not available then connect using GPRS/CDMA • If no connection is available then return with message saying so.
Device Synchronization Manager	• Composes the changes on the client database in XML format • Applies incoming data to local device database • Picks up client application update file from the pre-defined location in the middleware if the file is not available in the device.
Remote Data Access	• Connects to middleware to get updated appointment and campaign lists.
Device Application (Alerts/UI /Authentication)	• Displays alerts on campaign details, missed appointments, and so forth • Authenticates with middleware if connectivity is available or authenticates locally if connectivity is not available.

- Consolidation and decision support delayed
- Delay in sending the latest information to the field
- Expenses incurred for stationary, phone, and so forth
- Insufficient data to talk to physicians and chemists
- No history of calls available with sales representative
- Error factor due to late data entry
- No knowledge of current inventory status of an item in the field.

Solution

Since the CRM solution had already been implemented, an extension on to mobile devices was required with the same set of features. The handheld application had to integrate with the CRM back end seamlessly. This handheld base solution enables the sales representative to capture and transfer information from the field efficiently. The mobile application runs on the handheld, which is carried by the sales representative when they are on the field and has information such as customer data, product, sample, call history, appointment schedules, and product inventory. Figure 2 on the preceding page shows the deployed mobile solution.

The solution has four major components: handheld application, handheld database, middleware, and CRM application.

Handheld application. This application runs on the Windows Mobile devices and is used to capture the data from the field. The application also has a synchronization component to synchronize the handheld data with the server database at office.

Handheld database. This is the database that resides on the handheld. This database has the data specific to the individual sales representative to enable his work in the field.

Middleware. This component, residing on the enterprise end, is used to synchronize the data between the CRM database and handheld device. The middleware uses a staging database that acts as the server for the handheld device. The staged data is then synchronized with the CRM database using native APIs, providing seamless integration.

CRM application. This is the back-end database, which stores the enterprise information. The data specific to each sales representative is downloaded to the staging database and then to the representative's device. The updated data from handheld database is also uploaded first to the staging database and then updated to the CRM database.

The solution architecture, based on the components from our generic smart client model from Figure 1, are shown in Figure 3 on page 14. Table 2 on page 15 lists the components for both the middleware and client application with a brief description of its role as part of extending CRM application on to mobile.

The architecture and process flow can be summarized as follows:

- The sales representatives have handheld devices with mobile application installed.
- Sales representative creates a weekly (can be daily or monthly also) schedule for meeting the doctors, approved by supervisor.
- Representative connects to the enterprise network through GPRS and downloads the data specific to the sales representative in the handheld database.
- The representative meets the physicians and captures the sales call information using the mobile application. Representative also captures which (if any) samples or promotional materials were given to the physician.
- Data is uploaded and downloaded automatically without user intervention; latest data is available to the user via background synchronization.
- Sales representative adds/updates the physician information if a new physician has been targeted or information of existing physician has been modified.
- Sales representative books new orders from hospitals or chemists, aided with a real-time view of inventory status.
- The expenses incurred in meeting the physician and chemists are also captured using the mobile application.
- Managers can view the activities of the sales representative through the reports component. They can also create business plan and strategy after analyzing the data.
- Throughout the day, the manager can track the sales representative working pattern and the data entered.

Addressing Key Challenges

During development, the design team came across many challenges. The challenges and the way they were addressed is described below:

- Cannot make any modification schema to support mobile extension: This challenge was addressed by creating a staging area having a schema structure similar to CRM back-end database.
- Cannot update directly into the tables from the mobile device: This challenge was addressed by creating mapping tables and using wrapper code to call CRM back-end system API's.
- Understanding the schema structure in which data is stored: This challenge was addressed by going through the technical documentation of CRM and going through the table structure in the CRM system database to understand the each field and its use.
- Designing a staging area with the similar schema structure of the back-end CRM system for data to flow to the mobile devices: This challenge was addressed by first copying the structure of CRM back-end database on to staging database, then removing the fields that need not be on the device and last creating data exchange service for efficient integration with CRM back-end system.

"SALES REPRESENTATIVES OF THE PHARMACEUTICAL COMPANY MAKE ABOUT 10 FIELD CALLS A DAY TO MEET PHYSICIANS TO PROMOTE DRUGS, TAKE ORDERS, AND DISTRIBUTE SAMPLES. INFORMATION COLLECTED IS CAPTURED ON PAPER BY THE SALES REPRESENTATIVE WHO, AT END OF THE DAY, ENTERS ALL CALL-RELATED INFORMATION ON TO THE WEB-BASED CRM APPLICATION. THIS MANUAL PROCESS OF CAPTURING DATA LEADS TO DATA ERRORS AND DISCREPANCIES."

Pay-Offs

The sales force automation system has been well-received by the pharmaceutical company, especially by the 2000 sales representatives who are the target users of this application.

The implementation of the system has led to an efficient and effective field data collection process and improved communication between the sales representatives and management. Sales representatives in the field and managers in the office each have access to the information they need—data that is up-to-date and relevant to them—whether making field calls or planning marketing strategies.

About the Author

Kulathumani Hariharan is a senior solution architect working with the Wireless & Pervasive Technologies Practice of TATA Consultancy Services. He specializes in architecting enterprise mobile solutions and defining in the mobile middleware adoption strategy for the enterprise mobile customers.

Connected Consumer Experience in Automobiles

by Christoph Schittko, Darryl Hogan and Jon Box

Summary
What sets a vehicle apart from a hunk of metal and four wheels? It's all about features. Manufacturers are constantly adding newer media devices, more powerful motors, and softer seats, all in the name of improving the driver's experience. Yet with all these improvements, auto manufacturers have barely scratched the surface with respect to the capabilities of the myriad of software and services advances available in recent years.

There has been a lot of talk around services in the cloud and their application based on the consumption of services. Despite this, many reference architectures and papers aimed at demonstrating patterns for designing these systems have taken an idealistic approach to the application—frequently citing simple examples or applying scenarios that are not pragmatic. This paper defines a practical solution architecture based on a scenario one might encounter in everyday life. We intend to inspire architects to use the same approach to define innovative solutions for the problems they face.

The solution architecture defined here is a combination of real platform services that exist today and fabricated services that help round out the solution. This solution will demonstrate the application of Software + Services (S+S) to mobile application architecture as a means to extend the digital lifestyle beyond the desktop. Security, privacy, and data architecture will be addressed broadly.

Scenario

The Woodson family has just purchased a new car with a mobile computer on board. This PC serves as their guidance and entertainment system, but also comes preloaded with several productivity applications, such as travel planning, reminders, and a list manager. The software loaded on the Auto PC not only performs processing locally on the device, but also leverages services in the cloud to enhance the capabilities of the device and to ensure that the information presented by the software is up-to-date.

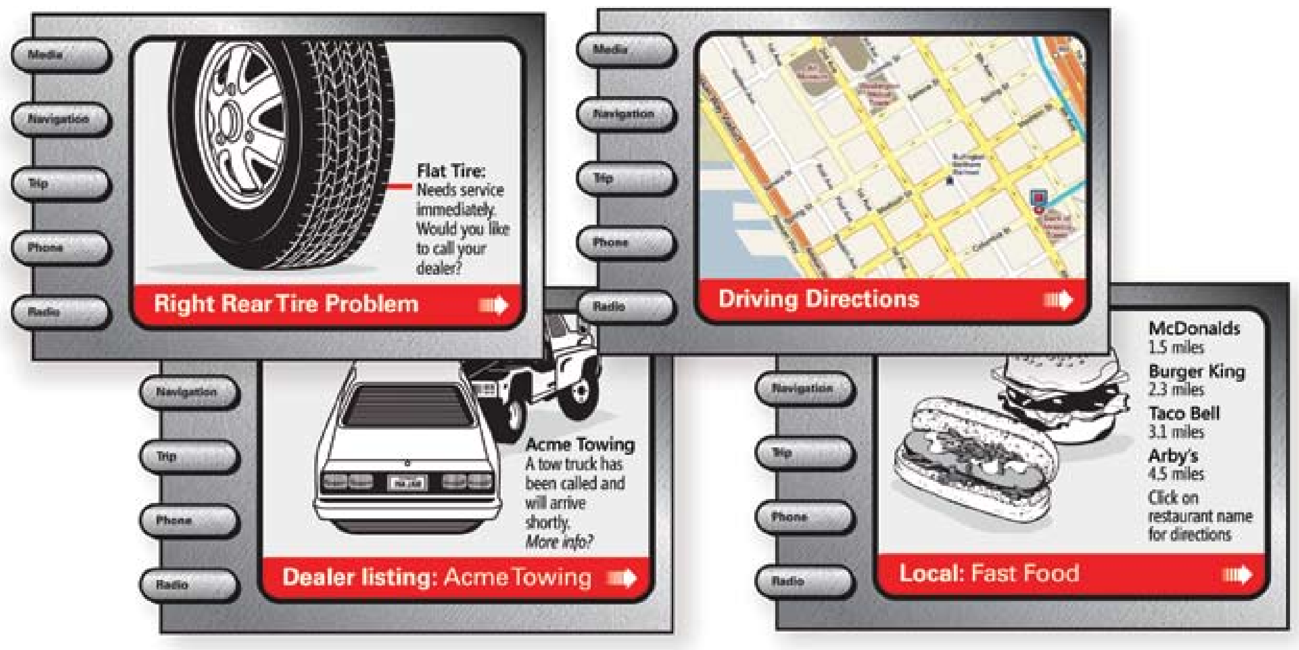
The Woodsons have decided to take their new car on the upcoming family road trip. Mary Woodson is not only the mom; she is the family event coordinator. Mary sits down at the family computer to plan their adventure using a Web-based version of the trip planning application shipped with their in-car computer. Mary plans the route they will drive and makes reservations for hotel and restaurant stops along the way. Mary unwittingly uses a mashup consisting of a number of services running in the cloud. All the information pertinent to the trip is stored remotely in an Internet-based location which is accessible only by Mary and her designates.

"THE INTERNET IS AT THE BEGINNING OF ITS TRANSFORMATION TO BECOME A PLATFORM FOR SERVICES IN THE CLOUD: PLATFORMS SUCH AS WINDOWS LIVE PROVIDE APIS FOR PRESENCE, ALERTING, CONTACT AND CALENDAR MANAGEMENT, AS WELL AS MAPS AND DIRECTIONS; AS ANOTHER EXAMPLE, BIZTALK SERVICES WILL OFFER A PUB-SUB PLATFORM AND MESSAGE ROUTING AND DELIVERY."

When the time comes to confirm the hotel reservations and pay for attraction tickets, Mary hesitates when she's asked for her credit card information. She's heard that entering your credit card information on a Web site can potentially lead to identity theft. She's soon relieved to discover that she can create and use a digital information card managed by her bank to present payment information to the various vendors. This option provides a measure of safety over presenting her credit card information over the Internet to each of these vendors individually. She can select the appropriate card right from her desktop as a source of payment information, allowing her bank to pass her secured credit card information to the necessary vendors without transmitting sensitive information from her PC.

The trip starts with a reasonable lack of eventfulness. The kids have chosen to take only a few of their own DVDs along for the trip. If they decide on a whim that they'd like to see something different they can always download a movie to the car. For Dad, his smartphone connects itself to the car via Bluetooth and his calls, text messages, and email are now directed to the vehicle's computer rather than his phone. An incoming text message from the home security system

Figure 1: The Auto PC offers applications to manage the vehicle and travel.


Media
Navigation
Trip
Phone
Radio
Flat Tire: Needs service immediately. Would you like to call your dealer?
Right Rear Tire Problem
Driving Directions
Acme Towing
A tow truck has been called and will arrive shortly.
More info?
Dealer listing: Acme Towing
McDonalds 1.5 miles
Burger King 2.3 miles
Taco Bell 3.1 miles
Arby's 4.5 miles
Click on restaurant name for directions
Local: Fast Food

indicates that one of the motion detectors has picked up movement in the backyard. Dad places a call to Bob Thomas next door who checks the situation to find it was only one of the neighborhood kids chasing a stray ball.

A few hours into the journey the engine light comes on. Data is immediately sent to a diagnostic service provided by the vehicle manufacturer. The service finds a warranty issue that needs to be tended to and returns a message to the on-board PC with a simple diagnostic report and the name and location of the dealer nearest the car's current location. Mom clicks on the dealer's address to get turn-by-turn directions and the family heads for the detour. The family uses the search functionality on the car's computer console to find a restaurant near the dealer for a quick snack. (Figure 1 illustrates this engine light scenario.)

After servicing their car they are back on their way, three hours behind schedule. Mom pulls up the travel itinerary she created at home. The dinner reservation at the restaurant in the next town needs to be cancelled and new dinner plans need to be made. Mom orders pizza online to be picked up at a nearby pizzeria and pays for it once again using her banking information card. After a great meal they all look forward to a memorable trip.

Solution Architecture

Architecture Options

As you can imagine from the scenario, the solution brings together services from numerous providers and applications that consume these services. The consuming applications could either be application containers following the composite User Interface (UI) pattern to allow dynamic provisioning of new services or it could be special purpose custom applications delivered as client applications. The general approach follows the S+S paradigm to provide a simple, yet rich and intuitive user experience across devices and audiences. The combination of client software and remote services is particularly important in this mobile scenario because the locally running software can improve user experience by masking the high latency over-the-air service invocations or temporary network connection problems that are common even on modern wireless wide area networks.

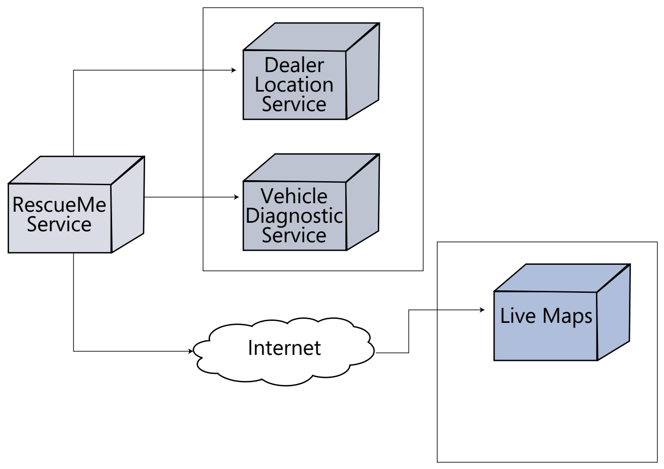
The services fall into three general categories (Table 1):

- *Common Platform Services*, commodity services published by third-party service providers. These are the kinds of services that other service providers will build on top of in order to extend their story, as well as to utilize the maturity and stability of others' infrastructure, specialization, and business efforts (one of the core principles of S+S). Many software and online businesses have realized the opportunity of building out a platform "in the cloud." Microsoft's offering of services under the Windows Live brand is one example for an Internet platform.
- *Manufacturer Services*, provided by the manufacturer either within the car or in the cloud to access vehicle, dealer, and manufacturer information.

Table 1: Connected Consumer service categories relationship to the Windows Live service taxonomy

Connected Consumer Experience Service	Windows Live Service Category
Common Platform Services	Building Block Service
Manufacturer Services	Attached Service
Value Add Services	Finished Service or Attached Service

Figure 2: The RescueMe composite services aggregates car manufacturer and third-party services.



- *Value Add Services*, provided by the car manufacturer or third parties to offer services that increase sales or customer loyalty or services offered to support special purpose devices sold by third parties.

These service categories refine the more generic service taxonomy for Windows Live services listed at: http://www.microsoft.com/online/default.mspx

Technology Environment (Constraints and Assumptions)

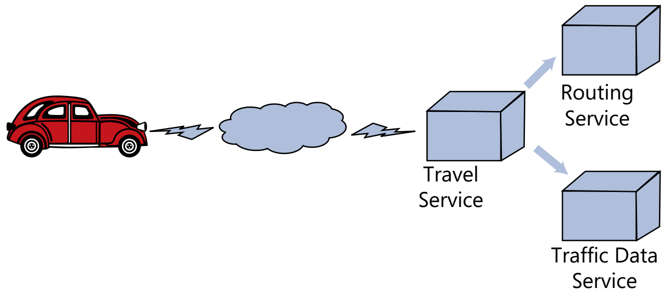
The solution requires an environment where Internet access is widely available, but not necessarily ubiquitous. Ideally, the car has the means to connect to the Internet without additional devices — for example, it does not require a cradled cell phone to enable network access to services provided by the car. Network access to the car enables application scenarios such as remote start, remote diagnostics, and notifications of events in the car.

Some applications that benefit from location information may not need the context provided by vehicle-specific data. Such applications could also run on commodity devices, such as GPS-enabled smartphones and personal digital assistants, because there is no dependency on the vehicle-specific data. These devices are often equipped with modern platforms like the .NET Compact Framework and can access SOAP-based Web services just like a server or a desktop computer. Visual Studio 2008 and .NET Compact Framework 3.5 add support for consuming WCF and WS-* based web services for WS-Addressing and message-level security based on WS-Security.

Value Add Services

In our scenario, the Woodson family touches many different types of software and services. For instance, the vehicle manufacturer provided a service to analyze vehicle diagnostic data and to notify the vehicle owner of a problem along with information on where to get the problem fixed. The trip planning service stored their travel itinerary and made it accessible from the Internet. These are value-added services which can be more completely described as services which provide a unique experience to the consumer. They differ from common platform services in that they are not considered general

Figure 3: The most flexible architecture consumes only remote services.



purpose or widely available. Instead, they are most likely developed to establish a competitive advantage and perhaps to extend the usefulness of another product.

In many cases, these services may be composites of custom code and one or more core platform services, allowing the service provider to make available features where they are lacking domain-specific knowledge. An example of a composite application would be a dealer locator service (Figure 2). An automotive manufacturer may know the location of their dealerships, but it is unlikely that they would have the data necessary to provide navigational guidance to the dealer from a specific location. The manufacturer would most likely rely on services provided by Windows Live Maps or Mapquest.

In and of themselves, value-add services would not be considered finished products. These services are domain-specific building blocks to build complete applications. We can say that these services provide the features used to create a more complex and complete piece of software.

The most flexibility is gained using a deployment model that consists solely of services running in the cloud, because changes only require updates to the servers hosting the services, not to each car that consumes the service. These services are typically HTTP-based services that can be invoked by a client — either an application or another service. The benefit of this approach is that all the executing software is centrally deployed to a data center making it easier to manage. The drawback is that the service consumer must have a connection available to make use of the service (Figure 3).

Although it is possible to run the complete solution on the client, this pattern constitutes a closed environment where access to more relevant and up-to-date data is not possible, making the solution less useful. This is the blueprint that exists today for many mobile computing systems, especially those based in automobiles. The upgrade path for these devices is non-existent and the limited functionality they provide is of minimal benefit to the owner of the device (Figure 4, page 20).

A preferable pattern for a mobile computing solution takes advantage of the ability to access software and data stored on the local device. In this case we can deploy the value-add logic to the device and make enough data available to the application that the application would be useful even when the connection is unavailable. This model decentralizes a great deal of integration and control logic and introduces maintenance and bug fix challenges, but the improvement in the user's experience will likely make the pains worthwhile (Figure 5, page 20).

Figure 4: The least flexible architecture consumes only local services offered by the vehicle.

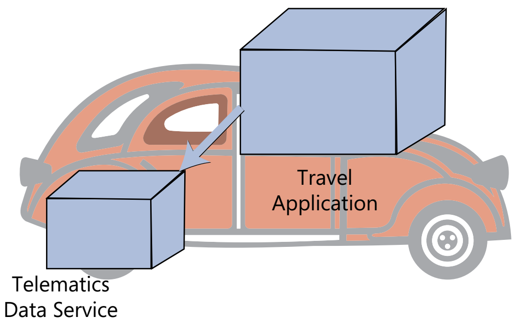
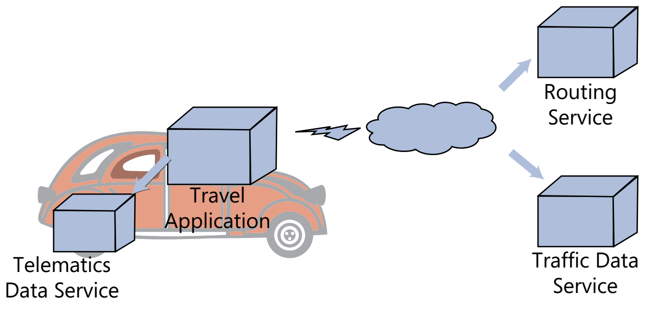


Figure 5: The Software+Services architecture combines benefits of local and remote services.



Services Technology Platform

The Internet is at the beginning of its transformation to become a platform for services in the cloud: Platforms such as Windows Live provide basic APIs for presence, alerting, contact, and calendar management, as well as Virtual Earth maps and driving directions; as another example, BizTalk Services will offer a pub-sub platform and message routing and delivery. These are all key ingredients for rich, robust connected applications, but current feature sets and SLAs reflect that these services are still early in their life cycle.

Future services could extend presence settings with a "driving in the car" setting that always allows traffic alerts and some alerts that the driver configured—the alert API could add "car" in addition to "IM application," "email," and "SMS" to the list of notification endpoints.

Mobile Client Technology Platform

There are several platforms to choose from when it comes to realizing the connected consumer experience and there are trade-offs between the platforms. The priorities and constraints dictated by the actual solution have to drive the platform selection. In general, Windows Vista Embedded enables the richest experience through the feature set of the operating system and the full .NET Framework, but it's also the platform with the largest footprint, the most demanding processor requirements, and highest licensing cost. Windows CE provides a lower cost alternative with lower hardware requirements, and more options to customize the operating system but fewer capabilities. A Windows CE-based platform should include the .NET Compact Framework to take advantage of the productivity benefits of managed code development and base class libraries. Finally, Windows Mobile provides a constantly improving richer experience. Platform services are provided through the compact

Table 2: Client platform decision points

	Windows Embedded .NET Framework	**Windows CE .NET Compact Framework**	**Windows Mobile .NET Compact Framework 2.0 (3.5)**
Target Scenario	Rich In-Vehicle Scenario UMPC	Low Fidelity In-Vehicle Scenario	Consumer Device
UX	Full WPF feature set	Windows Forms Silverlight in the Future Touch	Windows Forms Silverlight in the Future Touch
Communication	WCF SOAP, REST and JSON	WS-I SOAP Web Services WCF Client (3.5)	WS-I SOAP Web Services WCF Client (3.5)
Interaction	Speech with Vista or Third-Party Add-OnTouch	Speech Third-Party Add-On Touch if hardware support	Speech Third-Party Add-On Touch on PocketPC devices
Authentication	CardSpace	Username	Certificates
Data Storage	Full access to local storage devices and a variety of databases	Local file system Optional SQL CE	Local file system SQL CE
Development Tools	Visual Studio	Visual Studio	Visual Studio
Car Integration	Yes, via serial or custom ports	Yes, via serial or custom ports	No

framework, providing a programming model consistent with the skills of a broad set of developers. Table 2 on the preceding page lists the strengths and limitations of each platform for mobile application development.

Client Application

Several options for interacting with computing systems are available to us today. Rich client applications realize the greatest benefits consuming services because they can take advantage of local data storage mechanisms and computing power that is not available through Web-based delivery mechanisms. Web applications have the benefit of being centrally hosted and managed but, being connection-dependent, cannot fully realize the connected consumer experience. A rich client application, in addition to providing the most comprehensive user experience, is more readily able to take advantage of services that might only be available locally. It is more practical to consume and evaluate vehicle performance data locally in the vehicle rather than passing the data into the cloud for further processing. We also avoid any privacy and security issues when we don't transmit data from the car to remote services.

Lastly, a rich client application can handle identity and session more easily than a Web-based application. It is a trivial exercise to store information cards locally on a device and use those cards to establish identity with a service or another application.

Service Delivery Patterns

We can expect that Internet connectivity is widely available to our solution, but we cannot make the assumption that connectivity is ubiquitous. Each service delivery pattern has different strengths in terms of latency, flexibility, and functionality (Table 3). A rich client allows us to adopt a set of design patterns that allow us not only to account for those times when we are disconnected from the network, but also to use the additional capabilities of the client to enhance the user experience.

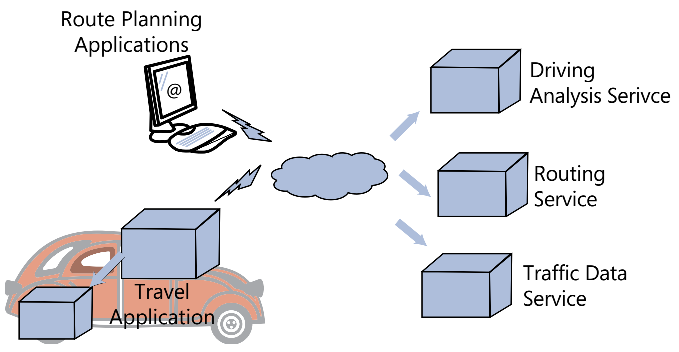
Many rich applications built these days are only shells providing input and output for a set of services living behind the user interface. In this pattern, the client is highly reliant on connectivity to provide any kind of useful interaction for the end user. This client may have simple caching to deal with network latency or conditions where the network is simply not available, but it does not extend its usefulness beyond what is available through the services it consumes.

We can extend this pattern to create a second, more useful pattern in which the client remains a dependant service consumer, but the computing infrastructure is extended to the client. In other words, we take advantage of the application's ability to perform processing tasks in the vehicle, thereby relieving the services of some of the processing burden. A typical result in this case is a more responsive albeit still underachieving application.

Table 3: Comparison of client architectures

Pattern	Latency	Flexibility	Functionality
Thin service consumer	High	High	Medium
Richer client	Medium	Medium	Medium
Smart client	Low	Low	High

Figure 6: Different heads allow for better utilization of services and a more differentiated user experience.

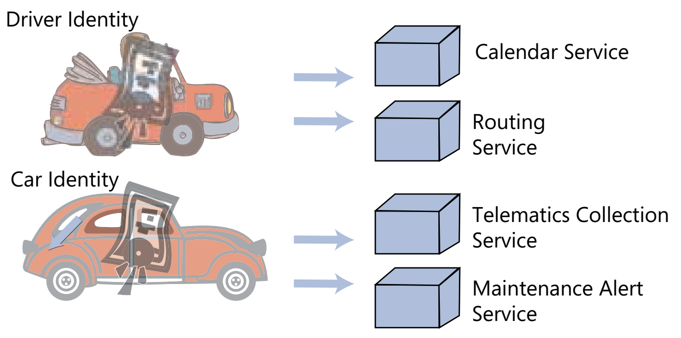


Short of downloading all functionality to the client device, we can implement a pattern in which services become simple data providers and consumers. The logic for how we utilize the data is embedded in the client and the client becomes the focal point of the user's experience. This pattern allows us to deal with latency caused by slow or non-existent network connections as well as presenting to the user an experience resembling her home PC. One issue related to this pattern is that of upgradability. Because the application is deployed to the client device it becomes more difficult to add functionality or to upgrade existing functionality. Services such as the .NET Framework's click-once deployment model and adoption of composite UI application patterns help overcome this challenge. New services can be readily upgraded and added, as is the case with any Web service.

It is possible for software vendors and service providers to offer multiple versions of their products—for example, a vehicle-independent version for devices offered through regular retail channels and a vehicle-specific version offered through the vehicle manufacturer that offers additional functionality (Figure 6). An enhanced model would "light up" the experience on the device and in the car when a customer buys both products. The mobile device application could also provide value-adds, such as the display of the current location of the car, remote start, and automatic data sync from the device when it's brought into the car. Both, the device application and the vehicle application can access the service provider's cloud-based services, which increases their utilization because of the larger target audience.

With the S+S architecture, client applications are not bound to the vehicle. Services can be offered through myriad commodity devices such as smartphones and mobile PCs with user experiences tailored to those platforms. Because data stored in the cloud is available to any and all devices, we can make the transition between devices almost completely seamless. The important tenet to maintain is that interaction with the devices should be a natural experience for the user. The growing collection of more and more powerful devices presents many options to present a consistent level of interaction between client and computer. The user interface may change to suit the form factor, but the level of service will remain the same.

Figure 7: Car and drivers need their own different digital identities to access services.



Security, Identity and the Client

Identity is a challenge in any system as are authentication, authorization, and privacy. This is especially the case with mobile applications. The greater the remove between the core application and the device, the more difficult it becomes to deal with matters of security. A richer client allows us to maintain security tokens on the device, creating an arguably more secure application.

Secure Communication

Privacy is a major concern for anyone exchanging data with services in the cloud. In the past, the focus was on HTTPS and SSL for transmission of encrypted information. The problem with SSL is that it is a point-to-point protocol and doesn't allow for data to be exchanged securely between multiple endpoints. Composite applications (comprising multiple services almost by definition) require an end-to-end approach to security. A better solution would be to obscure the data on the device and allow it to be transported over an encrypted or clear connection. This is the approach taken by the authors of the WS-Security protocol. The client device possesses a public key used to encrypt the data before it is release for transport. Only the destination service is able to the decrypt the message. Data in the message intended for different recipients can be encrypted with different public keys, thereby ensuring that data can only be read by the intended recipient. Client-side development platforms like the various versions of the .NET Framework implement WS-Security as part of WCF.

Digital signatures provide an added measure of trust. The client could use a unique identifier such as a private key from an X.509 certificate to sign the message, ensuring that the data was indeed sent by the party whose identity is claimed in the message; the digital signature would also provide the assurance that the data was not tampered with in transit. A digital signature is essentially a one-way hash of the originating data. If the hash cannot be reproduced by the recipient, the signature is understood to be invalid: Either the key pair does not match, invalidating the claimed identity, or the data has been tampered with in transit.

The S+S approach enables use of WS-Security, but it offers an even more secure option to improve privacy: Not transmitting any secure information at all. Moving computing operations that include personally identifiable data to the car or the device eliminates the need to transmit the information over insecure channels. Take CardSpace authentication as an example. A CardSpace identity tied to the car avoids transmitting personal identity or weak username/password combinations.

In the interest of privacy, service providers are encouraged not to require any sensitive information. For all exceptions, the application provider should request the user's permission for transmitting information to the service. By default, each application should follow Microsoft's guideline for secure computing and not transmit any sensitive information without explicit permission from the user.

Authentication and Authorization

Identity tokens can be used to authenticate and authorize users for services they would like to access. An information card such as a CardSpace card would allow the user to present a claim of identity to a service. The burden of verifying that identity could be held by the service or a relying party could be used to validate the identity. Authorization would still be the responsibility of the invoked service.

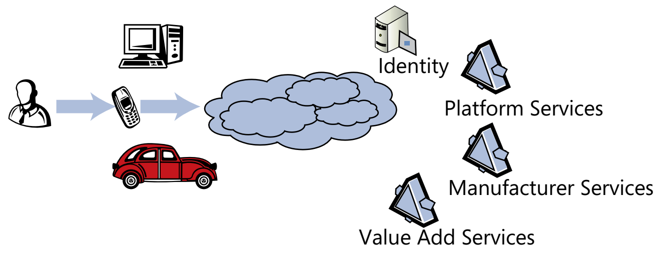
Authentication is very important in our mobile scenario for a number for reasons: We need to restrict access to the application to paying customers, but we also need to ensure that each user's private data is protected from unauthorized access. In our scenario, there's an additional concern: Remote access to the car. There are privacy concerns around accessing the car's location and travel history, but there are also safety concerns. Starting or stopping the car, for example, is a feature that needs to be guarded very tightly. You wouldn't want a malicious hacker to shut off your car's engine while you're driving down the highway.

"THE S+S APPROACH ENABLES USE OF WS-SECURITY, BUT IT OFFERS AN EVEN MORE SECURE OPTION TO IMPROVE PRIVACY: NOT TRANSMITTING ANY SECURE INFORMATION AT ALL. MOVING COMPUTING OPERATIONS THAT INCLUDE PERSONALLY IDENTIFIABLE DATA TO THE CAR OR THE DEVICE ELIMINATES THE NEED TO TRANSMIT THE INFORMATION OVER INSECURE CHANNELS."

On the Internet, user identity is typically established by entering a username/password combination but that's not the experience we expect when we get into a car. The key is the traditional means of getting access to the car and its services. We can employ a similar interaction model for in the connected services scenario with smart keys or CardSpace-based solutions.

However, there are a few interesting architectural concerns around identity in the car. For one, the car itself is a multi-tenant application because it can have multiple drivers potentially with different roles—the owner, the owner's teenage daughter, or a mechanic that services the car are a few examples. Many cars today offer preference settings for seat and steering wheel positions for different drivers based on the key they carry. This experience can be extended to computing

Figure 8: The Internet services platform enables connected consumer experiences in the car, on devices and the PC.



devices in the vehicle, making applications and data available based on the current driver's identity. Access to the entire system can be limited for guests in the car.

The concept of identity exists even for the vehicle itself (Figure 7). A digital identifity could be assigned to individual cars allowing access to manufacturer services that provide vehicle-specific services. Drivers on the other hand need a portable identity. Most computer users today have at least one digital identity associated with them. Extending that identity to be used in a vehicle is not trivial, but is entirely possible.

Multitenant Data Architecture

Another factor to consider is the problem of data storage and privacy. In order to make this type of computing experience useful a good amount of personal information would need to be stored in the cloud. This creates a challenge for the data architect who must make sure that the data is stored in an efficient manner while not compromising the security and privacy of a consumer using the service.

Arguably the best solution for a data store with a large number of tenants is the shared database, shared schema method. In this case, the data of every tenant is stored in the same tables with data associated to each tenant through metadata. This pattern places the guarantee of privacy and security on any application accessing the data and may dictate additional software development costs, but the cost savings for the long term maintenance of the data far outweighs this cost.

Conclusion

Connected consumer experiences such as the one outlined in this article present a great opportunity to add value to existing products. The current and upcoming generations of consumers are technology savvy and will rely on digital helpers everywhere, not just on their desktop at work. The ubiquity of computing devices and the wide availability of network connectivity present an opportunity for manufacturers and new service providers to connect with their customers in new and meaningful ways (Figure 8).

The car in particular is such an important part in many people's life. You bring kids to school, visit customers, or take road trip vacations. You may spend hours each week driving around and you find yourself in situations where some extra help can make a big difference to you.

The Microsoft Platform is very well suited for building S+S solutions on the client and on the server. Technologies like WCF and the .NET framework are well-suited to building cloud-based services because support for message exchange protocols—such as JSON, POX/REST, SOAP, and WS-*—guarantees interoperability with all kinds of service consumers. Often, services are not built from scratch but by aggregating existing services. Technologies like BizTalk Server or the cloud-based BizTalk Services are well-suited to aggregate building-block services into value-added services. The platform also offers Windows Live building-block services, such as contacts, alerts or photos, which can be included in value-added services.

Software + Services provide an excellent pattern for delivering services across a number of platforms. Flexible service delivery mechanisms allow us to quickly add new features with little interruption to existing systems. Advances in presentation technologies and device form factors enable us to present software to users in the most context-appropriate manner.

We're already seeing the combination of Software + Services emerging in many areas. Early adopters are proving the value of these solutions and setting the bar for others to meet. The tools and the platforms are there. It's only up to the application providers to build solutions that reach users in the best possible ways.

About the Authors

Darryl Hogan is an architect in Microsoft's Developer and Platform Evangelism Group. Darryl has extensive experience architecting and implementing numerous enterprise applications during nearly 15 years in the IT industry. In his current role, Darryl provides guidance and education to architects implementing enterprise solutions and enterprise architectures on Microsoft technologiers.

Christoph Schittko is an architect for Microsoft based in Texas where he works with customers to build powerful solutions that combine software + services for cutting edge user experiences and leveraging service-oriented architecture (SOA) solutions. Prior to joining Microsoft, Christoph assisted with companies adopting service orientation and delivering Software-as-a-Service (SaaS) solutions. Christoph has over 14 years experience developing and architecting software solutions in a wide variety of industries. He writes and speaks on Web services and XML at various conferences. Christoph holds an advanced degree in Electrical Engineering from the Friedrich-Alexander University Erlangen-Nürnberg.

Jon Box is an architect evangelist at Microsoft. He works with customers to utilize Microsoft technologies to build impactful solutions. Jon has been programming professionally since 1985. He has worked in a variety of environments and languages that include COBOL, Assembler, Clipper, C, C++ (Borland, ATL, MFC, Win32, COM/DCOM), VB5/VB6, and .NET. For more thoughts from Jon, see his blog at http://blogs.msdn.com/jonbox.

Architectural Journal Profile: Faisal Waris

In this issue, we catch up with Faisal Waris, an architectural consultant at Ford. *The Architecture Journal* asks him about the role, what some of his challenges are, and his views on architecture.

AJ: Faisal, can you tell us a little about yourself?
FW: I am a consultant at Ford Motor Company, working for Synova Corporation. Essentially, my main role at Ford is as an SOA architect, and I have been here for four or five years. I am also the co-chair of an AIAG workgroup that deals with business-to-business (B2B) messaging. AIAG stands for the Automotive Industry Action Group, a standards body for the automotive supply chain. It has a fairly large membership consisting of various OEMs and suppliers.

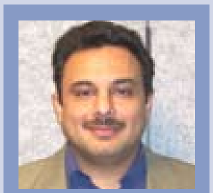
AJ: Four or five years at Ford as an SOA architect sounds like a fascinating role. Can you elaborate a little more on your work?
FW: The team I work in is part of the Enterprise Architecture Group, which takes on various roles. Firstly, we are similar in some respects to a research organization, looking at emerging technologies, performing proofs-of-concepts, and experimenting with software and devices to figure out if something will add value to Ford. Then, we engineer that technology for the mainstream application teams. The other role of the group is about setting standards, which can include defining standards and best practices around Service-Oriented Architecture (SOA) and other architectural topics. SOA governance is a major area that we're currently working on. This involves creating processes, frameworks and usage guidelines for enterprise-wide products and services that are being introduced into Ford. We also help application teams implement SOA Web services and troubleshoot when things go wrong.

AJ: Many of our readers may be implementing SOA. What recommendations would you share with them?
FW: One of the debates we have here—and, I believe, one that's probably happening in most other places—is the question, *What is a Service?* What does a service consist of? If I have an FTP job, is that a service? Does the word service only describe a Web service, or does a service in AJAX count also? I think a lot of companies, including ourselves, struggle with the definition. We have decided to adopt a very pragmatic way to deal with it. To resolve this we use the OASIS definition. You can go to OASIS and actually get that from their Web site. We decided to use that as our definition because a lot of people worked on it, so it's kind of a consensus view on how to define SOA. It's still a very generic definition, but what it allows you to do is to tailor it for specific use. If you think about it, almost anything can be a service, but if you want to have an architecture that connects all of these services together in a uniform way, deferring to the OASIS definition, then you have to start to think about what you really want a service to be. We found that if you want uniformity across your services then the first step is to establish a certain set of standards and protocols.

AJ: You mention a focus on governance. Do you think many organizations today struggle with governance?
FW: I think it depends on the maturity of the organization—or the maturity of SOA in the organization. If you are just starting out, then heavy duty governance is not what's really needed. What's needed at that time is helping and nurturing the nascent SOA in an organization. This can include SOA evangelism, talking to your application teams, and explaining concepts and how things get done. After a certain point in time—when everybody is comfortable and services are being built—then you can step back and put on more of a governance hat. As an organization we're now making this transition. We have done the evangelizing, and SOA has become accepted. We are now in the phase where we need to start thinking about how we manage all of that, making sure the right services are being built, avoiding duplication; given that we have different application teams with different points of view, how do we broker the right service so we have the right consumers and the right providers and the right set of interfaces.

AJ: You also mentioned co-chairing an AIAG workgroup. Can you give the readers a little kind of background on AIAG, especially related to the mission of the group?
FW: AIAG is more than just an action group; it is both a vertical for the supply chain and a standards body. It brings together suppliers and OEMs, and gets them working together on common problems. There are engineering type standards, and there are process standards, but more and more they are getting into eCommerce type standards—these are standards related to the exchange of information between suppliers and OEMs. I am part of the group that works with those sets of standards, and our goal is to enable seamless integration between automotive supply chain partners by defining standard business processes. A standard process could be something like inventory management process—say *Kanban, MinMax*—Quality or Warranty processes. We then take these processes and perform proof of concepts using Web services and related technologies (such as ebMS). We demonstrate that many implementations of a business process (with interfaces defined by a set of WSDLs) can interoperate securely and reliably. Our goal is to promote this in the industry.

AJ: The current issue of the Journal is about mobile applications


Faisal Waris
Synova Corp.

Faisal Waris works for Synova Corp. and is a consulting SOA Architect at Ford Motor Company. Lately, he has be active with the Sync program (www.syncmyride.com). Sync is the result of a partnership between Microsoft and Ford and is one of the first production applications of Microsoft's Automotive PC. Faisal is also the co-chair of a workgroup at Automotive Industry Action Group (AIAG, www.aiag.com) where we works on automotive electronic commerce standards.

and devices. Some readers may have heard about Ford's new Sync product. Could you tell us a little about that?
FW: Sync is a partnership between Microsoft and Ford, and is one of the first applications of the Microsoft Automotive PC in production use. Sync is very interesting. In its current context, Sync provides services related to entertainment, voice recognition, operating the phone, and media devices. One way to think about it is as a general purpose computing platform in the automobile. With this paradigm, the sky is the limit; there are a lot of possibilities as to what you can do with it. Of course, there are challenges related to what you can put in a car from a user perspective because of many considerations. But a new world is beginning and we are very excited to be a part of that.

AJ: How do you see the role of software in the automobile evolving over the coming years?
FW: Ford is obviously thinking of many different areas, some of which I can't cover in this interview for confidentiality reasons, but a general purpose computing platform in the car opens up a lot of dreams and possibilities. There is a potential for providing many services inside the car that can operate on voice recognition especially one that can perform well in an automotive environment.

AJ: We find that many readers of the Architecture Journal are aspiring architects, perhaps senior developers looking for the next step in their career, and thinking about what they need to become an architect. As someone who has been doing this role for some time, what kind of advice would you give a would-be architect today?
FW: I think that there's a gradual process where someone transitions from developer to development lead, and then on to architect. There are also very different types of architects. At Ford we have infrastructure architects, solution architects, and even specific architects like SOA architects. If you are an aspiring architect, I believe that being current with the literature is key because being able to understand how to take a set of technologies and stitch them into a solution requires fairly broad knowledge, and fairly current knowledge of what's out there. I have to do a lot of reading just to keep up with everything that's going on. I'm also more of a hands-on architect, so even though I am playing an architecture role I like to go in and roll my sleeves up and tweak with code. I've found that's this is really beneficial. A lot of architects distance themselves from doing any kind of hands-on work, which can be a mistake because you really can't architect unless you understand the whole picture and sometimes you have to understand at a very deep level to be able make the right kind of decision. My advice would be to play around a lot with new technologies and experiment. You don't have to write production applications, but what you can do is experiment so that you have a sense of what something can do, what are its limits, what are its capabilities, and that overall gives you a much better perspective as to what will work and what won't work.

AJ: One of the questions we always love to ask the people we interview is "What's the one thing that you regret most in your career, and what did you learn from it?"
FW: That's a hard one to answer! I can't actually remember anything that I really regret. I have had many chances to go purely into management, and I have steered myself clear of that. I've always had one foot in the technical area. I think that has helped me to be where I am today. Being in this position I do think there may be a limitation that I start to run into, and the question can be how do I advance in this position? Should it be a move into management, or do you stay in an architect consulting role, or how do you move forward? That's one of the things that I need to figure out.

AJ: Related to your career, what do you hope to accomplish in the next few years and over the long term?
FW: One of my personal quests is to establish a new way of dealing with B2B messaging. If you look at what we have today in terms of messaging, it is mostly Electronic Data Interchange (EDI). There's some XML being implemented, but if you look at it, it is still mostly EDI over FTP. My personal quest is to enable B2B Web services, and I believe we have all of the components out there. We have the WS-* specifications that work well in B2B space. For example, we have WS-Addressing for asynchronous messaging, WS-ReliableMessaging for robust delivery of messages, WS-AtomicTransaction for transactions, WS-Security and WS-SecureConversation for security, and so on. Many of the toolsets are now implementing these standards, so one of the things that I am working on here at Ford and at AIAG is to establish these as the new standards for B2B messaging. Of course, it's very hard given the large installed base of existing technolgies—but slowly, we start to see people recognizing the value of exchanging messages with XML because you can do a lot more with it. Unlike EDI, many aspects of XML messaging can be managed just by leveraging metadata (such as XML Schema, WSDL, WS-Policy, etc.). Plus, you can create robust integrations (through reliable messaging), which is difficult with something like EDI. That's my personal cause, so it's very rewarding and exciting.

The other area of technology I am personally interested in is the semantic Web technologies. Now this may be getting a little old at this time—I think it has gone over the "hype curve" and may even be in the "trough of disillusionment" —but when I work with information, I see how it can be managed better, and I always go back to the capabilities promised in the vision of the semantic Web. I hope that this need will be recognized by others and maybe there'll be a resurgence.

At this point I am happy with what I am doing and very, very busy, so I haven't had much time to think about the long term. I definitely like the architect role and I don't know whether I want to grow out of it any time soon.

AJ: Faisal, thanks for sharing some of your insights and thoughts!

If you would like to nominate someone who would make a good candidate for the profile in the next issue of the Journal, please contact the editors at editors@architecturejournal.net.

Mobile Data Architecture

by Rodney Guzman

Summary
Applications that are occasionally connected have a reputation for being difficult to implement. The challenges are layered, and the heart of the problem is how data is managed. Each occasionally connected client consumes and produces data at random intervals. Islands of data are produced when disconnected, and must be reconciled later when connectivity is restored. Technologies exist to move data to and from the clients, however, only the simplest of data reconciliation tasks are handled without custom code intervening. For example, SQL Replication is a fantastic technology for moving new data from one system to another. But when a record is updated by more than one client, should the last one in always win? Also, what happens when new data entities are created on multiple clients, but they each represent the same unique entity instance? Unfortunately, data models are usually too complex to be handled by generic conflict resolution processes.

This article will discuss how to build your data model to support an occasionally connected application. The data model supports the introduction of a complex conflict resolution process built in .NET. A point-of-service sample application is introduced with steep requirements, including: Web and smart client entry points; smart clients will be occasionally connected; data entities that span multiple tables; and a complex conflict resolution process which includes support for new records that can be created on the Web application and/or smart client application instances.

Scenario

For the purposes of this article, a sample event management application is introduced. This application is a hosted Web site with a robust data model. The business demands that this application be accessible in remote, disconnected areas where the events are held. At the events, the company's products can be purchased and attendee information updated just as they can on the Web site. Each event lasts for potentially multiple days with thousands of attendees. During the event, there is no regularity at which anyone can assume to have network connectivity. Shortly after the event is completed, the data on each client much be reconciled. The simplified database diagram shown in Figure 1 is referenced as a sample subset from this application.

Re-use of existing components to minimize the cost of building a smart client application has been demanded by the business. The business does not want to rebuild their business objects already abstracted away correctly from their ASP.NET application. They want the same business objects to be deployed with the smart client Windows Presentation Foundation (WPF) application. SQL Express will be in use on the desktop, and the data schema will re-used from the Web application.

When You Are Connected

You can never rely on when occasionally connected applications will be connected, which imposes a tremendous burden on your application. An application must be connected at certain times to perform base functions. For our example scenario, to optimize the event experience for attendees, pre-registration information should be accessible on the disconnected client during the event, so the device is synchronized with the central data warehouse prior to the event. Regardless of how the information is interchanged, the device needs to be connected before the event to obtain this information.

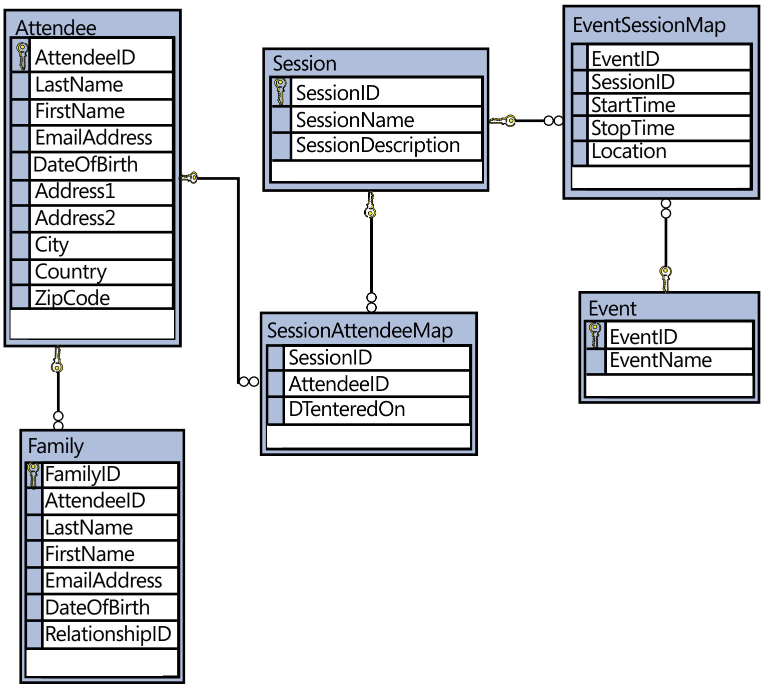
The initial requirements for the application included a model that supported an application that never required the user, at any time, to be connected to perform a function. Ironically, the user has to be connected at some point in the life cycle of using the application in order for data to be synchronized to and from the server. The lack of control over when the application might be connected poses complexities that cannot be ignored. In the case of this application, requirements were modified to force the application to be connected at key moments in time.

Complexity of Never Knowing When You Are Connected

SQL Replication supports very well the concept of data interchange with occasionally connected clients. However, if you are managing a multistep process in which each step relies on some form of connectivity, knowing where you are in the process can be challenging. In the event management application, to rely exclusively on SQL Replication, the process flow would have to resemble the following:

1. The list of available events would be replicated to each client device. If a new event is added, or an existing one modified, it would need to be synchronized to each device.

Figure 1: Extremely scoped down version of the SQL tables



2. Authorization for who can see which event would be replicated to each client device. If authorization changed, it would need to be replicated.
3. On the local device, the user selects the event to synchronize, placing a record in the EventDownload table in the local database which includes the host name of the client machine and the event to replicate.
4. The EventDownload table is replicated back to the server.
5. Filtered SQL Replication brings down the requested events to the user's device.

This process flow seems simple enough but requires connectivity throughout the process. Imagine having a conversation with a nontechnical user about why he could not see the event on his device because the authorization granting him access changed after he synchronized his machine. Relying on a model where you cannot ever control connectivity can be complicated to debug and maintain.

Relying On Key Connected Moments

Ultimately, connectivity at some point is required. In contrast to the pure SQL replication model, a more direct approach can be taken to simplify the overall processing. Enforcing connectivity at key points avoids several of the round trips described in the previous section. For example, rather than synchronizing the list of events to the local database, a Web service can be invoked to retrieve a list of authorized events the user can synchronize with, changing the flow to:

1. The user launches the application and retrieves a list of events via a Web service call. Only authorized events are shown to the user.
2. The user picks an event and calls the Web service again. On the server a record is set in the EventDownload table with the host name of the user device.
3. Filtered SQL Replication rules associate events to client host names, specifiying which events will be replicated to which client device (Figure 2).

A whole host of additional information can be obtained through the Web services, from the number of records to be synchronized to progress updates on whether a replication has occurred or is complete.

Conflict Resolution

SQL Replication and ADO.NET provide no silver bullet for conflict resolution. Most of our applications are simply too complicated to handled by out-of-the-box solutions. We cannot always rely on the "last one in wins"—typically, we need to wait for data to be accumulated from multiple sources before making a decision on what "the best" record is.

This outlook seems abysmal, but there are techniques to cut through this complexity. In the event management application, we rely on the strengths of SQL Replication as a cornerstone to the solution. A perceived strength of SQL Replication is that it is fantastic for managing SQL inserts. For updates, however, there were too many scenarios where we needed fine control over the conflict resolution process at scheduled intervals.

The event management application presented several complications to this process. For example, an attendee record is not represented by a single table in the database. This complicates replication of information that has not changed as change is what drives SQL replication. A logical record of the attendee was formed, which is a record that spans multiple SQL tables. When a change occurred on any component of the logical record, a logical transaction was produced. This concept is important because it allows us to move information to the data warehouse even if change did not occur.

Most applications suffer from this complexity, and it's what makes replicating data back and forth so difficult. We broke down this complexity in the event management application, and the pattern we followed could be implemented in any disconnected application.

Logical Records

In the context of the event management application, a logical record for an attendee is not only the information within the Attendee table, but all related information. Selected sessions, product purchases, and family members all make up the attendee logical record. If any one of these pieces of information is changed, regardless of what table the change was made in, then the attendee logical record has been changed.

A base requirement for this application is that if an attendee's information is touched at the smart client, a validation of the entire

Figure 2: EventDownload table specifies which events will download to which client hosts.

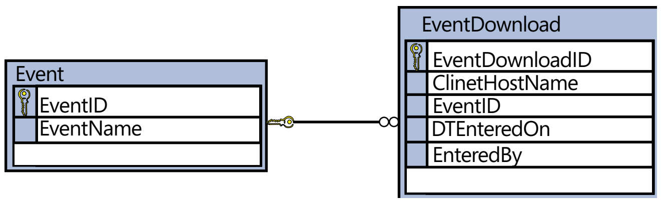
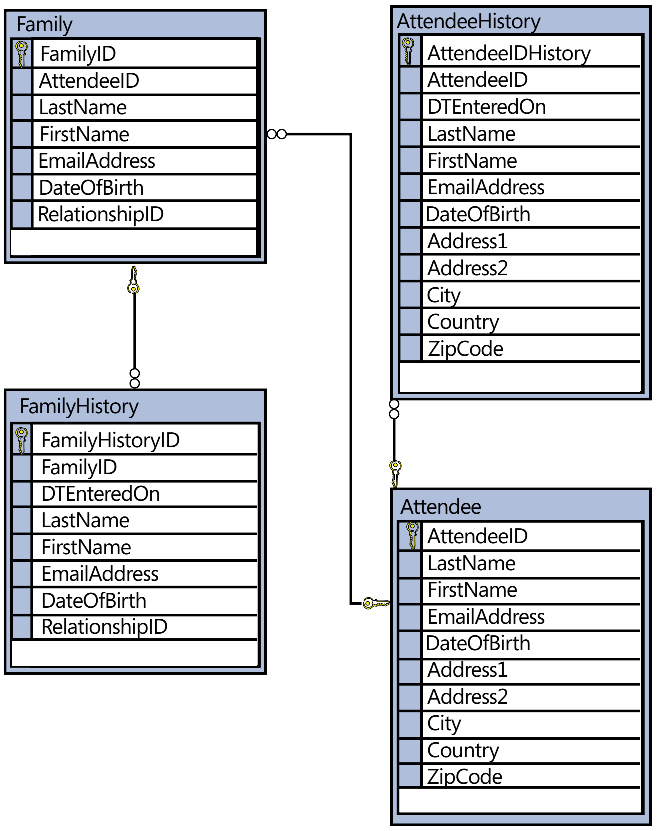


Figure 3: History tables on key parent tables allow "transactions" to be maintained.



attendee logical record has occurred. Therefore, even if only a single table was modified, each table's data that represents an attendee logical record has been validated as being correct at that moment in time. This poses an interest dilemma in that standard SQL replication processing cannot automatically synchronize all table data related to a logical record if the data has not been touched. We managed this problem by forcing a change to occur in the database to trigger the appropriate set of events—what we call "logical transactions"—for SQL Replication.

Table History

The existing Web application's database has a history table associated to every user modifiable table in the database. The purpose of these history tables is to keep a history of all changes, when the changes were made, and who made the changes. A requirement of the smart client application is that all of the local history changes are recorded and then replicated back to the central data warehouse. All of the history tables are populated by a trigger on the respective parent table. Any time an insert or update is performed on the parent table, a new history record is produced (Figure 3).

The history tables constitute a transaction history of change to each table, a fact which became important to us when considering how updates to a record can be replicated back to the central data warehouse. Instead of replicating a change to a record back to the central data warehouse, such as an update to the parent table, we could replicate the new inserted history records and reconcile all the information centrally in an intelligent manner.

In our design, we would always have at least one record in a history table. If a record is created on the smart client, then the trigger would fire and produce a history record. If a record was replicated centrally to the smart client, the trigger would fire, also ensuring that a history record would be present.

In order to take advantage of history records as transactions, and to support the concept of logical records where a record is represented across multiple tables, we needed to bind history records across history tables. This requirement resulted in the concept of logical transactions.

Logical Transactions

A logical transaction is a representation of an attendee across all tables at a moment in time. What was missing from the history tables was a way to associate them together. When a change occurred in a parent table, the trigger will fire to populate the respective history table. This trigger would now need to be modified to perform the following functions (Figure 4):

- Make a copy of the parent table and insert it into the respective history table (as before)
- Obtain a new transaction ID that was date and time stamped
- For every history table that represents the logical record, update the last history record inserted with the new transaction ID

The third point is important in that even though we made a change to a single table, and only one new history record was produced, in that moment of time all of the other tables that represent the logical record have been validated. The functionality of obtaining a new transaction ID and updating all the latest respective history records is encapsulated in an SQL user-defined function and inserted into each trigger.

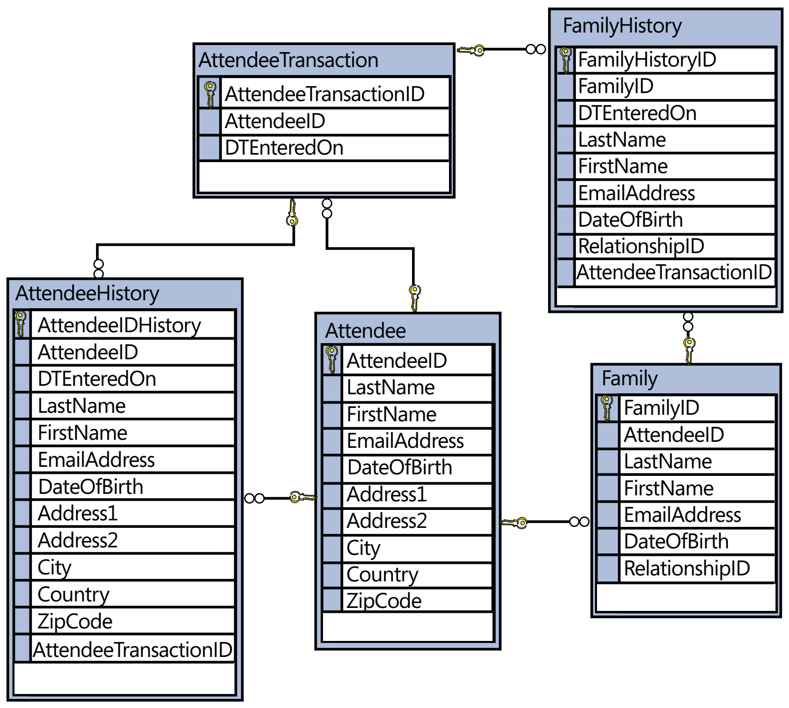
Logical transactions solve a key problem we had when dealing with information being updated on multiple devices. This problem stems from the complexity of synchronization multiple devices at different times for the same information. Consider this scenario:

Device A updates Attendee X on Monday, and device B updates the exact same information for Attendee X on Tuesday. In this application, because the same information was updated, the information updated on device B is deemed more relevant than A because it was updated at a later time. However, device B is synchronized on Thursday and device A on Friday. In the Thursday evening conflict resolution processing, no updates from device A are known, so device B's updates are taken. In Friday evening's conflict resolution processing, device A's information is reconciled. Without any date/time stamping of the logical transaction, it is difficult to understand when the change occurred and whether or not device A's information is more or less relevant than device B's. Logical transactions provided a clean representation of the information for all data entities in the system.

Updated Records

When an SQL table record is updated, SQL Replication can be used to move those changes from one database to another. Sounds simple enough when stated this way, however, knowing if you should overwrite the changes you have with an update from another system is not so trivial. The last-one-in-wins rule does not always apply. There may be parts of the record you wish to keep rather than just overwriting it.

Figure 4: Logical transactions (AttendeeTransaction) keep history records related.



Refining how one requires updated records to flow within your application is important in simplifying as much as possible the overall design. In the case of disconnected mobile applications, the flow of information flows from a central server out and lives and grows temporarily on each disconnected device, and eventually flows back to the central server. When examining this flow we made some assumptions and requirements on how the event management application would operate.

Information from the central server only flowed once to the mobile device. For example, when an event was requested to be synchronized, and all the attendees' information was replicated down to the device, any updates made to the attendee information in the online Web application would not be replicated to the device. Even if we intended updates to be replicated, we could not rely on the users to re-synchronize their devices throughout the event. However, during the events, the attendees may update their information online. Our design allows multiple devices and the online system to update the same records and for reconciliation later by our .NET conflict resolution process. This is enabled by only replicating what has been inserted and not updated. Only SQL inserts (no updates) are communicated between systems. We can then pick a time of our choosing to reconcile all the information centrally.

In the event management application, after an event has been replicated to a device, a replicated attendee can be updated by the WPF application. When the attendee record is updated, a history record is produced. With the history record, a new logical transaction is created. Both of these are SQL inserts. What has been inserted is replicated back to the central server. The parent record itself is not replicated as this would overwrite the record on the server. The .NET conflict resolution process examines the logical transaction queue and determines new changes are present. It examines the history records that have been produced and determines the best approach to integrate those changes into the parent records.

Inserted Records

SQL inserts are much simpler to manage with SQL replication as there is, on the surface, no conflict resolution to handle. However, what happens when a new attendee is added to the disconnected device that already exists elsewhere? In our application, an attendee could have registered in the online application after the event information was replicated; at the event, the attendee could be added to the mobile application; and later, the attendee could be working with another user with their own instance of the application. Multiple instances of the same attendee record can easily be created.

We began to think of records that have been inserted on the client as not having the same rank as records inserted on the central server. When the inserted record on the client was replicated back to the server, the information contained in the record must be validated before becoming accessible to the application. To prevent these replicated records from being seen by the ASP.NET application, they would need to be tagged and excluded from SQL queries. The .NET conflict resolution process would examine these records, determine how to integrate the information, and decide whether or not to copy the newly inserted information into another record, retire the record, or make a new attendee available in the ASP.NET application (in our example).

For each SQL table that has records that can be replicated from the device to the central server, a new column was added to maintain the state of the record. Each record could have one of the following states: created on the server, not reconciled, merged, retired, or reconciled. If the record is created with the ASP.NET application, it would be tagged as "created on the server." This state tag assists with filtered replication of records from the server to the client. If a record is created on the client, then it is "not reconciled." When this record is replicated back to the server, then it is in the same "not reconciled" state, identifying the record to the .NET conflict resolution process as a record that needs to be processed. The ASP.NET Web application will ignore any records in the "not reconciled" state. Once processed, depending on the action taken by the conflict resolution process, the record's state would be changed to "merged," "retired," "reconciled," or "unknown."

Conflict Resolution with .NET

The requirements of the event management mobile application to manage conflict resolution go beyond what can be handled with out-of-the-box technologies. With logical transactions and state associated to inserted records, our goal was to get the data all in one spot in order to be able to make intelligent decisions about how to handle the information. Part of the puzzle is to determine when to actually make these intelligent decisions. Standard out-of-the-box technologies make decisions at the time the records are moved. In contrast, we needed to have a complete picture of all the inserted records (with parent tables and history tables) before determining how to reconcile the information. A business process was put in place to run the .NET conflict resolution process at 1 A.M. server time to reconcile the previous day's replicated information.

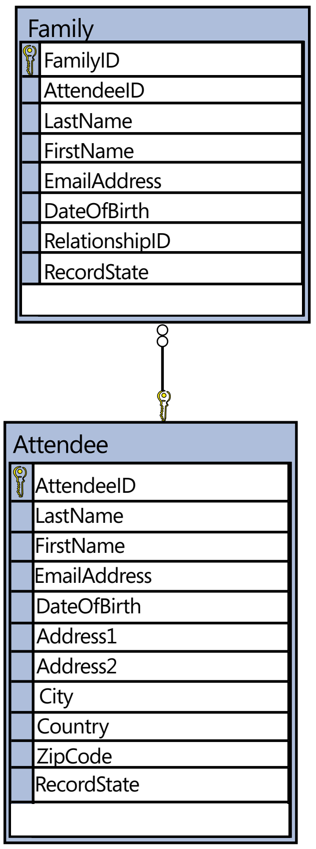
When the process begins, it obtains all logical transactions from the day that process last successfully executed to the day previous to the current date. An attendee may be represented by multiple logical transactions as the information could have been updated multiple times on one or many machines. For multiple updates for the same attendee on the same machine, only the last one entered is processed. From the remaining records, logical transactions are grouped by attendee and sorted by date and time entered.

As each logical transaction is processed, it is examined for whether or not the attendee record was originally replicated from the central server or entered on the client machine. If originally replicated, then the history records have already been replicated back and associated correctly. We are now free to implement as much detail as necessary in determine what business rules should be executed on the information. Each logical record referenced by the logical transaction has one or more tables associated to it. We can decide to simply overwrite more recent information across all tables, or one or more of the tables, or specific columns within any of the tables. This all depends on the business rules and can be as complex as necessary.

If a logical transaction is associated to an attendee record that was created on the client, then a determination must be made as to whether the attendee already exists in the system. At this point in time, this record is not viewable by the ASP.NET application as it is in a "not reconciled" state. A composite key is constructed from the attendee records to uniquely identify them. The existing attendees are examined to see if there is a match to the new record. One of three pathways is taken at this point: the new attendee record already exists, is brand new, or this is not enough information to make a decision. If the attendee already exists, then all of the history records associated to the attendee record in the "not reconciled" state are updated to point to the pre-existing attendee record. The same rules then apply as if the pre-existing record had been updated. If the attendee record definitely is new, then its state is changed to "reconciled" and it is made available to the ASP.NET application. If no determination can be made, then the record's state is changed to "unknown" and an exception management Web application is run to determine what to do with the record.

This process is repeated for each logical transaction. All the work that has been done was in preparation for this process to simplify it as much as possible and to centralize all complexity into a single spot.

Figure 5: RecordState column added to remember where the record came from and how it was processed.



Conclusion

Mobile applications are challenging to build. There is no magic framework we can all use to enable disconnected scenarios. We have powerful plumbing, but without customization, it only solves the simplest scenarios. The approach described in this article is generic enough to be applied to across different disconnected applications. Relying on the power of SQL Replication to perform only inserts allowed us to focus conflict resolution in the .NET process. Although much data work had to be done to augment the database schema to support it, we believe it simplified, as much as possible, the conflict resolution process.

This approach is not new, and it has limitations that make it appropriate for only a subset of mobile applications. Many mobile scenarios, for example, require real-time resolution of data as soon as it has been replicated to the central server compared to the many hours of delay proposed in this solution. Unidirectional replication of only inserted records may not be feasible, as our choice here was to centralize when and where we perform conflict resolution. Data loss due to conflict resolution has also been relegated to manual intervention in this solution. We took advantage of the requirement to retain and replicate all changes to records centrally, which became the transactions we based our conflict resolution process on. If you have no such requirement, then a burden is placed on replicating more data than you may prefer or be able to do. Ultimately, the business process driving your solution will dictate how cavalier you can be with your choices.

Rodney Guzman is the CTO and cofounder of InterKnowlogy. He got his start in software systems working on submarine sonars during college. For seven years at SAIC, Rodney was the lead developer and architect on such projects as a large Java SOA HTTP/XML based Web portal on military hospitals throughout the country. In 1998 Rodney moved to Stellcom to work on more Microsoft projects, including Site Server implementations and an enterprise security framework for Pacific Life, that allowed custom policies derived from AD groups and attributes to drive personalization and security on ASP Web sites. At InterKnowlogy, Rodney steers the technology direction, acting as lead architect on its largest projects, such as a large SOA implementation with a smart client framework, creating large Microsoft Web properties (CommNet and Partner Campaign Builder), and large MOSS implementations. Rodney architected the WPF/MOSS Scripps Cancer application. Rodney has spoken in numerous Microsoft events and has written numerous articles. He has sat on the Commerce PAC and Microsoft Architectural Advisory Board, and is a Solution Architect MVP.

Test-Driven Development and Continuous Integration for Mobile Applications

by Munjal Budhabhatti

Summary

This article demonstrates how test-driven development and continuous integration addresses the unique challenges encountered when creating Windows Mobile applications.

Current State of Mobile Development: Issues and Challenges

Globally, the number of mobile phone subscribers is approximately 2.5 billion and is expected to grow to 4 billion by 2010. The mobile device is now a richer platform for application delivery due to such an exponential growth and wide spread usage. The critical factor, as always, is the end user experience: application usability, reliability, and performance.

Complicating matters, the software development world is moving from weekly and monthly deployment cycles to continuous deployment. So how can one ensure that a user always has the best experience?

Many that have looked at the agile space will be familiar with two of the core extreme programming practices: The driving of development with automated tests, a style of development introduced by Kent Beck called test-driven development; and a software development practice of frequently integrating builds called continuous integration, as articulated by Matthew Foemmel and Martin Fowler.

These practices are not new to the software world. However, mobile application development has lagged in taking advantage of the test-driven development and continuous integration endowed by the enterprise software community. This is partially a result of limited or unavailable mobile platform support in existing toolsets such as NUnit/MSTest or Cruisecontrol.net/Team Foundation Server.

A few mobile testing tools allow recording user interactions via a graphical representation of the client device but do not provide granular control over the tests. Other tools either demand scripting on a mobile device or expect tests to be executed, manually, on the device. As a result, mobile application testing is inefficient and complex, hindering productivity.

Test-Driven Development

Test-driven development (TDD) is an evolutionary approach where the development of code is driven by first writing an automated test case, followed by writing the code to fulfill the test and then refactoring.

TDD essentially is test-first development + refactoring

Red/Green/Refactor—the TDD mantra—is an order to the task of programming:

1) *Red:* Write a small automated unit test that doesn't pass, and perhaps doesn't even compile at first. (See Figure 1, page 32.)
2) *Green:* Write the code necessary to pass the failing test. Ensure other tests pass as well, if present, in the suite. Check-in the code in the source code repository. (See Figure 2, page 32.)
3) *Refactor:* Making existing code beautiful, in small incremental steps, without changing the intent. (See Figure 3, page 33.)

This technique is thus reverse to traditional programming—developing code followed by writing a test, which is executed either manually or automatically. Why embrace such a change, especially when one might tend to think that it's extra work? In reality, test-driven development is risk-averse programming, investing work in the near term to avoid failures (and even more work) in the late term—Kent Beck has called it "a way of managing fear during programming."

"TEST-DRIVEN DEVELOPMENT IS A WAY OF MANAGING FEAR DURING PROGRAMMING—FEAR NOT IN A BAD WAY—BUT FEAR IN THE LEGITIMATE. IF PAIN IS NATURE'S WAY OF SAYING 'STOP!' THEN FEAR IS NATURE'S WAY OF SAYING 'BE CAREFUL.'"

—KENT BECK

The benefits of TDD

- *Design improvement.* Writing a self-contained test case enforces the creation of decoupled code—not tightly integrated with other code—thereby increasing the cohesion of the code while decreasing the coupling.
- *Documentation.* A well-written unit test case provides a working specification and communicates the intent of the code clearly. In addition, whenever the code changes, the unit test case must be updated to pass the test suite. Hence, a unit test case always stays in sync with the code naturally. This is unlike traditional unit test cases, developed with Microsoft PowerPoint or Microsoft Word. While

Figure 1: Sample test cases

```
namespace OutlookContacts.Tests
{
    //Ensure that OutlookContact compares correctly with other contacts.
    [TestFixture]
    public class OutlookContactTest
    {
        [Test]
        public void EnsureNullContactIsNotEqualToContact()
        {
            OutlookContact contact = new OutlookContact();
            Assert.AreNotEqual(null, contact);
        }
      [Test]
        public void EnsureContactsWithSameNameAreEqual()
        {
            OutlookContact contactMain = new OutlookContact(“foo”);
            OutlookContact contactOther = new OutlookContact(“foo”);
         Assert.AreEqual(contactMain, contactOther);
  }
     }
}
```

such documents start off with good intentions, over time, the result often becomes out of sync with the underlying implementation.

- *Safe change in the system*. TDD provides continuous feedback about whether the changes made in the code worked well with the other parts of the system.
- *Fail fast*. Unit testing can isolate problems quickly, reducing debugging activities and allowing the system to fail fast.
- *Beautiful code*. Beautiful code means code which expresses intent clearly, can afford changes to add features, and has no duplication. Refactoring retains the behavioral semantics of the code—functionality is neither added nor removed. It is about improving the code quality which brings business value.

Automated unit test execution is one of the vital requirements of TDD. However, because the testing tools are still evolving, the automated execution is not currently viable in mobile application development. Implementing TDD in this environment is therefore quite challenging, if not impossible.

Unit testing

Testing is customarily thought to be as a methodical process of proving the existence or lack of faults in a system. When a test case is written before writing the code, the test case becomes a specification, instead of a mere verification of the feature.

Tests are also a way of documenting *found* defects. Let’s assume a defect was discovered in quality assurance while testing newly deployed bits. Even if this defect was very trivial to fix, TDD demands a test case. First, write a test case that simulates such a failing behavior, and then write code to pass the test. Such a practice would ensure that defects, no matter how petty, do not creep through the system and regression testing becomes part of the test suite. Automated test execution locally, before committing the changes to the Source Control Repository (SCR), would further reduce the *broken builds* phenomenon.

It is important to prepare the test environment on an emulator as close to the target hardware as possible. Developing and testing Windows Mobile applications on an x86 emulator makes little sense when targeting hardware exclusively for ARM architecture, an architecture dominant in low power consumption electronics. Furthermore, in the real world, components often have dependencies on other objects, databases, or network connections. It is very easy to fall into a trap of *assuming* that these dependencies work flawlessly. Hence if tests are written without taking dependencies into consideration, an incorrect feedback is possible for those tests which fail due to dependency problems.

One way to safeguard against dependencies is to build the object graph or set up the database in a required state before executing the test case. This would solve the issue but would increase test execution time and build time.

A more elegant approach would be to instantiate test objects and replace object dependency by implementing mocks or stubs—objects that imitate the behavior of real objects. This ensures isolated test execution and hence reliable test results. Caution should be taken while faking the real objects with mocks or stubs. It is probable that the entire unit test suite executes faultlessly, yet the product might fail in quality assurance testing. I have found in my own experience that complementing mocks and stubs objects with integration tests provides a true sense of confidence.

Figure 2: Sample code

```
namespace OutlookContacts
{
    public class OutlookContact : Contact
    {
        public override bool Equals(object other)
        {
            if (other == null) return false;
            OutlookContact otherContract = (OutlookContact)other;
            return otherContract.AccountName == AccountName;
        }
    }
}
```

Continuous Integration

Martin Fowler has described continuous integration (CI) as a software development practice of frequently integrating builds, often multiple times a day. A typical CI workflow, as shown in Figure 4, would be:

Developer:

1) Writes a new unit test case.
2) Executes the unit test case locally on the emulator and confirms that the test case is failing.
3) Adds or modifies code to pass the test case.
4) Executes the unit test case locally on the emulator and confirms that the test case is passing.
5) Commits the code to the SCR.

CI Server:

6) Downloads the source code whenever there is a change in the SCR.
7) Compiles and inspects the source code, and creates new binaries.
8) Sets up external dependencies such as database schemas, and resources (configuration files, satellite assemblies).
9) Deploys the new binaries to and executes the tests on the emulator.
10) Packages and deploys the product to staging environment.
11) Generates feedback based on the results of the build.

"CONTINUOUS INTEGRATION IS A SOFTWARE DEVELOPMENT PRACTICE WHERE MEMBERS OF A TEAM INTEGRATE THEIR WORK FREQUENTLY—USUALLY EACH PERSON INTEGRATES AT LEAST DAILY—LEADING TO MULTIPLE INTEGRATIONS PER DAY. EACH INTEGRATION IS VERIFIED BY AN AUTOMATED BUILD (INCLUDING TEST) TO DETECT INTEGRATION ERRORS AS QUICKLY AS POSSIBLE. MANY TEAMS FIND THAT THIS APPROACH LEADS TO SIGNIFICANTLY REDUCED INTEGRATION PROBLEMS AND ALLOWS A TEAM TO DEVELOP COHESIVE SOFTWARE MORE RAPIDLY."

—MARTIN FOWLER

Source Code Repository

All the essential files required to build a product reside in the source code repository (SCR). It plays an important role in the software development life cycle and CI. SCR tools such as Subversion and Visual Studio Team Foundation source control enable teams to work collaboratively—on the same or different artifacts simultaneously, track code changes effortlessly, and work on different versions of files concurrently.

The CI server obtains the latest source code from SCR, locates all required dependencies, and builds the product independently from the previous build output. SCR allows the team to be more productive—a new team member does not need to reconfigure third-party libraries, project structures, or IDE settings for the project. Moreover, it reduces debugging time by allowing the team to remove the current changes which would be small and incremental if using TDD practices. The system could be safely reverted to a previous version of the code.

It is vital to include all dependencies in the SCR: This includes the Windows Mobile SDK, .Net Compact framework installer, Virtual Machine Network service drivers and other third-party components and utilities.

Automated build

Contradictory to some misconceptions, the automated build in mobile applications is much more than a simple code compilation. It assembles source code from the SCR, compiles code to create binaries and dependencies (such as configuration objects, resource assemblies, and so forth), inspects and deploys compiled binaries to a mobile device or an emulator, loads database schemas, and executes tests remotely on the mobile device.

Build tools such as Nant and MSBuild do not support mobile application deployment, partially because mobile device support in these tools is still immature. In addition, unit testing tools such as NUnit and MSTest have a similar problem of executing tests remotely on a mobile device. To avoid these limitations a new tool is essential.

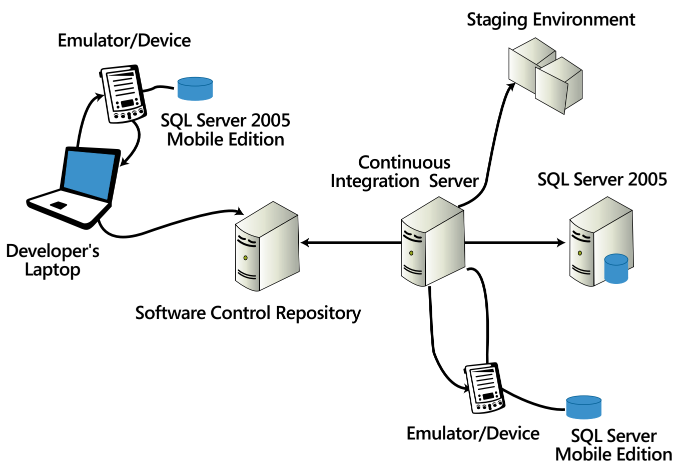
wMobinium.net

To address these problems, I created a tool *wMobinium.net* which assists TDD and CI implementation in .Net mobile applications. wMobinium.net is a unit testing tool that supports automated deployment and automated remote unit test execution. It has a Visual Studio add-in to support TDD in .Net

Figure 3: Sample refactoring

```
namespace OutlookContacts
{
    public class OutlookContact : Contact
    {
        public override bool Equals(object other)
        {
            if (other == null) return false;
            return CheckEqualityForOutlookContactObject(other);
        }
        private bool CheckEqualityForOutlookContactObject(object other)
        {
            OutlookContact otherContract = (OutlookContact) other;
            return otherContract.AccountName == AccountName;
        }
    }
}
```

Figure 4: Continuous integration in .Net Windows Mobile application


Emulator/Device
SQL Server 2005 Mobile Edition
Developer's Laptop
Software Control Repository
Continuous Integration Server
Staging Environment
SQL Server 2005
Emulator/Device
SQL Server Mobile Edition

mobile applications. It is a freely available tool on the CodePlex open source Website (see Resources).

Automated deployment

Unlike desktop application development, mobile application development faces unique challenges in deployment: first, a deployment is required for unit testing features on the device and second, deployment is necessary after a successful build to deliver the working solution on a staging environment for quality assurance testing.

For every build, the newly compiled binaries and dependencies must be copied to a program folder on a device. When an application entails testing on multiple devices, there will be added complications to the deployment process. To circumvent these problems, wMobinium.net offers a deployment tool which implements Window CE's Remote Application Programming Interface (RAPI), and facilitates file and folder deployments. This relieves the pain of manual deployments.

Frequent commits

One of the key components for a successful CI implementation is frequent commits and hence frequent builds. With longer commit intervals, team members tend to work in isolation and the build is more prone to integration issues. When a team member spends more time on a feature and encounters integration problems while committing the code, he tends to be reluctant to remove the changes and revert to a previous version of the code. This reluctance might increase the amount of time and resources spent on debugging activities. In an ideal scenario, team members check-in the code at intervals of 30-60 minutes or less. The check-in duration could be extended by a few hours, but should always be less than a day.

Faster Successful Builds

Once a developer commits code to the SCR, waiting for the feedback from CI slows down the development process. Longer waits result in decreased productivity. Furthermore, some of the subprocesses of the build, such as deployment and testing, are executed on a device/emulator making these processes inherently slower than it would have been on a desktop.

To reduce build times, concentrate on the weakest link—the component that takes the longest time to execute. More often than not, the cause would be an external dependency, such as a database or other objects. Accessing a database and setting up test data for each test case is a resource-intensive operation. As I mentioned earlier, mocks or stubs, should do the trick. If it is impractical to do so, move the test cases to secondary or nightly builds—scheduled builds that execute at night when most of the resources are idle. Test cases targeting tests scenarios on multiple devices should be added to secondary or nightly builds as well.

Failing builds cause the most frustration—as if the entire chain of software churning has been stopped. The code in the SCR is no more reliable and the team is blocked from getting the latest source code. The team needs to resolve the issue quickly by fixing the build. If a test case is causing the failure and fixing the problem might require a longer duration, it is safe to ignore the test case temporarily to allow the build to succeed. However, it is vital to track these ignored test cases on an easily accessible project wall, a physical white board or a virtual board using collaborative software. The ignored tests should be fixed at a later time and added back to the test suite.

Automated unit testing

It is quite common to see that the same defects resurface after a few builds and we often hear a quality assurance analyst say, "but this defect was already fixed in a previous build." Boomerang defects reveal the importance of writing a test for each encountered defect before modifying the code base. Once the test case is fixed, the entire test suite must be executed and passed before checking in the modified code to the SCR.

I have been on a few projects where the team executes the test suite manually on a mobile device. Imagine a mobile application developer who spends few minutes changing the functionality, but spends double the time to test it manually. This will not only discourage a developer but will also affect productivity.

wMobinium.net resolves this annoyance by automating the entire unit testing workflow. Unlike traditional unit testing, wMobinium.net presents the test case selection on the desktop, executes tests on the device, and displays results on the desktop. It takes care of some of the complications such as the following:

Remote execution of test cases

In order to execute the test cases remotely on a mobile device/emulator, the tool serializes metadata information of the selected test cases, starts a conduit process and executes tests on the device. To provide correct reporting to the CI server, the remote process must be started synchronously and monitored continuously, which is quite a challenge.

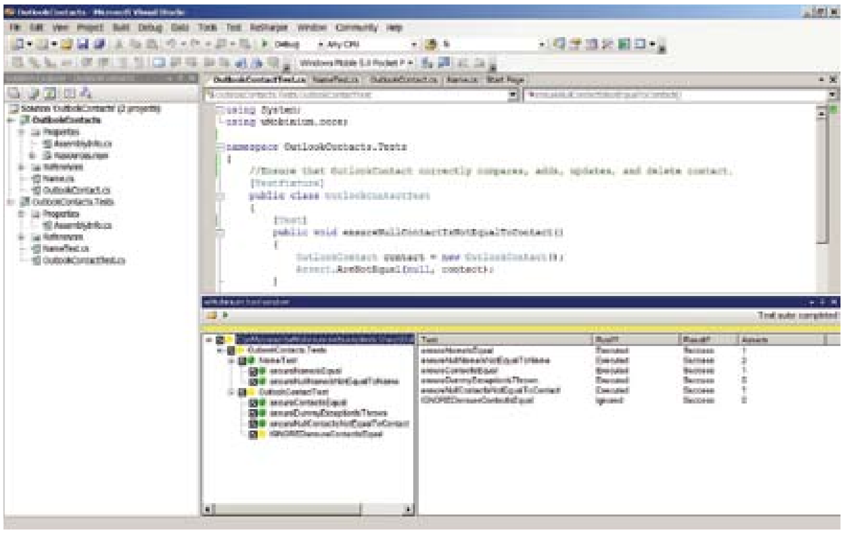
Serializing test results to desktop

In the absence of support for remoting in the .Net Compact Framework version 2.0, the device must communicate with the desktop using sockets. The events must be serialized, sent to the desktop through a socket, deserialized, and propagated to the appropriate event listeners.

wMobinium.net add-in

The tools described here assist the CI server to continuously build, deploy, and test a .Net mobile application. It would be convenient if the unit testing feature was supported as an integrated tool in Visual Studio.

Figure 5: wMobinium.net add-in



```
using System;
using wMobinium.core;

namespace OutlookContacts.Tests
{
    //Ensure that OutlookContact correctly compares, adds, updates, and delete contact.
    [TestFixture]
    public class OutlookContactTest
    {
        [Test]
        public void ensureNullContactIsNotEqualToContact()
        {
            OutlookContact contact = new OutlookContact();
            Assert.AreNotEqual(null, contact);
        }
```

```
Test                                    Result    Result    Asserts
ensureNamesEqual                        Executed  Success   1
ensureNullNamesIsNotEqualToName         Executed  Success   2
ensureContactsEqual                     Executed  Success   1
ensureDummyExceptionIsThrown            Executed  Success   0
ensureNullContactIsNotEqualToContact    Executed  Success   1
IGNOREDensureContactsEqual              Ignored   Success   0
```

wMobinium.net add-in, a Visual Studio add-in (Figure 5), is a part of the wMobinium.net toolset. After activating the add-in, all the available tests in the solution are displayed in the tool window. A typical workflow would be:

1) wMobinium.net tool displays available test cases in a solution.
2) User selects test cases to execute.
3) Selected test cases are serialized and sent to the connected device/emulator.
4) Test cases are executed on the device/emulator and results are sent back to the desktop.
5) Tool displays results on the desktop.

Benefits of Using CI

Stakeholders and project sponsors always favor reliable outcomes, clear communication, project visibility, and superior quality of software. Software development, however, seldom offers such qualities without the right processes and practices.

When everything seems to be going well, any defect might suddently jeopardize the development schedule. Especially during "Big-bang" integrations, even small issues—like missing configuration entries, out-of-sync database, or missing dependencies—could be extremely detrimental when encountered together.

Continuous integration enables faster feedback. At every change—adding new or modifying existing features, no matter how big or small—the CI server would integrate the new parts which would pass through the entire automatic build cycle—compilation, testing, inspection, and deployment. This provides visibility of the progress of the project, enhances the quality of the software developed, and builds the morale of the team.

CI does not provide these functionalities out-of-the-box. It is very possible to implement CI without including automated tests or inspection in the builds process; however, such a setup would be the least beneficial. Many, including me, consider that CI without testing is not CI at all.

Conclusion

From a user perspective, TDD and CI implementation is the same in a traditional desktop application as it is in a mobile application. The user creates a new failing automated test case, writes the code to pass the test, and refactors the code without changing the intent. The CI server polls for the latest source code creates new binaries, executes the tests, and generates the feedback. However, in a mobile application the implementation differs in the remote execution of test cases, notification of test results, and build deployments. These complexities are handled by the wMobinium tool.

In my past experience at one of the biggest microfinance organizations in Africa, my team and I developed a .Net mobile application. In the absence of supporting tools, the development implementing TDD and CI, although arduous, improved overall application design, reliability, and performance.

With the release of .Net Compact Framework version 2.0, the performance of .Net mobile applications has improved radically. The newer version provides improved developer productivity, greater compatibility with the full .Net framework, and increased support for device debugging. Combining .Net Compact Framework with TDD and CI (using wMobinium.net) would bring greater benefits to an organization and take the mobile application platform to the next level.

With the proliferation of new mobile devices today, the mobile application is becoming a crucial part of a broader enterprise product offering. It is pragmatic, more than ever, to bring mobile application development out from isolation and include it in enterprise-wide test-driven development and continuous integration efforts.

Resources

"Continuous Integration," Martin Fowler and Matthew Foemmel
http://www.martinfowler.com/articles/continuousIntegration.html

Continuous Integration: Improved Software Quality and Reducing Risk, Paul Duvall, Steve Matyas, and Andrew Glover (Addison-Wesley Professional, 2007)

Test-Driven Development: By Example, Kent Beck (Addison-Wesley Professional, 2002)

wMobinium.net
http://www.codeplex.com/wMobinium

About the Author

Munjal Budhabhatti is a senior solution developer at ThoughtWorks. He possesses over 10 years of experience in designing large-scale enterprise applications and has implemented innovative solutions for some of the largest microfinance, insurance and financial organizations in Africa, Asia, Europe, and North America. He spends most of his time writing well-designed enterprise applications using agile processes.

Case Study: Support Technicians on the Road

by András Velvárt and Peter Smulovics

Summary
Monicomp is the largest maintainer of banking equipment in Hungary. The organization installs, maintains, and repairs Point of Service (POS) terminals, Automatic Teller Machines (ATM) machines, and other similar banking equipment using hundreds of technicians scattered all around the country. Any company that needs to perform these duties and has to follow ISO 9000 needs all the IT help it can get. The organization has to track where every meter of a two kilometer cable has been used, so that if there's a problem with the cable, it can go back to each and every shop that the cable was used in, and perform repairs before more problems arise. In addition, clients require up-to-the-minute information on the Web—which can range from how the country-wide software updates are going right down to the faulty POS terminal in Mr. Smith's pet shop.

As you may expect, a paper-based way of working can't keep up with these requirements. It would be unrealistic for technicians to go out to the field, and send paper worksheets in for processing at the end of the work week. You need a 21st century solution. The "MÛVÉSZ" system, developed by a software development and consulting company called Response, covers the full range of the service operation from inventory to billing; from trouble ticketing to distribution and verification of work for more than 100 technicians performing the maintenance and installation tasks. In the article, we will show how the architecture of this system is designed and describe the challenges and solutions of creating mobile subsystems.

Worksheet Workflow

Before we look at the architectural overview for the application, let's begin with an overview of the important workflow processes in the organization. For Monicomp, the most important entity in a mobile workflow is the *Worksheet*. The worksheet is a work item that needs to be performed by a field technician. Let's examine the key concepts of a simplified workflow:

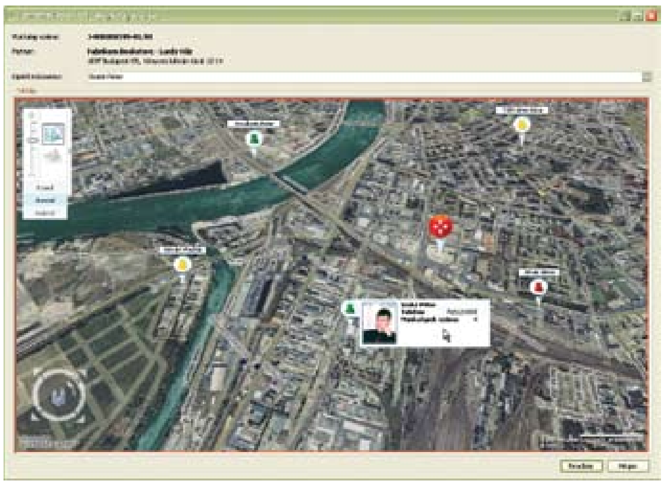
Creating the worksheet. The worksheet is usually created by the dispatcher. When a worksheet is created, it has the following key attributes:

- Type of worksheet which categorizes the work to be performed: installation, repair, or uninstallation
- A reference to the partner or address where the work is to be performed
- Information from the original trouble ticket that the dispatcher recorded (if any)
- Identifiers and additional data of the devices to be installed or uninstalled
- The identifier of the worksheet itself
- Deadline of the work

Assigning the worksheet. To start the workflow, the dispatcher assigns a technician to the worksheet after reviewing the workload and the geographical position of the available technicians. To make this easier, technicians' geographical positions are gathered via Global Positioning System (GPS) tracking devices and displayed on an integrated Virtual Earth map, with color-coded icons indicating their current workloads. (See Figure 1.)

Downloading the worksheet. After the worksheet has been assigned to the technician, a Short Message Service (SMS) message is sent. The SMS notifies the technician and the Ultra-Mobile PC (UMPC) device that there is new work to be done. With a synchronization through General Packet

Figure 1: Using Virtual Earth Services to select the technician closest to the target with the smallest workload

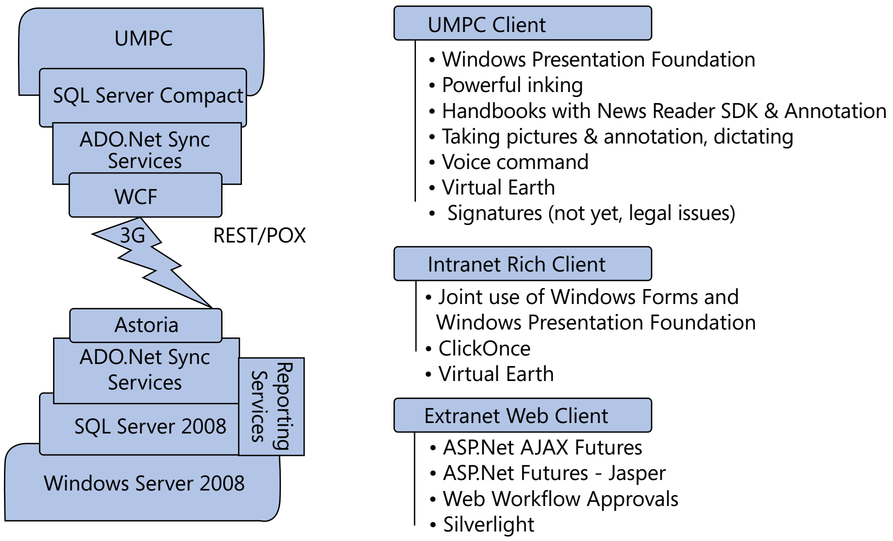


Radio Service (GPRS), Edge, 3G, or a wireless network (depending on the availability in the technician's area), the new worksheet is downloaded to the Worksheet application running on the UMPC.

Filling out the worksheet. The technician reviews the received worksheet, travels to the location where the work is to be performed, and does his job. During the repair, installation, maintenance, or uninstallation work, he records the details of the device he's working on (such as serial number), and assigns equipment and materials from his inventory to the device. He can also use the Worksheet application to review technical specifications of the device in question if the repair turns out to be difficult. These specifications can also be annotated with Ink, and these annotations are shared in a knowledgebase.

Other data to be recorded includes the time of arrival to the scene, the number of hours it took for the technician or technicians to perform the job, and travel distance to and from the location. You can learn more details about the Worksheet application in the "User Interface with WPF" section later in this article.

Figure 2: Overview of the main components and their communication links



Closing the worksheet. After the worksheet is filled, it is printed on a mobile printer, and signed by the client. The signature could also be captured on the touch-sensitive display of the UMPC using Ink (so that technicians would not have to carry printers with them), however, due to legal issues, this functionality has not yet been used.

Uploading the worksheet. The database on the UMPC is synchronized, uploading the changes on the worksheet to the central server. The business logic on the server registers the work undertaken, removes the used-up materials from the technician's inventory and performs all other necessary housekeeping. The synchronization mechanism is also very handy to create up-to-date backups of the technicians' databases. In case a UMPC is broken or lost, the technician can quickly resume his work because his database can be recreated as it was at the time of his last synchronization.

Two Types of Synchronization
The UMPC clients can perform two types of synchronization. To save bandwidth and battery life, the quick synchronization only moves the data directly related to the technician's worksheets. The full synchronization (docking) downloads a complete set of data to the UMPC (for example, changes in the people force or software updates); therefore it is usually only performed via Internet connection at the technician's home.

Architectural Overview—What and Why?
The architecture, whether it's defining connections between black and white boxes, displaying characters in a command line, calling Web services, or using an Office Business Application, is by nature determined by the type of quality of the connection. Mobile computing, despite today's always-connected ideology, in reality is occasionally-connected. Given this, how do you plan an architecture for occasionally-connected scenarios?

In this section, we summarize some key issues that we confronted in planning and implementing this mobile, high-complexity architecture. To start, let's look at Figure 2, which shows the main components and communication links of our mobile architecture.

Occasionally Connected
The major consideration for an occasionally-connected architecture is synchronization, be it one way, two way, dealing with conflict resolution logic, and presenting this through the user interface. Not dealing with these issues can cause consistency problems; solving the problems makes the business logic much more complicated than a pure offline or online scenario. In order to decrease the time to market we selected the ADO.NET Synchronization Services, which were introduced along with other parts of the toolset to create a fully automated, less error prone, and standardized solution.

Key System Components
At the heart of the system is a multilayered application server, which performs data storage and business logic functionality. Several different types of clients utilize its services:

- A Windows Forms application that is used at the headquarters by the storekeepers, the dispatchers, and technicians performing repair jobs that cannot be done at the locations. The application is deployed using ClickOnce, which helps ensure seamless version upgrades.
- An extranet Web application with fluid and responsive interface used by the biggest clients to place hundreds of orders in a single batch, prioritize, and track the progress of these orders.
- A Windows Presentation Foundation (WPF) application running on hundreds of UMPCs used by technicians who perform installation, repair, and maintenance tasks throughout the country.

The Server Configuration
As a backend server, we found the new Windows Server 2008 to be an ideal solution for Software + Services scenarios like this one—primarily because it is capable of hosting Windows Communication Foundation (WCF) services to serve and queue HTTP requests at the kernel level, while at the same time being easy to manage from an infrastructure perspective.

We used an instance of SQL Server 2008 running on the server to provide capabilities like spatial data storage, and to serve as the data store as a reporting solution for the dispatcher application and the external website. The previously mentioned synchronization problems were tackled by the ADO.NET Synchronization Services, along with Microsoft's "Database in the Cloud" solution called Astoria, which you can read about in Issue 13 of the Architecture Journal. Using Astoria we created a WCF service and through the service, using Representational State Transfer (REST) and Plain Old XML (POX) protocols, read and modify the underlying database. To establish the needed business logic in the appropriate places, we found that the synchronization services, Astoria, and database triggers worked together well.

Dispatcher, Storage, and Billing Application
To aid the work of storekeepers, group leaders, and dispatchers at the headquarters, an easy-to-use application was needed to have all required information instantly available. This Line-of-Business (LOB) application integrates with positioning systems (using spatial data storage), creating a good user experience (using a mixture of WPF and Windows Forms), and displaying maps (using Virtual Earth Live Services).

External Web Site
As this was a new solution, we had some flexibility to look at technologies without incurring great risk of having to maintain any existing applications. We implemented a custom workflow step to be able to approve, reject, or modify records that were visible to the end customer, primarily to avoid displaying stale data. The solution for this was derived from the "Windows Workflow Foundation Web Workflow Approvals Starter Kit", which we were able to customize with little effort. Another opportunity for widening the technology horizon was fast prototyping offered by the "Jasper" scaffolding solution that came with ASP.NET Futures: starting with a simple interface, we found that it was easy to go into deep customization without losing the investment in the original forms. The ASP.NET Ajax Futures' user-friendly back button and history support made interface changes fluid.

UMPC Application
We found that the application created for the UMPC device was our opportunity to add the most innovated thinking. The UMPC device and the Worksheet application perform many different tasks:

- Recording the work performed by the technician, thus filling out the worksheet
- Communicating with the server over 3G, Edge, or WiFi, using ADO.NET Synchronization Services and a local SQL Server 2008 Compact Edition instance for data storage
- Voice recording and camera for documentation, with additional annotation via Ink
- Displaying XPS format handbooks using the Microsoft News Reader SDK for the ATMs/POSs, featuring annotation and collaboration capability, with voice-, soft-button-, or stylus-based navigation
- Showing map and route information for the next target to visit.

What is the infrastructure under this? From the developer side, it was really a matter of putting well-defined building blocks together, such as Enterprise Library, WPF, ADO.NET Synchronization Services, SQL Server 2008 Compact Edition, WPF, and the .NET Framework. Since the UMPC is a full-featured PC with an x86 compatible processor, running familiar operating systems like Windows XP Tablet PC Edition or Windows Vista, we found that the actual development process is much like a normal desktop application development. From the designer and software ergonomics side however, we did find development a bit more complex. We explore this in the next section.

User Interface with WPF

Moving Away from a Traditional User Interface
Windows Presentation Foundation gives UI designers the chance to reconsider old habits of UI design and offers the opportunity for some out-of-the box, lateral thinking.

Had we taken a traditional view of the UI, we would have probably enumerated the functions of the software, created groups, and then

Figure 3: The user is taken immediately to the most used part of the application.

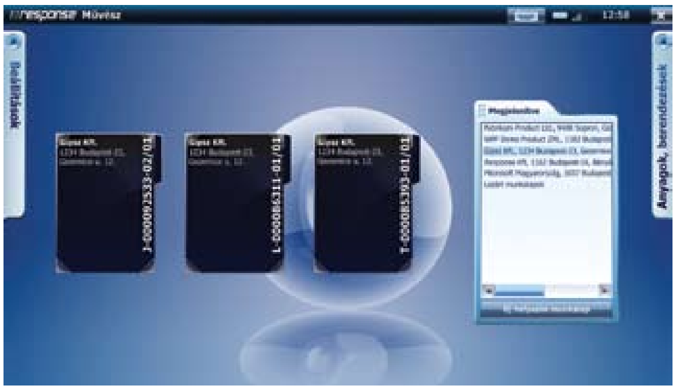


Figure 4: Opening the folder reveals further details and access to editing its contents.

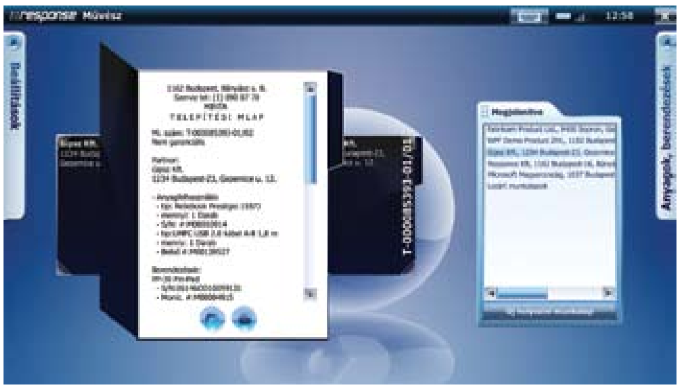


Figure 5: The inventory is always accessible on the right side.

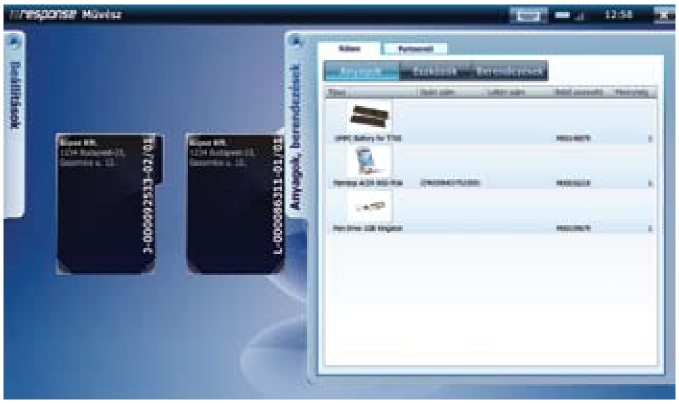
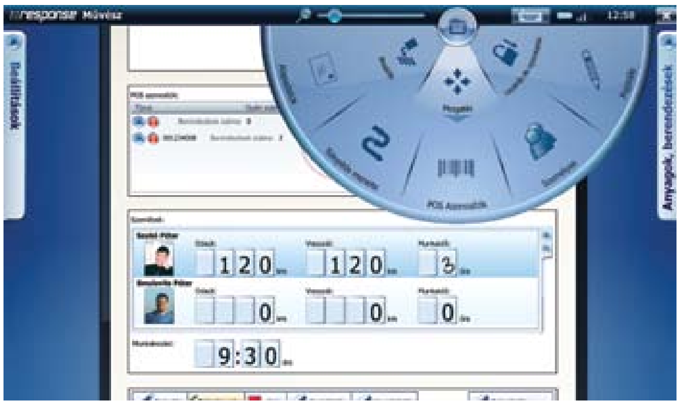


Figure 6: Pie menus are ideal for taking advantage of spatial memory. Huge sensitive areas help with finger interaction.



displayed a main menu after the program had started, most likely with the most frequently used options first. We decided instead however to eliminate the main menu altogether—the software starts in the most used view, with the worksheet overview screen. The additional functionality of synchronization, version upgrades, reference repository, and access to the inventory of the user were implemented as complementary, slide-out panels.

On the worksheet overview screen, worksheets are visualized as folders. The folders already show the most important data on the outside (Figure 3), but further details are displayed when you open a folder by tapping it on the touch screen (Figure 4). We also placed the "Edit Worksheet" button inside the folder, thus ensuring that no worksheet is opened for editing before the user has a chance to look at the details.

Taking Advantage of Spatial Memory

According to Wikipedia, "Spatial memory is the part of memory responsible for recording information about one's environment and its spatial orientation." For example, spatial memory helps you by remembering that you have last seen the pen you are looking for on the right side of the table.

If you take advantage of spatial memory, and always put the same thing on the same side of the screen, your users will instinctively remember where to find it. To implement this in our case, the bottom slide-out panel contains the reference materials, and the right side slide-out panel contains the inventory—all the equipment they have in the backs of their trucks, ready to be used when installing or repairing devices (Figure 5).

The inventory panel has two functions. Initially, the technician checks whether he has the replacement parts with him. Later, when filling out the worksheet, he records the parts he has actually used for the repair by dragging and dropping items from the inventory onto the worksheet. In both cases, the inventory can be found on the right side, accessed, filtered, and sorted in the same way. We have chosen the right side because, during user testing, we found that most of the technicians are right-handed, thus they hold the UMPC in their left hand, and the pen in their right hand. With this being the case, the drag-and-drop operation required to add an item from the inventory onto the worksheet is more convenient.

We chose to include a pie menu on the worksheet editor screen (Figure 6). We found that first time users quickly understand how the menu works with the help of subtle, but fast animations when opening the menu, submenu or selecting a menu item. The large sensitive areas of the menu help when the menu is used with a finger—your finger is not very precise compared to a mouse or a stylus, but it is always at hand. The pie menu also takes advantage of the user's spatial memory, since it is much easier to remember that the "Close worksheet" icon is on the right side, than to remember that it is the third item in a dropdown menu.

In real life, placement of items is a way of organizing them and we wanted to take this concept across to the user interface. For example, you may have the habit of keeping important documents on the left side of your desk, and less important ones on the right. We have allowed the same for our technicians in the worksheet overview: the folders can be moved, rotated, and even resized (Figure 7).

Natural Interaction

However accustomed you or I may be to the mouse, it is not a natural way of interaction. You have to move an odd-shaped object on your

Figure 7: Customizable placement of folders allows spatial grouping to suit the habits of the individual user.

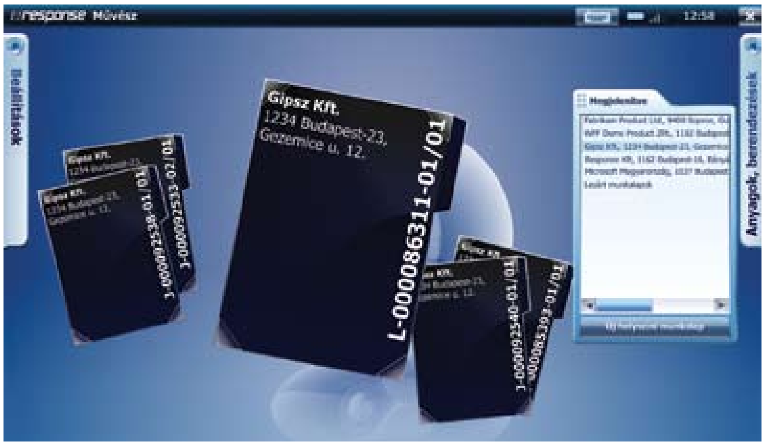
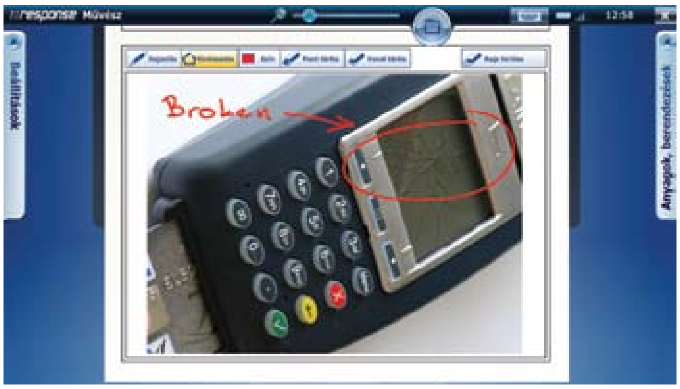


Figure 8: On the drawing canvas, the technician can record any type of graphical information.



desk, while looking elsewhere for feedback to see whether your cursor has reached the intended destination. When it comes to interactions like point-and-click, drag-and-drop, or drawing, using a touch screen is much more intuitive and productive, even for seasoned mouse-users. With this said however, a finger or even a pen or stylus tends to be a lot less precise than a mouse. To accommodate this when we designed the user interface, we were keen to keep the buttons large and to make sure they don't touch each other. For the same reason, when using a scrollbar, we had to make sure that it was much wider than usual.

During our user testing we found that a scrollbar is something that you may not want to have at all in a touch-screen environment. If you think about it, using a scrollbar is somewhat unnatural — you have to press a down arrow or drag a small thumb downwards, when you want the scrolled content to move upwards. When you are scrolling a document, it is actually moving in the opposite direction than your mouse or scroll-wheel. A much more natural way of scrolling is to drag the content itself in the direction you want it to move. Leaving behind this unnatural way of scrolling is part of what makes the most recent generation of mobile devices so enjoyable to use.

Using Ink for Input

When it comes to recording information, we found that nothing beats the versatility of pen and paper from a usability standpoint. Using ink gives the technicians a way to draw diagrams, write text, or annotate part of a document freely. In the digital world, Tablet PCs, UMPCs, and Pocket PCs with touch screens can work similarly. We found that using digital ink was useful for creating drawings, annotations, and recording the customer's signature confirming that the work was performed.

Using the Ink API in Windows Presentation Foundation (in the System.Windows.Ink namespace) made using digital ink a breeze in the above scenarios. In just a couple of hours, we were able to add the drawing canvas with the ability to store, edit, and resize the ink strokes—with most of the time spent on creating the icons that switch between drawing modes. (Figure 8 shows an example of an annotated picture.) Text recognition (or Ink Analysis) is also very straightforward, and works for many languages, including English (UK or US), French, Spanish, Korean, German, Japanese or Chinese.

Without using Ink Analysis, the way the technician is filling out the form would be fairly cumbersome without a mouse and a keyboard. A good example is the field where they record how many hours they spent with the repairs. We found that the user had to point (click) at the proper textbox, select its contents if there is any, click to pop up the on-screen-keyboard, enter the number of hours and close the keyboard. That's five clicks, and a lot of context change in the user's mind ("I want to enter a number. I need to select where I want it. I need to bring up a virtual keyboard. I need to use the keyboard, close the keyboard, return to the main task of filling out the worksheet"). Compare this with the act of writing the number "3" in the proper field with the pen (Figure 6), and you will appreciate the natural simplicity of this new-old way of interaction.

Conclusion

Creating occasionally-connected mobile applications comes with many challenges. Luckily, hardware and software technologies that make such an application feasible are starting to become more mainstream—UMPCs with the hardware capabilities of PCs from a few years back, GPS trackers, mobile Internet connectivity, sophisticated software libraries, ink and speech recognition built into the operating system, and multiple online services; the list goes on. With the system that we put together for Monicomp we find that it's all coming together; enabling developers to build mobile solutions that only sci-fi writers dared to dream about a decade ago.

About the Authors

András Velvárt approaches software development the Lincoln way: "If I have six hours to chop down a tree, I will use four hours to sharpen the axe." In the past, he was one of the few who helped spread the idea of the World Wide Web in Hungary. He has always enjoyed trying and implementing the newest technologies—many of his proof-of-concepts ended up being useful additions to projects, from the UI/usability side to the development process side. He has been working with Microsoft technologies for 12 years. He was Lead Developer, Architect, and Project Manager on many of Web, desktop and enterprise projects, until he founded his own software development and consulting company, Response. András can be reached at andras.velvart@response.hu.

Peter Smulovics is a Microsoft guy, searching for the needle in the haystack for more than 10 years. He worked on projects like SQL 2005 Analysis Services, Visual Studio.Net 2005, Microsoft Business Framework and Portal, ADO.Net Entity Framework, Dynamics Great Plains and Solomon, doing testing, developing, group leading, and architecting. Meanwhile, being part of user groups, presenting at Developer Days, TechEds and Microsoft Forums, leading .NET introduction in the country—these were all part of his daily job. Currently, he is working in the Development and Platform Adoption group and giving architectural aid and workshops for Enterprise customers on many different products. Peter can be reached at smulovics.peter@microsoft.com.

Disclaimer: Although the solution described in the article is based on the actual working system at Monicomp, due to the nature of the topic, some details have been changed to protect company interests, while others are currently in the planning or implementation phase.

ARC

Microsoft®

THE ARCHITECTURE JOURNAL™

Input for Better Outcomes

098-108819

Subscribe at: www.architecturejournal.net