The Mono Project

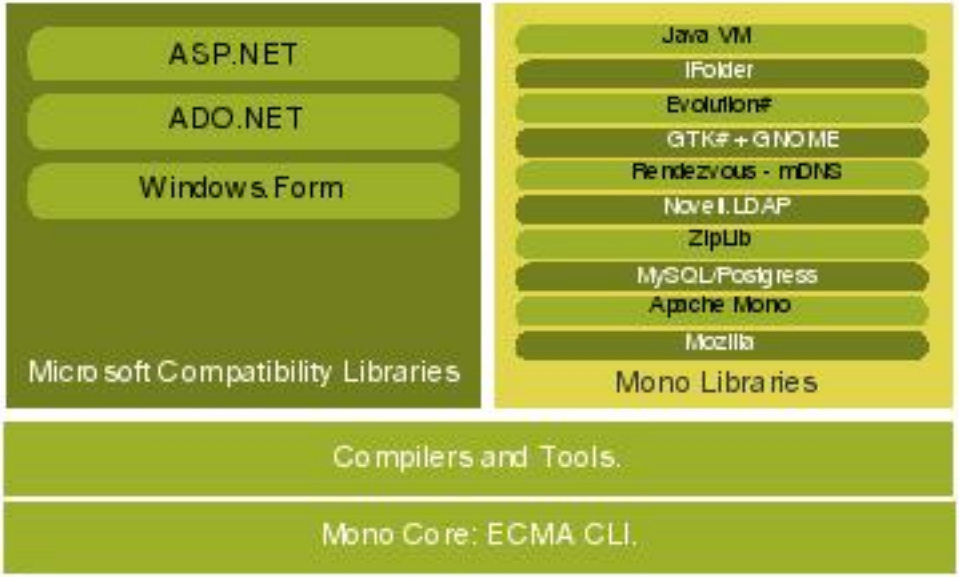
Miguel de Icaza
miguel@novell.com

Novell

Mono.

An Open Source implementation of .NET.

- Multi-platform (x86, ppc, sparc, s390, hppa, arm)
- Multi-OS (Linux, Windows, MacOS X, embedded, others)



Novell.

Background.

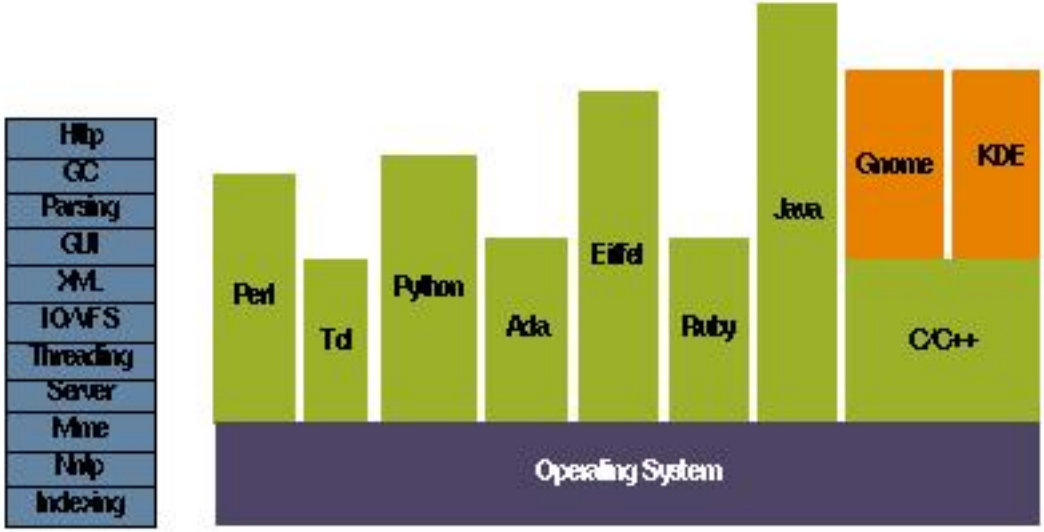
Free Software appeal.

- Social.
- Technical.
- Personal.

1997: Linux becoming common as a server OS

- Server apps, relatively simple.
- Linux craze: the universal data connector.
- Desktop: what we were all using.

Unix:
Everyone Building its Own Platform



Gnome Project.

Bring Linux to the desktop.

- Started in 1997
- Fill the gaps on the desktop offering.
- Raising the programming level:
  - Component-based software.
  - Raise programming level: language bindings.

Technically:

- C-based APIs, contracts to bind it.
- CORBA used for exposing components APIs.
- Far from perfect solutions.
- Bindings came out of schedule.

The Ximian Background.

Ximian: focused on making Linux succeed on Desktop.

Evolution: our large application

Standards compliant, best email/calendar/address.
Complex: multi-threaded, multi-process
Too many standards.
Innovation was hard
At peak for 1.0: 17 developers, 2.5 years Too Expensive.

We needed better tools.

Late 2000: .NET Framework Beta.

The .NET Framework had:

- Garbage collection.
- New high-level language.
- Multi-language support.
- Specifications were released.

Exactly what we were looking for

But only available on Windows!

Mono Launch.

Bring .NET features to Linux.

- C# compiler, VM, Core Class Libraries.

Open Source: well suited to distribute the work.

- Classes can be efficiently distributed.
- API set in stone, sets the design parameters.

Needs to be open source:

- Hard to fund something like this for just building apps.

The team: no experience in compilers or VMs.

Mono today.

Mono: An Open Source implementation of .NET

- Based on the ECMA/ISO standards.
- C#, VB compilers.
- Works with third-party compilers:
  - Delphi, Eiffel, Cobol, Fortran, Mercury, Python.NET, PerlSharp, Nemerle.

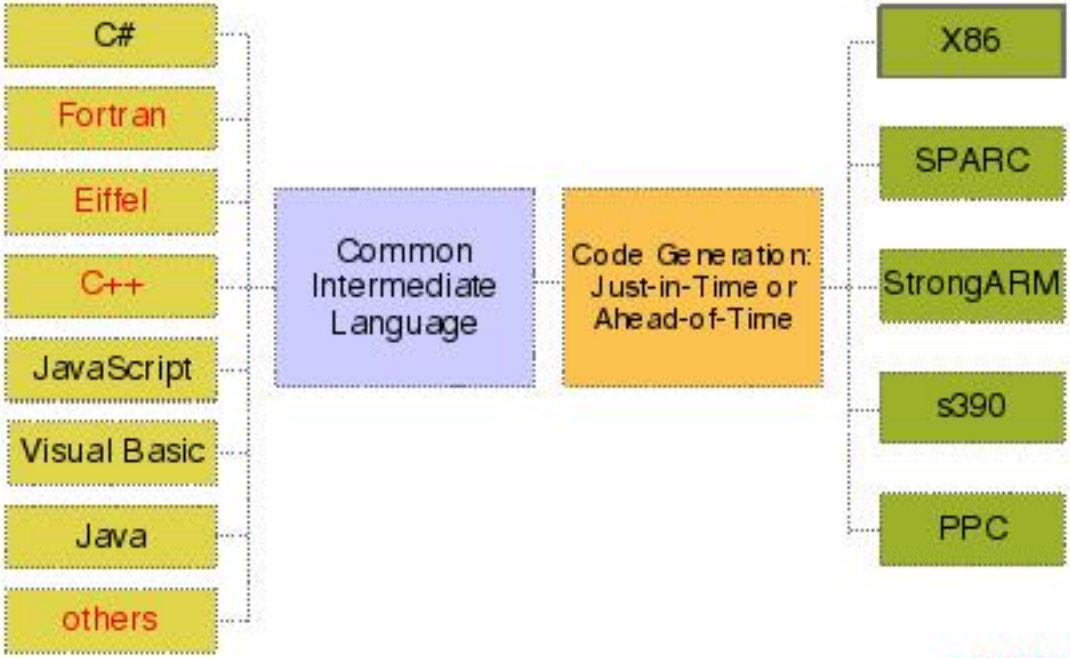
Binary compatible:

- Develop on Windows, Deploy on Linux

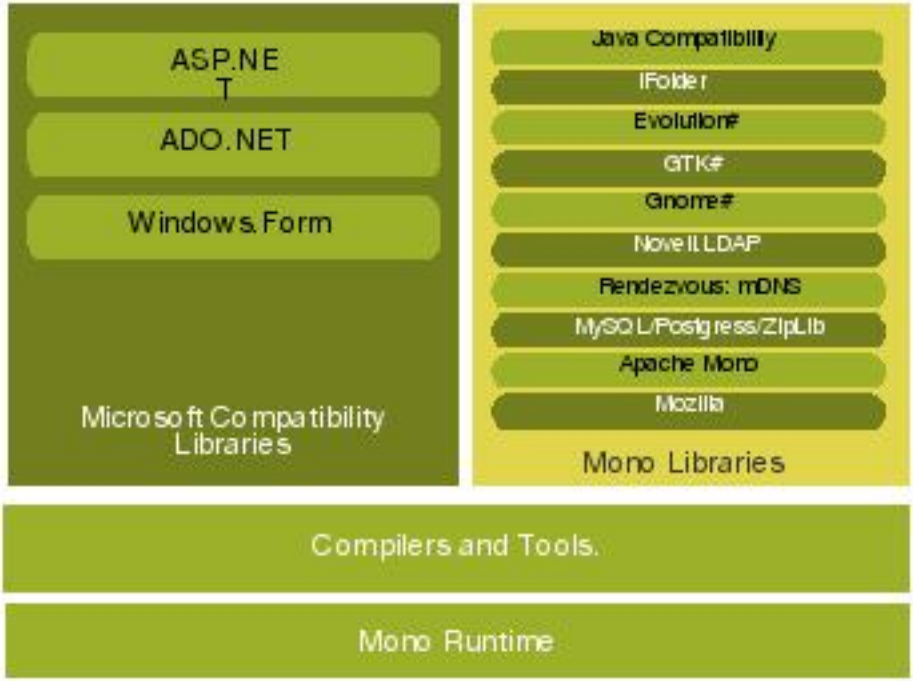
Developers:

- 16 Novell engineers.
- 190 CVS developers.

Multi-language Support.



The Two Stacks.



Novell.

Mono C# Compiler.

- C# written in C#
  - Fast: 2.5 seconds to rebuild itself.
  - 54,000 lines of code.
  - Includes JIT time.
- Implements C# 1.0
  - 2.0: generics and iterators.
- Self-hosting since Jan 2002.
  - Six months since start.
  - Two developers.

Virtual Execution System.

Two execution systems today:

- JIT (ppc, x86, sparc, s390)
- Interpreter (rest of the world).

Object system/metadata

- Clear defined interfaces between this and the JIT.
- Can install more than one JIT (historical need).

GC: Boehm's GC with type descriptors.

- Reduces wasted memory, better tunning.
- Layout objects in a GC-friendly way.
- The most **reused** GC engine.

Mono Runtime.

JIT compilation services.

- Can load AOT code.
- Instrument code on the flight.
- Pluggable interface for profiling/hooks.

Embeddable:

- We use it in our file manager.
- In use on databases like the Virtuoso server (OpenLink).

Mono: JIT history.

First generation.

- Six months to develop, 2 developers
- Again, no experience.
- CIL->Tree re-incarnation of trees
- Burg-based tree-pattern matching.

Problems:

- Hard to port.
- Much duplicated code in burg rules
- Hard to add optimizations.

Inspiration: LCC compiler design.

Discoveries.

LCC book:

- Closest documented description for a simple compiler
- Good for a first JIT foundation.

Advanced Compiler Optimization

- Steven Muchnick

NEC's Citeseer.

Mono CodeGen: second generation.

Second generation goals:

- Add support for pre-compiled code (Ahead-of-Time)
- Portable (x86, sparc, s390, ppc)
- Strong optimizing compiler platform.

Two internal IRs:

- CIL stream to tree (MIR).
- Tree pattern matching produces List of Instructions (LIR)
- Supports SSA.
  - inlining, intrinsic inlining, ssa, constant folding, copy propagation, dead code elimination, arrays bounds check elimination, peephole pass, pluggable regalloc.

Single optimization phase: controlled by flags.

- No distinction between ¨quick' and ¨good' jit.

Some new optimizations.

Today:

- Extensive inlining of intrinsic operations.
- SSA-based representation, good foundation
  - Arrays bounds check elimination.
- Many burg-based improvements.
  - From 16 year old Ben Maurer.

Need:

- For AOT: implement graph coloring regalloc.
- Invariant code motion.

VM research.

Rob Pike's paper on Systems Research.

- Some of his points apply equally well.

Barrier is too high:

- 3 years to build a full stack.
- Hard to innovate with so much cruft.

Research and production licensing might help.

- Mono Runtime: LGPL Compilers: GPL
- Class libraries: MIT X11.
- Microsoft/Sun licensing for research blows.

Some interesting pointers.

GCC's WHIRL-based compiler.

- Hard to evolve GCC's backend, too much cruft.
- SGI Broke the tight frontend/backend integration
- Open64 first released in 1999, WHIRL IR.
- WHIRL: 4-layers of IR, multi-staged (C++, Fortran, C "bits")
- Pluggable engines at each level
- WHIRL to CIL compiler: bring GCC languages to Mono.

Rice's C# REPL

Mainsoft:

- CIL to JVM compiler.
- 'Build with VS.NET, deploy ASP.NET and ADO.NET on J2EE'
- Works out of the Mono codebase.

.NET limitations.

P/Invoke: better than JNI, Great for us.

- Many limitations: too specific, too limited.
- COM/C++ integration: band aid to larger problem.
- Underspecified.

Some pieces are really solid.

- But a lot of MS APIs are one-off throw aways.
- Core is mostly good.

Next step: typelib-based compiler and runtime integration

- Corba, XpCom, Uno, Obj-C: typelibraries.
- Avoid stub creation.

Information.

Mono:

http://www.go-mono.com

Email:

miguel@ximian.com

Blog:

primates.ximian.com/~miguel/activity-log.php

Code Generator Architecture.

Translation:

- Stack-based CIL to Tree-based nodes.
- Tree pattern matching on trees (monoburg)
  - Produces List-of-Instructions.
- Optional: SSA transformation.
- Loop identification, du-chains, liveness analysis
- Constant folding, copy propagation, dead code removal.
- Global Register Allocation.
- Peephole optimizations.

Code generation:

- List-of-instructions to native code.
- Call and branch patching.

Novell®

Novell