Windows Vista
Windows Phone 7

Softwarearchitekturen

Referent: Frank Urrigshardt

Übersicht

- Windows Vista
  - Historische Entwicklung
  - Programmierung
    - NT-Programmierschnittstelle
    - Win32-Programmierschnittstelle
  - Systemarchitektur
    - Betriebssystemstruktur
    - Hardware-Abstraktionsschicht
    - Kernschicht
    - Ausführungsschicht
- Windows Phone 7
  - Architektur

Softwarearchitektur von

WINDOWS VISTA

Historische Entwicklung

- Grobe Einteilung in 3 Abschnitte
  - MS-DOS (**M**icro**S**oft **D**isc **O**perating **S**ystem)
  - MS-DOS-basiertes Windows
  - NT-basiertes Windows

MS-DOS

- Ab 1981 vermarktet
- Einzelnutzer-Betriebssystem
- Kommandozeile als einzige Benutzerschnittstelle
- Letzte Version von 2000
  - DOS 8.0

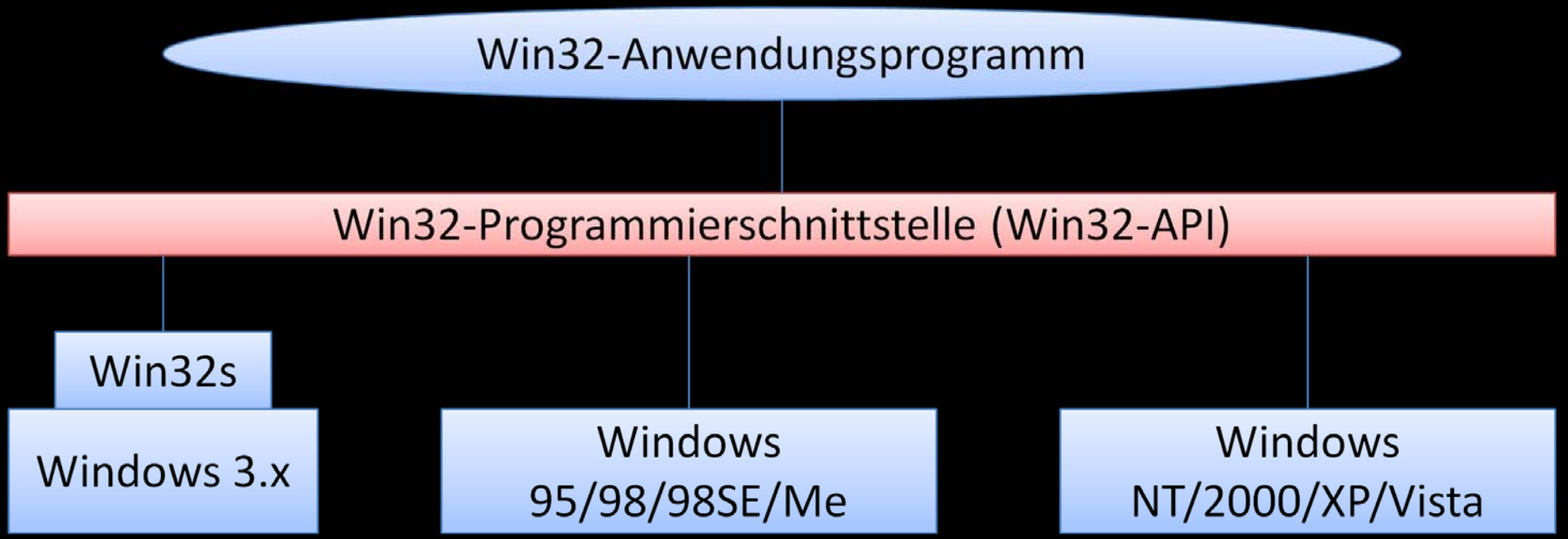
MS-DOS-basiertes Windows

- Lediglich grafische Benutzerschnittstelle auf 16 Bit MS-DOS aufgesetzt
- Alle Programme im selben Adressraum des Hauptspeicher
  - Fehler in Programm führt zu Absturz des Systems
- Windows 1.0 als erste Version, aber schlecht verkauft
  - Erst Windows 3.0 sehr erfolgreich

- Weitere MS-DOS-basierte Windows-Versionen
  - Windows 95/98/ME
  - Virtueller speicher, Prozessverwaltung, 32-Bit-Programmierschnittstelle (Win32-API)
  - Aber im Kern noch immer MS-DOS welches 16-Bit Assemblercode ausführt
  - Dadurch generelle Instabilität

NT-basiertes Windows

- **N**ew **T**echnology
- Entwicklung relativ früh parallel zu MS-DOS basiertem Windows
- Komplett neuer 32-Bit Kern der auf Sicherheit und Zuverlässigkeit ausgelegt ist
- Win32-API für maximale Kompatibilität
- Windows NT 3.1/2000/XP/Vista/7/...

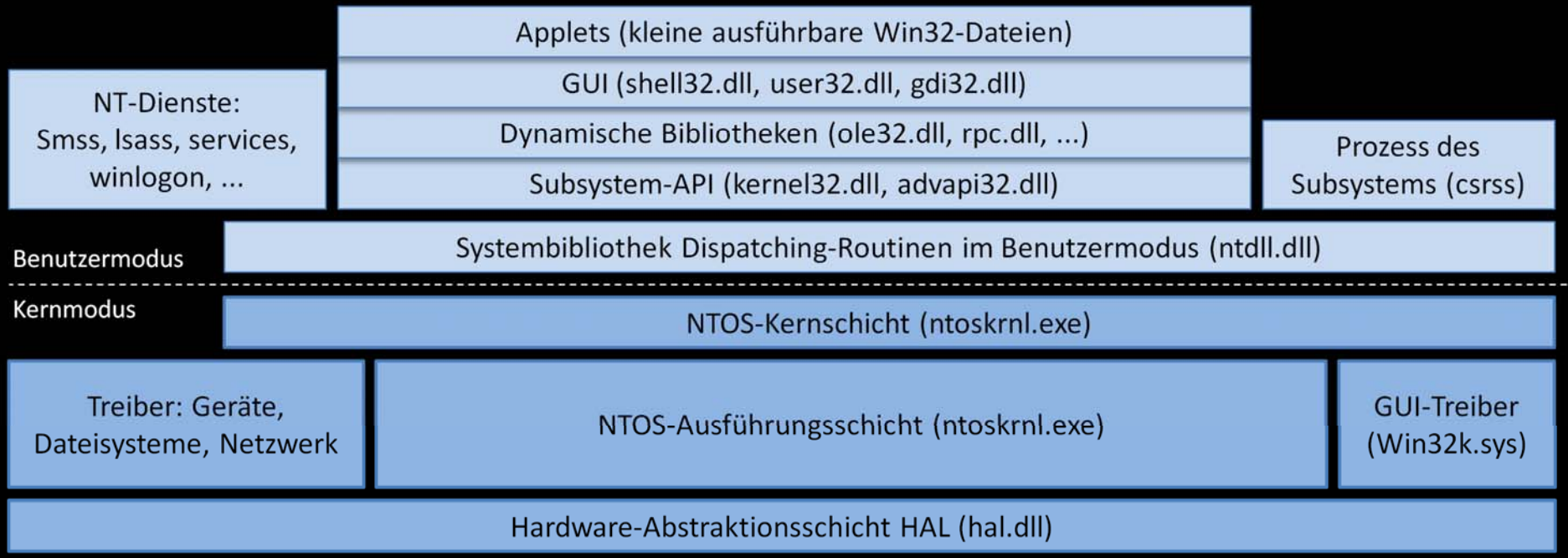


Win32-API

Anwendungsprogramme dank Win32-API sowohl unter MS-DOS - basierten Windows als auch unter NT-basierten Betriebssystemen nutzbar

Ein Überblick über die Programmierschichten von Windows Vista

PROGRAMMIERUNG



Programmierschichten

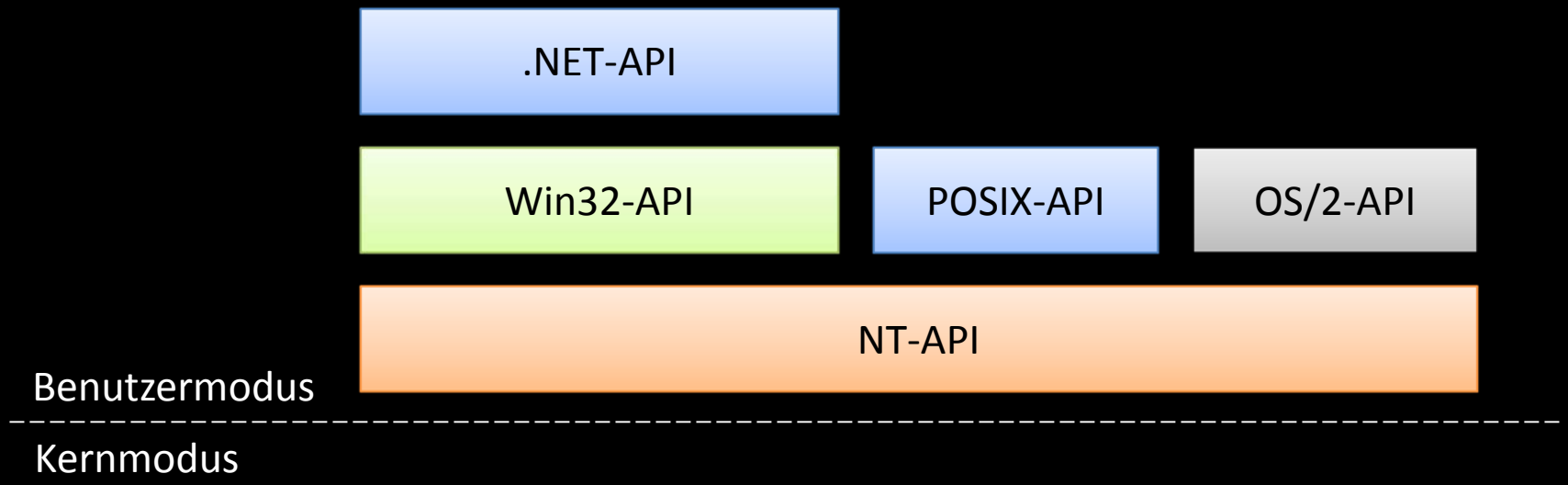
Fokus auf Benutzermodus, Kernmodus stark vereinfacht

- Ntoskrnl.exe Herzstück von NTOS
  - Stellt Schnittstellen für Systemaufrufe bereit
  - Kann nicht direkt von Anwendungsprogrammen genutzt werden
- APIs (*Personalities*) delegieren Aufrufe zum NTOS Kern
  - Win32-API
  - POSIX-API (*Interix,* kann nachinstalliert werden)
  - OS/2 (Ab Windows XP verworfen)

- .NET dagegen auf Win32-API aufgebaut
  - Fungiert in vielen Fällen nur als Wrapper
  - Aber
    - einfachere Schnittstellen
    - zusätzliche Objekttypen
    - automatische Speicherbereinigung durch .NET-Laufzeitumgebung

NT-Programmierschnittstelle

- Native NT-Programmierschnittstelle
  - Unterste Schicht im Benutzermodus
  - Schnittstellen werden in NTOS-Ausführungsschicht implementiert
- Warum nicht gleich für NT-API programmieren?
  - Um Kompatibilität zu MS-DOS-basiertem Windows zu gewährleisten



Struktur der APIs in Windows Vista

Win32-Programmierschnittstelle

- Vollständig dokumentiert
- Vielfach nur Wrapper um NT-API-Funktionen
  - Win32 Parameter werden für NT angepasst
    - Z.B. Pfadnamen
  - Win32: CreateProcess()
  - NT: NtCreateProcess()

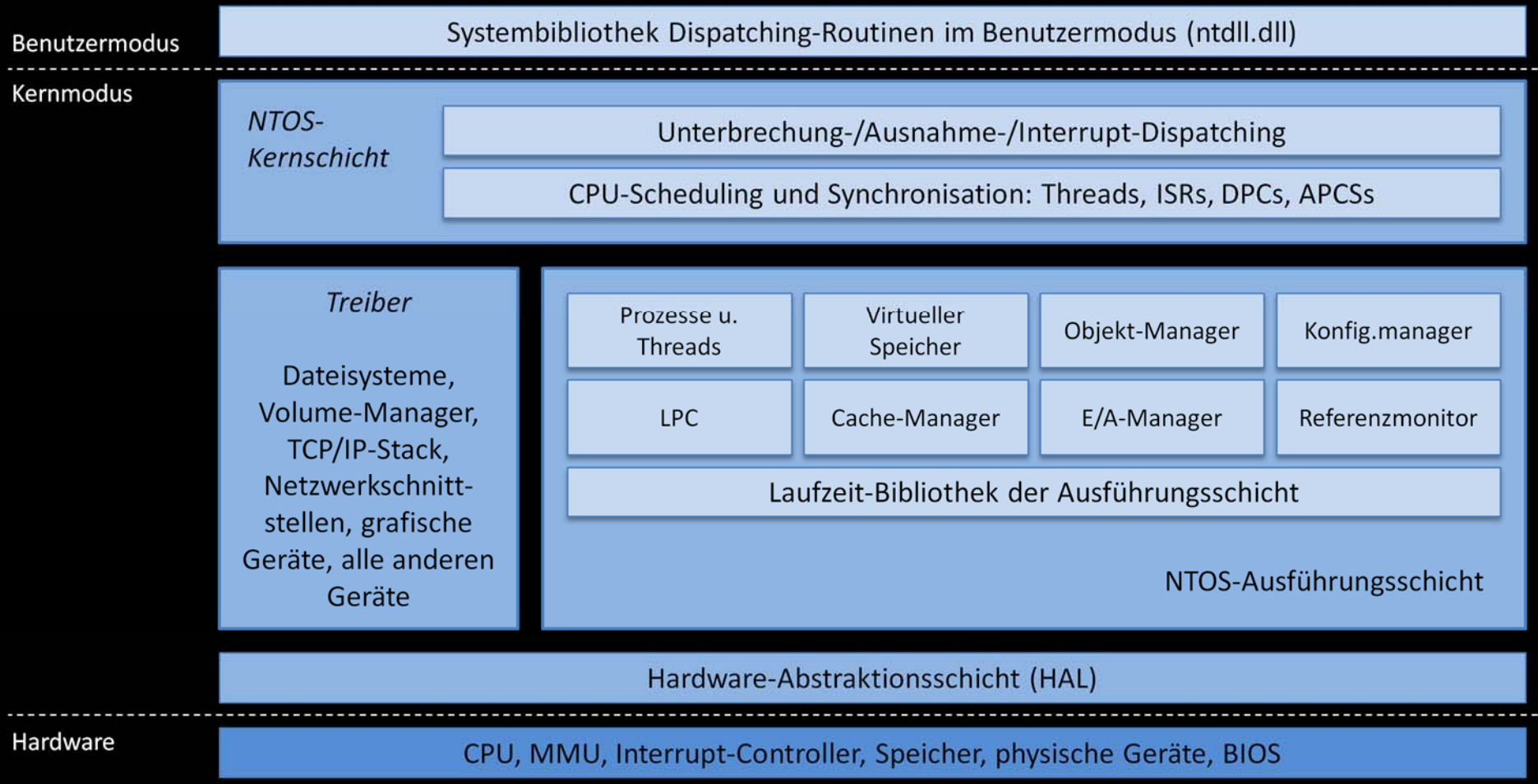
Bereitgestellte Funktionalitäten

- Ausgewählte Beispiele
  - Erzeugung und Verwaltung von Prozessen und Threads
  - Memory-Mapped-Dateien
    - Datei in virtuellem Speicher
    - Lesen und Schreiben ohne Festplattenzugriff
  - Demand-Paging
    - Änderungen an virtuellem Abbild je nach Bedarf auf Festplatte übertragen oder umgekehrt

- Über 60 Aufrufe für Dateiein- und ausgabe
  - Erstellen, Löschen, Dateiattribute setzen, etc.
- Durch NTFS wird Verschlüsselung ermöglicht
  - Ab Windows 7 kann Betriebssystempartition ebenfalls komplett verschlüsselt werden
- GUI-Operationen
  - Erzeugen, manipulieren, etc. von Fenstern, Scrollbalken, Icons und co.
  - Erzeugen geometrischer Objekte
  - Verwaltung von Farbpaletten
  - In win32k.sys implementiert, keine Delegation an NT-API
    - Komplett im Benutzermodus

Blick auf die interne Organisation im Kernmodus

SYSTEMARCHITEKTUR



Der Kernmodus von Windows Vista

Betriebssystemstruktur

- NTOS-Kern besteht aus 2 Schichten
  - NTOS-Kernschicht (kernel)
    - CPU-Verwaltung
      - Implementiert Thread-Scheduling und Synchronisation, sowie diverse Interrupts und Unterbrechungsroutinen
  - NTOS-Ausführungsschicht (executive)
    - Beinhaltet die meisten Dienste

- Hardware-Abstraktionsschicht
  - **H**ardware **A**bstraktion **L**ayer (HAL)
  - Unterste Schicht des Kernmodus
    - Direkt über der Hardware
  - Abstrahiert Hardwaredetails
    - Stellt somit einheitliche Schnittstelle für Zugriff auf diverse Hardware zur Verfügung

- Treiber
  - Alle Kernmodusfunktionen außerhalb des HAL und des NTOS-Kerns als Gerätetreiber implementiert
    - Laufwerke
    - Monitore
    - Dateisysteme
    - Kernerweiterungen
      - Antivirensoftware
      - **D**igital **R**ights **M**anagement (DRM)

Hardware-Abstraktionsschicht

- Maschinennahe Schichten konfrontiert mit variierenden Hardwarefunktionen
  - Geräteregister, Interrupts, DMA etc.
- Angestrebtes Ziel
  - Maximale Portabilität
  - Also Kompilieren des Betriebssystems für unterschiedlichste Hardware ohne manuelle Anpassung

- Funktioniert natürlich nur bei Hardwareunabhängigen Komponenten
  - Arbeiten mit internen Datenstrukturen und Abstraktionen
- Hardwarespezifika die nicht hinter Compiler versteckt werden können, z.B.
  - Unterschied zwischen x86- und SPARC-System
    - Nicht nur unterschiedliche Befehlssätze, sondern komplett verschiedene Architekturen
- Ca. 1% des NTOS-Kerns Assemblercode

- HAL stellt Versuch dar, so viele Hardwareabhängigkeiten des NTOS-Kerns wie möglich durch dünnen Layer über der Hardware zu verstecken
  - Hardwareabhängikeiten zentral im HAL
- Kern und Treiber mit geringem Aufwand portierbar
- HAL Development Kit
  - Anpassung des HAL an „eigene“ Hardware

Kernschicht

- *Kernel Layer*
- Oberste Schicht des Kernmodus
- Abstrahiert ebenfalls darunter liegende Funktionalitäten
- Zwei Hauptaufgaben
  - Verwaltung der CPU
  - Synchronisationsmechanismen

- Kernschicht stellt höheren Schichten *Threads* zur Verfügung
- Aber auch Ausnahmebehandlungen und diverse Interrupts implementiert
- Datenstrukturen die für Realisierung von Threads benötigt werden, in darunterliegender Ausführungsschicht implementiert
- Ebenfalls *Scheduling* und *Synchronisation* in Kernschicht

Ausführungsschicht

- *Executive Layer*
- Untere Schicht des NTOS-Kerns
- In C programmiert
- Dank HAL weitestgehend unabhängig von Hardware
  - Konnte so auf diverse Prozessoren portiert werden
    - MIPS, x86, PowerPC, x64, etc
  - Nur bei Speicherverwaltung manuelle Anpassung

- Komponenten der Ausführungsschicht nach Prinzip der Datenkapselung implementiert
  - Interne Datenstrukturen und Funktionen werden vor anderen Komponenten versteckt
  - Bieten Schnittstelle an, die von anderen Komponenten genutzt werden können
    - Über ntoskrnl.exe teilweise auch den Treibern zur Verfügung gestellt

Objekt-Manager

- Verwaltet zentral alle Kernmodusobjekte der Ausführungsschicht
  - Dateien, Threads, Prozesse, Treiber, Semaphore, ...
- Speicher allokieren, freigeben
- Erzeugt *Handles* für höhere Schichten, über die letztere auf Objekte zugreifen können
  - Ähnlich zu Pointern in C++
- Verfügt über eine Art *Garbage Collector*

E/A-Manager

- Verwaltung von Ein- und Ausgabegeräten
- Stellt Dienste zur Konfiguration von Geräten und Zugriff auf diese bereit
- Bindet nicht nur physische Geräte ein
  - Kern lädt z.B. Dateisysteme und Netzwerkstacks dynamisch

- Viele Treiber können auch im Benutzermodus ausgeführt werden
  - Großer Vorteil für Stabilität des Systems
  - *Blue Screen Of Death* in früheren Windows Versionen hauptsächlich wegen fehlerhafter Gerätetreiber im Kernmodus

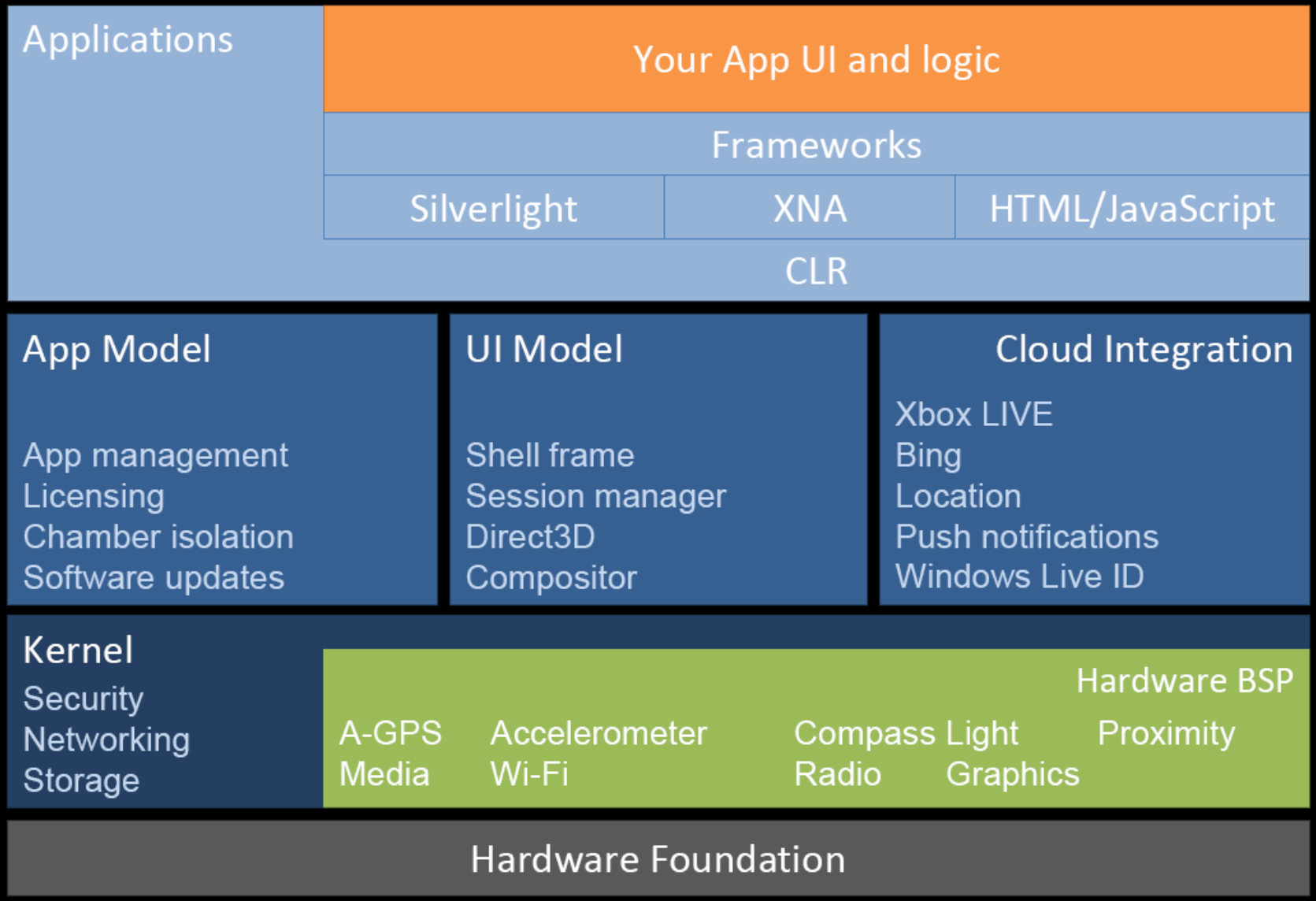
Konfigurationsmanager

- Implementiert Registrierung
- Während Bootvorgang liest HAL Systemkonfiguration aus BIOS aus und schreibt diese in die Registrierung
- Konfigurationsmanager speichert Daten im Dateisystem in *Hives*

WINDOWS PHONE 7

- Weiterentwicklung von Windows Mobile
  - Basiert wiederum auf Windows CE für eingebettete Systeme, Thin Clients und Handhelts
- Auffälligste Neuerung ist grafische Benutzerschnittstelle
  - Erinnert nicht mehr an Desktop-Version
  - Speziell für mobile Endgeräte mit Multi-Touch-Bedienung entwickelt

- Geschlossenes System (ähnlich iOS)
  - Kein direkter Zugriff auf Dateisystem
  - Apps (*Windows Phone Marketplace*)
  - Datenaustausch mit PC nur über Synchronisation mit Hilfe von *Zune*
  - Mindestanforderungen für Handys vorgeschrieben



Windows Phone 7 Architektur

Architektur

- *XNA* Framework zur plattformübergreifenden Spieleprogrammierung
  - Windows, Xbox 360 und Windows Phone 7
- *Silverlight* für herkömmliche Anwendungen
  - Ursprünglich zur Erstellung von *Rich Internet Applications*
  - GUI wird in XAML gestaltet, darunter liegende Logik in C#
    - e**X**tensible **A**pplication **M**arkup **L**anguage

- App Model
  - Verwaltung von Apps
    - Lizensierung
    - Softwareupdates
  - Einzelne App als Paket (.XAP), bestehend aus
    - Metadaten, Icons, DLLs, etc.
  - Nur Ausführung von Anwendungen mit gültiger Marketplace Lizenz
  - Sandbox-Mechanismus zum Schutz des Systems

- UI Model
  - Grundlegende Funktionen für visuelle Darstellung
  - UI-Komponenten als Stack organisiert
    - Werden von Shell Frame zur Darstellung zusammengesetzt
  - Zeitlicher Verlauf von Seitenaufrufen wird gespeichert

Danke für die Aufmerksamkeit