Introducing: NET Micro Framework

Colin Miller, Director

Lorenzo Tessiore, Lead Developer

Microsoft, .NET Micro Framework

Microsoft .net Micro Framework

Agenda

- .NET Micro Framework Background
- How it fits in MS Embedded story
- Benefits of the Micro Framework
- Demos
- Architecture
- Runtime Features
- Runtime Implementation
- Roadmap

.NET Micro Framework History

- Roots in Microsoft Research
- Developed from scratch for inexpensive, consumer applications on very power limited devices
- Developed to provide a platform that could be extended by a large number of developers
- Shipped commercially on the SPOT watches in 2004
- Shipped with MSTV Set-top boxes
- SumoRobot Kits
- Will ship with Vista as the SideShow feature
- Continued development for internal products shipping next year

Convergence of trends in embedded

- Old trends – addressing productivity and scarcity of resources
  - Movement to standardized OS
  - Movement to higher level languages
- Newer trends
  - Movement of 32 bit processors into 8 & 16 bit space (ARM Cortex M3)
  - Proliferation of low power communication alternatives
    - Zigbee, BT, Z-Wave, ANT, low power WiFi,...
  - New network protocols
    - Mesh networks
- Enabling more rapid development of integrated embedded solutions

New Opportunities

- Industrial Automation
  - Corporate IT Servers to Shipping Palettes
- Home Automation
  - Presence and Home Dashboards
- Healthcare
  - Body Monitoring, Elder Care
- Retail
  - Card Readers, Point of Purchase Devices
- Remote Displays
  - Conference Messaging, Remote Controls

What is needed?

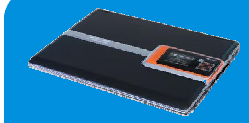
- Decrease development time
- World class tools
- Increase pool of developers
- Vertical integration story
- Easily updateable in the field
- Low cost
- Power efficient

- .NET Micro Framework
  - Based on .NET Framework
  - Embedded extensions
  - Visual Studio integration
  - Small memory footprint
  - Processors w/o MMU
  - Built from the ground up to be power efficient

Extending the MS Embedded Story

.NET Micro Framework


Wearable Devices


Auxiliary Displays



Health Monitoring


Remote Controls


Set-top boxes


Sensor Networks

Windows CE


Pocket PC Phone


Windows Mobile Smartphone


Windows Automotive


Portable Media Center


VoIP phones


Mobile handhelds


Gateways

Windows XP Embedded


Retail Point-of-Sale


Windows-based terminals


Medical devices


Entertainment devices

Selecting an Embedded Platform

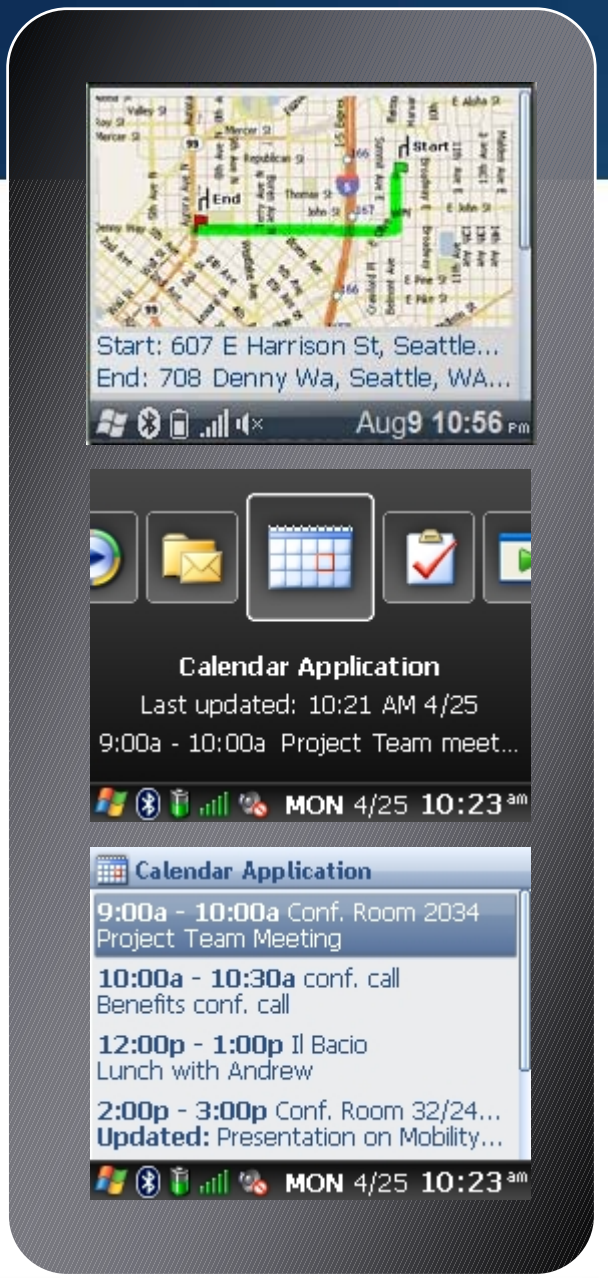
	.NET Micro Framework	Windows CE	Windows XPe
Example Devices	Sensor Nodes, Aux displays, Health Monitoring, Remote Controls, Robotics	GPS Handhelds, PDAs, Automotive, Set Top Boxes	Thin Clients, ATMs, Kiosks
Device Features	Connected, Small, Wearable, Graphical UI	Connected, Graphical UI, Server, Browser, RAS, DirectX	PC-class performance, PC networking
Footprint	250-500KB managed code Full featured	300KB+ without managed code 12MB with managed code	40MB + Depending on features
Power	Very low power	Low power	Mains power
CPU	ARM7, ARM9 No MMU	X86, MIPS, SH4, ARM, with MMU	X86
Real-time	Not Real-time	Hard Real-time	Real-time capable through 3rd party extensions
Managed vs native code	Managed via .NET Micro Framework, native code through interop only.	Supports both, managed code requires .NET Compact Framework	Supports both, managed code requires .NET Framework

The Microsoft .NET Micro Framework
Extending .NET to smaller devices

- Small .NET runtime for embedded devices
  - Managed Code reliability and productivity
    - No unsafe instructions
    - Memory Management/Garbage collection
    - Exception Handling
  - Lowest cost .NET platform
    - Memory footprint for the platform – 250K RAM
    - No MMU required
- Develop and debug in Visual Studio
  - Full-featured debugging on device
  - Familiar tools decreasing cost of resources and training
  - Increased productivity
- Use C#, a subset of .NET libraries, and WPF
  - Leverage code and data structures
  - Familiar coding decreasing cost of resources and training
  - Extensible Emulation

Runtime Features
UI/Shell

- Object model based on Windows Presentation Foundation (WPF)
- Input event routing
- Layout system
  - Content sizing
  - Text flow
  - Rich support for nested controls
- Bitmap fonts
- Images
- Pens, brushes, colors
- Vector primitives
- Alpha blending



Embedded Specific Features

- Power management
- Managed Code Drivers
- Validation in post-compilation
- Prioritized Persistence
- Customizable Bootloader with optional signatures

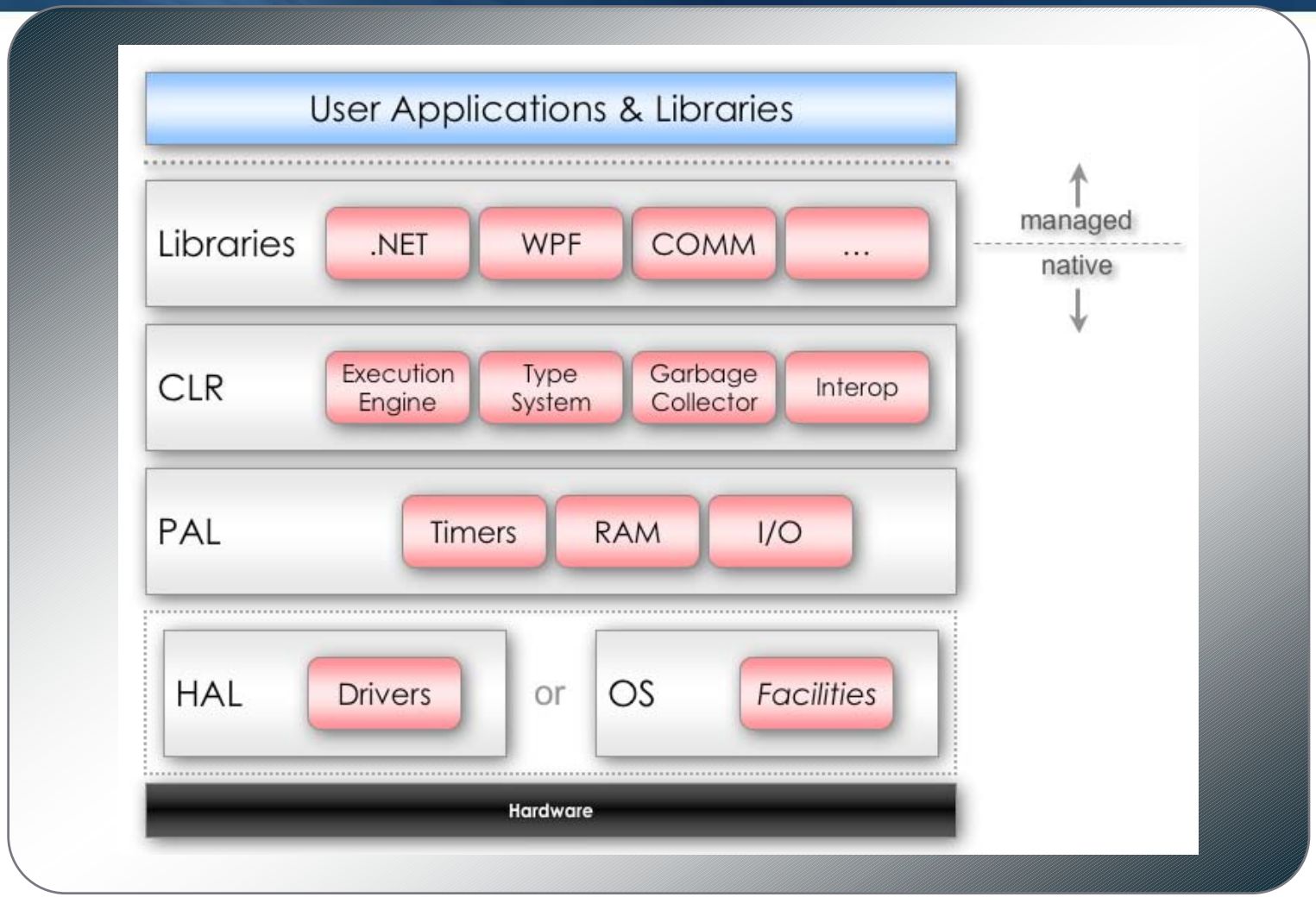
Tools

- Extensible Device Emulation
  - Runtime on x86
  - Define hardware in XML
    - RAM and Flash, clock speed, IO,LCD metrics, any public Property
  - Definition honored by the runtime
- Microsoft Visual Studio 2005
  - Project system/templates
  - Integrated Post-build processing
  - Intellisense support
  - Deploy to device (USB/serial) or emulator
  - Interactive debugging from IDE
    - Breakpoints
    - Variable inspection
    - Tracing

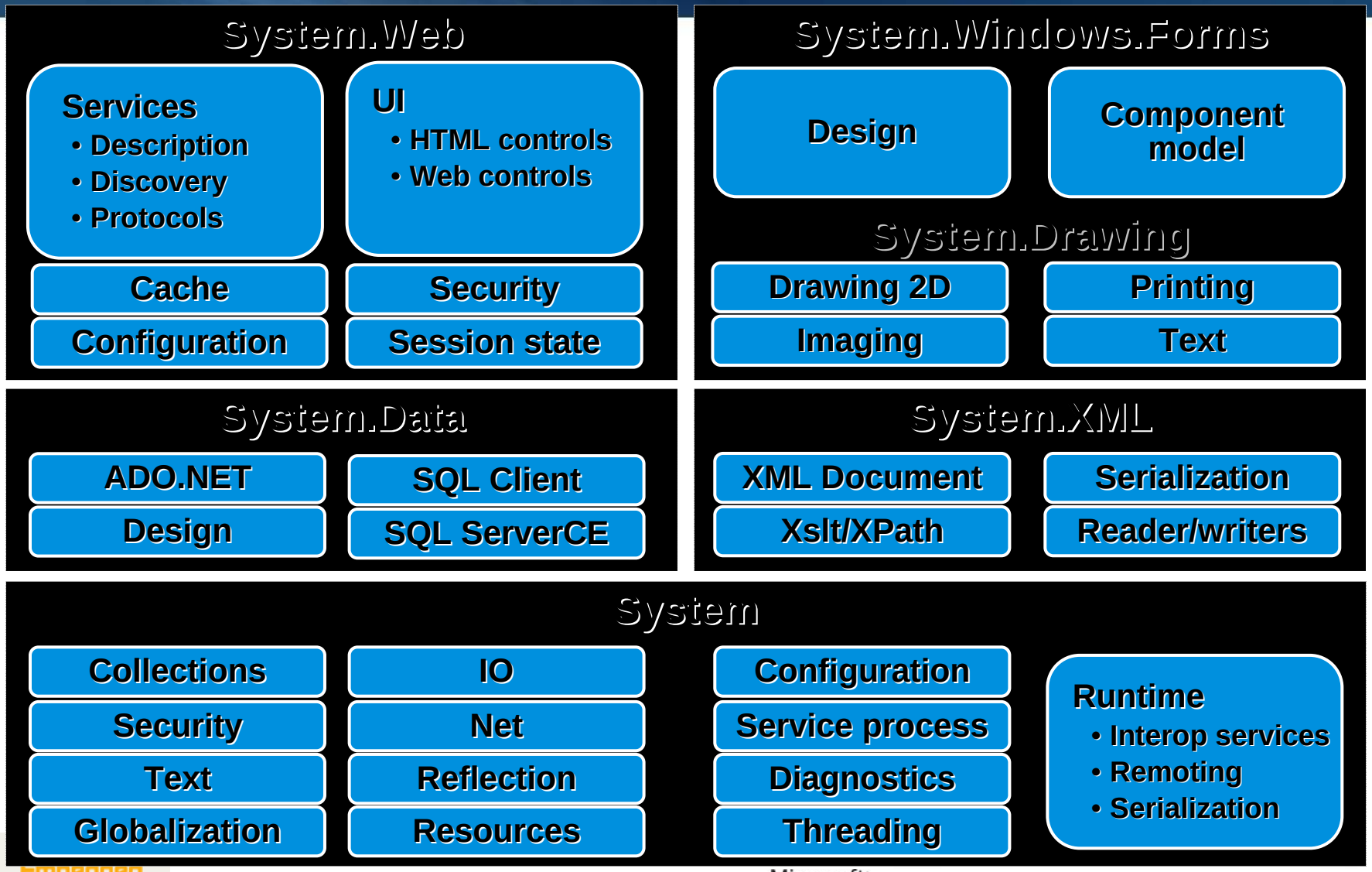
Architecture
Goals

- Bootable .NET
  - Minimal native code core
  - Application space entirely managed
- Safe
  - No direct access to hardware resources
  - Managed drivers to safe access
- Secure
  - Signed assemblies only
- Extensible
- Portable

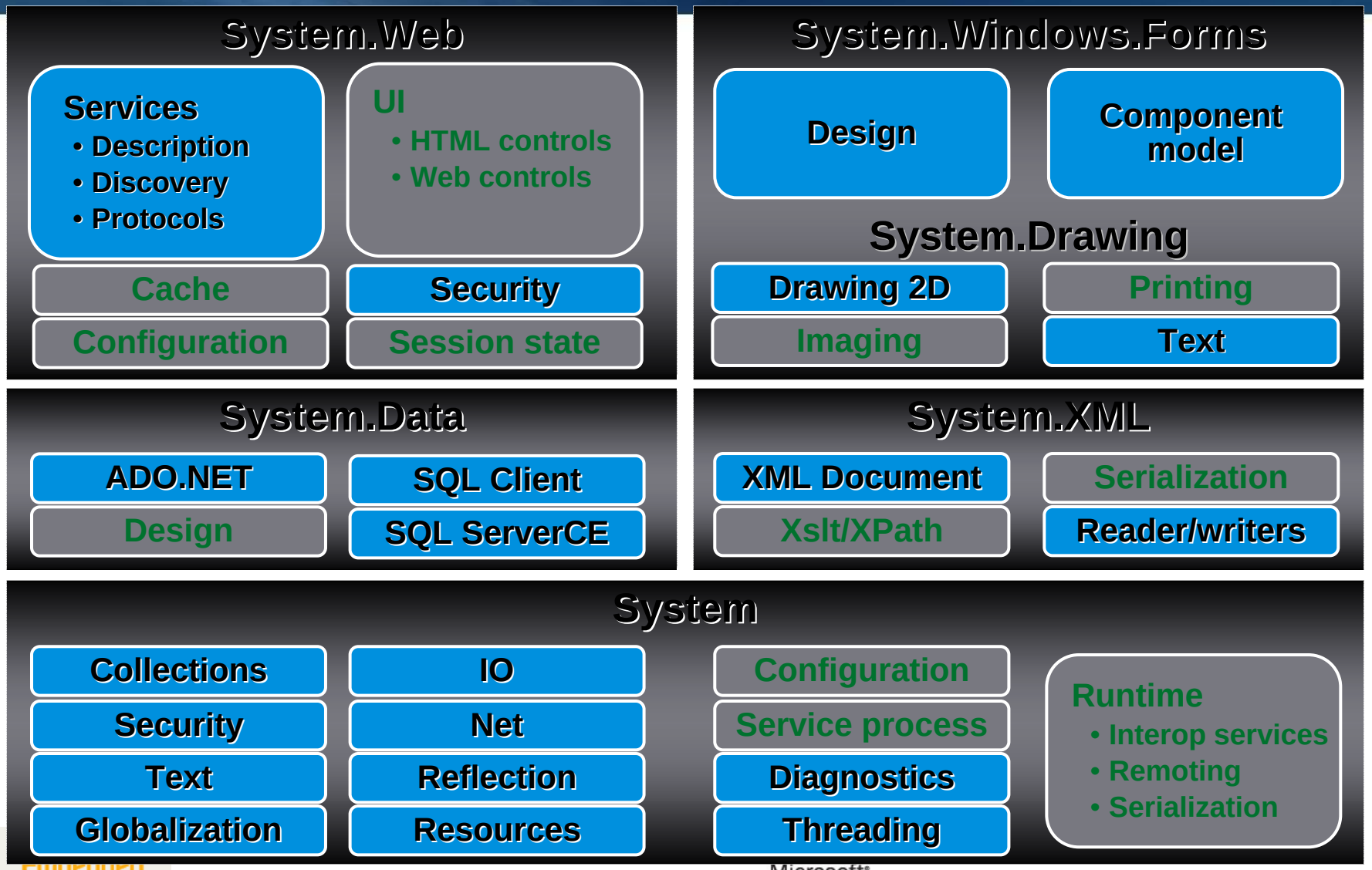
Architecture



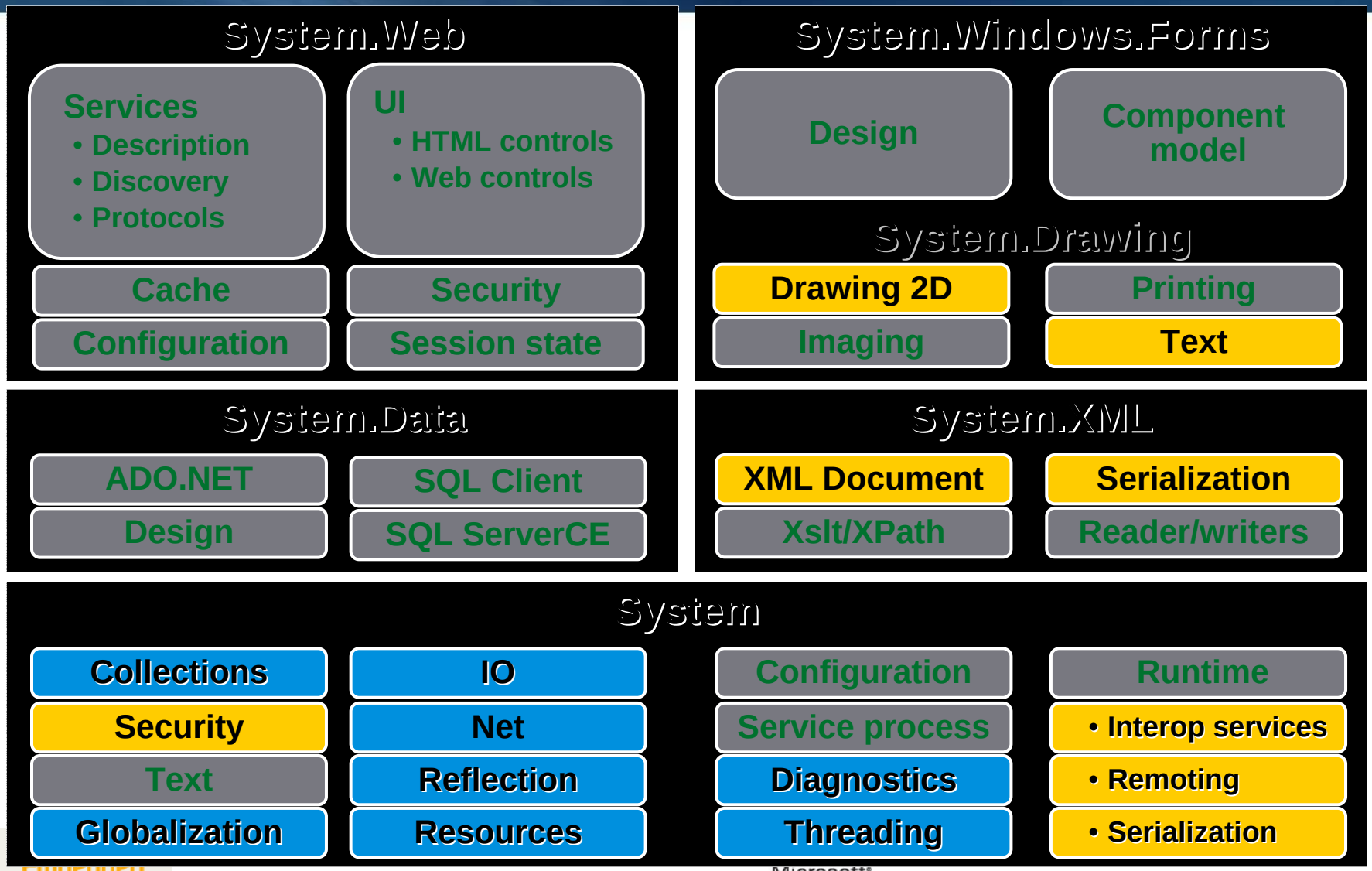
.NET Framework



.NET Compact Framework

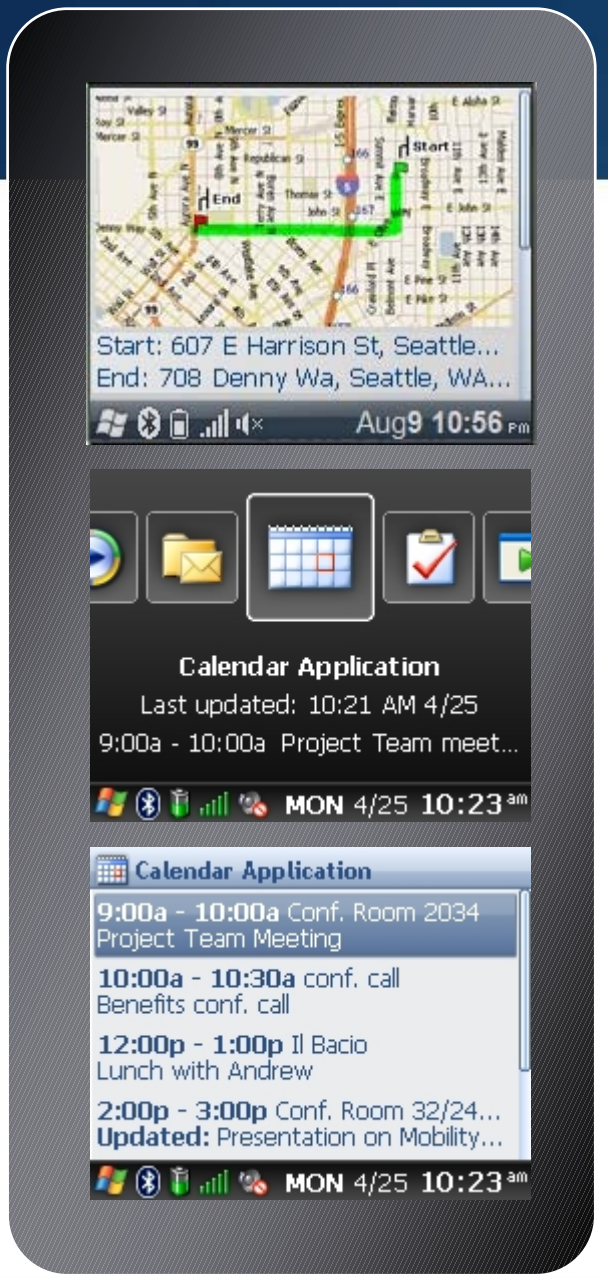


.NET Micro Framework



Runtime Features
UI/Shell

- Object model based on Windows Presentation Foundation (WPF)
- Input event routing
- Layout system
  - Content sizing
  - Text flow
  - Rich support for nested controls
- Bitmap fonts
- Images
- Pens, brushes, colors
- Vector primitives
- Alpha blending



Runtime Features

- Common Language Specification (CLS) compliant
  - Enables multiple languages to use the libraries
- Subset of core libraries
- Derived from Common Language Infrastructure (CLI) v1.0
- Supports Common Intermediate Language (CIL) execution

Runtime Features

- Multi-threading and synchronization

```
using System.Threading;
…
Thread myThread = new Thread(new ThreadStart(this.MyWorkerThread));
myThread.Priority = ThreadPriority.AboveNormal;
myThread.Start();
```

- Timers

```
using System.Threading;
…
Timer myTimer = new Timer(
   new TimerCallback(this.MyTimerCallback), null, 10, 100));
…
```

Runtime Features

- Serialization
  - ~60% smaller than .NET Framework
- Reflection
- Remote Procedure Call (RPC)
  - Invoke methods on remote objects
  - Implementation is specific to .NET Micro Framework
- Security
  - XTEA (symmetric)
  - RSA (asymmetric)
- Exception handling
- Delegates / multicast delegates
  - Typed Function pointers
  - Enables events dispatching and event driven programming vs. poll driven

Runtime Features

- Networking
  - Support for wired and wireless Ethernet
  - Sockets from System.Net namespace
- Managed Drivers
  - Direct control of GPIO, PWM, I^2C, SPI bus, and USART in C# code
  - GPIO interrupts safely dispatched into managed application space

Runtime Features

```
using System.Threading;
using Microsoft.SPOT.Hardware;

…
InterruptPort myButton = new InterruptPort(
                (Cpu.Pin)12,
                true,
                Port.ResistorMode.PullUp,
                Port.InterruptMode.InterruptEdgeLow
        );

MyButton.OnInterrupt +=
        new GPIOInterruptEventHandler( this.MyInterruptHandler );
…
}

public void MyInterruptHandler(Cpu.Pin id, bool edge, TimeSpan time)
{
    // handle the interrupt event here
}
```

Runtime Implementation

- Small footprint (ROM/FLASH)
  - Runtime only: ~120K
  - w/ min. framework: ~250K
  - w/ max. framework: ~500K
- Low RAM requirements
  - ~70K system overhead

Runtime Implementation

- Assembly loader
  - Performs minimal validation of post-processed assemblies
  - Registers types with type system
    - Integrated support in Visual Studio
- Microsoft Intermediate Language (MSIL) interpreter
  - Support for all MSIL instructions (except for four *unsafe* ones)
  - Low execution latency
  - Enables a pure virtual execution environment

Runtime Implementation

- Execution Engine
  - No native thread scheduler
  - Manages native *work queues*
  - Invokes native code
    - Non-traditional interoperability
- Garbage collector
  - Non-incremental mark-and-sweep
    - 50ms per 1M (depending on # of objects)
  - Support for non-volatile storage
    - Provides a basic object persistence mechanism

Runtime Implementation
HAL

- Bootstraps the runtime
- Provides low-level abstractions of hardware resources
- Handles interrupts
- Asynchronous cooperative multi-tasking
  - Queued work items
- Compact
  - ~40K (including base drivers)

Current Environment Support

- Currently supported chipsets (as a bootable runtime)
  - ARM7TDMI @ 27MHz (384K RAM, 1M Flash)
  - ARM7TDMI @ 50MHz (4M RAM, 2M Flash)
  - ARM920T @ 96MHz (4M RAM, 2M Flash)
  - ARM Cortex M3 (under development)
  - XScale in prototype
- Currently supported platforms (as a hosted runtime)
  - Dual ARM7TDMI @ 100MHz (32M RAM, 1G Flash)
  - Windows XP (x86)
  - Motorola 68k

Porting

- Porting Training
  - First session held July 2006
  - Combination of OEMs and ISVs
  - ISVs include:
    - 3SOFT
    - Weschler Software
    - SJJ/EMAC
    - Embedded Fusion
    - OpenNetCF
    - MCP Data
  - Additional Training to be scheduled
  - Porting Kit publicly available early 2007

Windows SideShow™

- Windows SideShow is a new technology available in Windows Vista™
- SideShow allows mini-applications (called gadgets) to send data to devices
  - Examples include: email, calendar, RSS, stock quotes, media player remote controls
- Open platform for ISVs
- Windows SideShow support is available on the .NET Microframework
- Consider Windows SideShow for your device if:
  - You want to receive information from the PC and render it
  - If you want to have 2-way control of applications on the PC (e.g. Windows Media Center, PowerPoint)

Roadmap

- SDK Beta 2 – now
  - Open Beta
  - Supports Sockets
  - Supports emulation on the PC
- SDK RTM later this year
- Add-ons starting Q1 2007
  - WSD
  - Generally available TCP/IP stack
  - more

Partners

Web Site: www.aboutNETMF.com

Microsoft .net Micro Framework