Thoughts on the Future of Runtime Systems

Ben Zorn

Software Design and Implementation Group
Microsoft Research

The opinions expressed in this presentation are those of the presenter and do not represent the views of Microsoft Corporation

Big Picture

- Respirocyte* – “post-biological” era
  - 1 micron nanomedical device intended to replace red blood cells
  - 236 times more oxygen / unit volume vs cell
  - 18 billion atoms, onboard nanocomputer
- Questions
  - What language used to program this device?
  - When will managed languages be used to program every application?
  - What fraction of future applications will be “high dependability”?
- My position
  - Pace of technological innovation is gated by the quality of software infrastructure
  - Most important technology focus of our time
    - MREs important part of a future we can now see

Terminology (Alphabet Soup)

- Managed runtime environment (MRE)
  - Other names: Virtual execution environment (VEE), Virtual Machine (VM), runtime system
- My MRE definition
  - Delivery format that includes
    - Platform neutral intermediate language
    - Metadata for reflection, runtime checking
  - System services (reflection, GC, etc.)
  - Libraries (base class library, frameworks, etc.)

Outline

- MRE evolution over the last 40+ years
- Experience with current commercial MREs
  - Headtrax client in C#
- Future challenges and opportunities
  - Areas of investment
  - Encouraging ideas

MREs Increasing in Role, Function

- Increasingly dynamic software ecosystem
  - Dynamic libraries
  - Components, plug-ins, applets
- Enhanced programmer productivity
  - High-level (e.g., Visual Basic controls)
  - Less bookkeeping (e.g., GC vs malloc)
- Increasing focus on security, privacy
- Language-level feature integration
  - Threads, security model, memory model, etc.

Implications of MRE Evolution

- Increasing overlap with OS
  - Example: isolation mechanisms
    - Use OS processes or CLR AppDomains?
  - Projects: KaffeOS – adding OS functions to MRE
  - What is the right boundary?
- Increasing leveraging of metadata
  - Types, reflection, security – expect more in future
  - More data at runtime sustainable?
- Increasing use in new domains
  - Systems, real-time, embedded, etc.

Commercial MREs a Huge Success

- Productivity benefits real, measurable
  - Higher-level abstractions available
  - Code reuse via libraries
  - More errors detected statically, dynamically
  - Reduced bookkeeping, programmer effort
- Many performance challenges overcome
  - Increased engineering, tools, programmer understanding
  - Sophisticated optimization, runtime systems
  - Successful integration of managed / unmanaged code
- Important application domains remain

The HeadTrax Experience Report

- HeadTrax study (Ovidiu Platon, July 2003)
  - Multi-tier internal MS app manages HR information
  - Client / server - focus on client experience
  - Client configuration: 128 Mb, 1 GHz CPU
- Implementation
  - Client written in C# with .Net Framework 1.1
  - Network interaction via web services and database APIs
  - Security important – strongly signed binaries, encryption
- Preliminary numbers (startup)
  - Cold start 23 seconds
  - Warm start 10 seconds
- Report available at: http://gotdotnet.com/

Improving Performance

- Implemented
  - Made web service calls asynchronous
  - Cache data locally
  - Lazy instantiation of proxies
  - Show UI before populating
- Cold **23 -> 10** secs, warm **10 -> 8** secs
- Proposed
  - Merge assemblies, DLLs
  - Merge threads, use thread pool

Observations

- 10 seconds is still a long time to wait
  - 1500 16+ Kb chunks read from disk at 6 ms / seek
  - Disk is an imposing bottleneck
- Logical and physical organization are at odds
  - E.g., 21 assemblies, 50 DLLs for 1 app
  - Determining “correct” granularity is difficult
- Abstraction can hide high costs
  - XML serialization uses reflection, C# compiler
- Issues not unique to HeadTrax
  - Eclipse, unmanaged apps have similar challenges

Using MREs for Systems

- High performance key to success
  - I/O at startup, during dynamic loading
  - Memory footprint cannot be ignored
  - CPU overheads due to safety, GC, exceptions, security
  - Developer / MRE impedance mismatch
    - What does a developer have to know?
- Next steps are clear, in progress
  - Improved optimization, tools
  - Increase developer experience, education

Future Directions for MREs

- Innovation, experiments, experience needed
- Key challenges
  - Concurrency
  - “Metadata scale” and data locality
  - Error recovery
  - Core architectural issues
    - Modularity, componentization, versioning
  - “Managed code at the bottom” – an all-managed OS
- Singularity Project at MSR
  - Motivation and focus

Concurrency

- Wake up! Chip multiprocessors are here!!!!!!
  - AMD, Intel, IBM all will have dual-core CPUs
  - Technology clearly outpacing research
- Language constructs are brittle, error-prone
  - Threads, shared-memory best approach?
- HW / SW trends toward fine-grain transactions
  - Speculation HW reusable for commit/abort
- Directions:
  - “Atomic” section (e.g., Harris et al.) promising approach to ease programmer effort, reduce errors
  - All alternatives (e.g., *Lisp) need revisiting now

Locality and “Metadata Scale”

- Memory wall growing exponentially
  - Caching, prediction, compression will mitigate
  - GC, MREs (JIT, etc) offer hope here, but…
- Increasing metadata trend exacerbates problem
  - Reflection allows almost arbitrary inspection, creation, execution
  - Metadata required for dynamic checking
- Directions
  - Rethink metadata availability at runtime
  - Increase static checking, improve tools, combine efficiently with dynamic checking

Error Recovery

- Exceptions can be improved
  - Exceptions express control – data consistency left to programmer
- Correct software requires maintaining and reasoning about consistent states
- Increasing the granularity of consistent states
  - Reduces total number of states
    - Easier for human and checking tools to reason about
- Directions
  - Transactions (again) increase granularity of consistent states
  - Expressive annotations, checking tools critical
    - Best error recovery is never encountering one

Modules, Components, Versions

- Modularity – language support still inadequate
  - How to define large-grain decomposition units?
  - Proposals exist (e.g., IBM MJ)
- MREs are currently one-size fits all
  - Are domain-specific MREs valuable, feasible?
    - Beyond J2EE, J2SE, J2ME
  - What mechanisms are necessary to enable?
- Versioning is a critical part of solution
  - How many components in an MRE?
  - Can they be individually up-leveled?
  - How does this look to an application?

“Managed Code at the Bottom”

- All-managed OS / MRE will be necessary
- Keys to building successful systems
  - GC in the kernel
    - Performance, accounting, integration
    - Encouraging research results
  - Type safety in system code (e.g., GC)
    - Typed-assembly language for runtimes
  - Meeting hard resource constraints
    - Space, real-time, hardened to failure
  - Design with compiler / runtime optimization in mind

The Singularity Project

- Revisit OS design from the ground up
- Central focus on high dependability
- Leverage current experience
  - Type-safe (managed) code everywhere
  - Isolate components as much as possible
  - Use software analysis tools in every component at every development stage
  - Be willing to trade performance for correctness
- Result: a research prototype OS / MRE

Summary

- MREs absolutely necessary system component
- Existing commercial MREs
  - Greatly successful, increasing in impact
  - Improvements continue, outcome promising
- Big challenges remain for future designs
  - Accelerating technology trends
  - Core architectural questions
  - Managing complexity key to future success
- Future – MREs everywhere!!! If not, then what?
  - MREs are only the start – checking tools critical too