SSCLI: The Shared Source Common Language Infrastructure for Research and Teaching

Mark Lewin
Manager, University Research Programs
University Relations Group
Microsoft Research
Redmond, Washington, USA

Academic Days, April/May 2005

Agenda

- Introduction and Motivation
- Overview of .NET
- SSCLI Overview
- Use in Research and Curriculum
- The SSCLI Toolkit
- Summary
- Additional Resources

What is SSCLI?

SSCLI is a shared source implementation of CORE TECHNOLOGIES that underlie Microsoft's .NET architecture.

SSCLI builds and runs on FreeBSD, Windows XP, and Mac OS X, and is portable to other platforms.

SSCLI is designed and documented for ACADEMIC RESEARCH and TEACHING.

What is .NET?

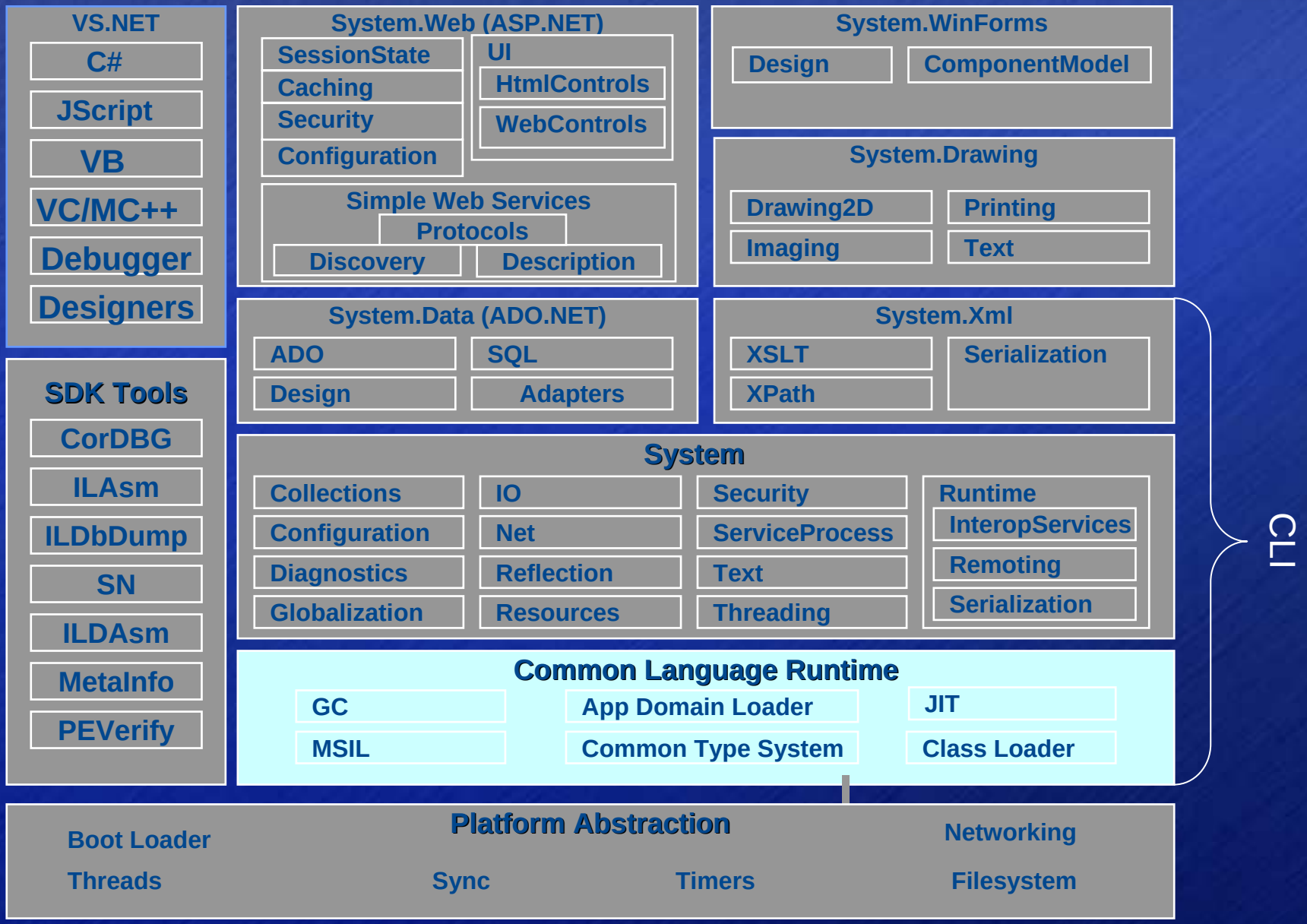
An architecture for creating and running component software

- **An extensible family of compilers and tools**
- **A language-neutral programming framework**
- **A managed execution runtime environment**

C# / CLI Standards

- ECMA
  - Work began Summer 2000 within ECMA working group. Participants included Intel, HP, IBM, Netscape
  - CLI and C# standards ratified Dec 2001 (ECMA 335 and 334)
  - Structure of the CLI standard
    - Partition I – Concepts and Architecture
    - Partition II – Metadata
    - Partition III – CIL (MSIL)
    - Partition IV – The Libraries and Profiles
    - Partition V – Annexes
- ISO
  - National Ballots completed – approved (with comments)
  - Ballot resolution October 2002
  - Standardized December 2002

WHERE THE CLR FITS IN THE .NET WORLD



The Common Language Runtime (CLR)

- Supports Component Software
- Unified runtime services: memory management, virtual object system, structured exception handling, metadata, etc.
- Multi Language Support
- Safe Deployment and Execution
- Improved Performance & Reliability

Assemblies

- Assembly = PE header + MSIL + Metadata + EH Table
- Unit of deployment
  - One or more files, independent of packaging
  - Self-describing via metadata (“manifest”)
- Versioning
  - Captured by compiler
  - Policy per-application as well as per-machine
- Security boundary
  - Assemblies are granted permissions
  - Methods can demand proof that a permission has been granted to entire call chain
- Mediate type import and export
  - Types named relative to assembly

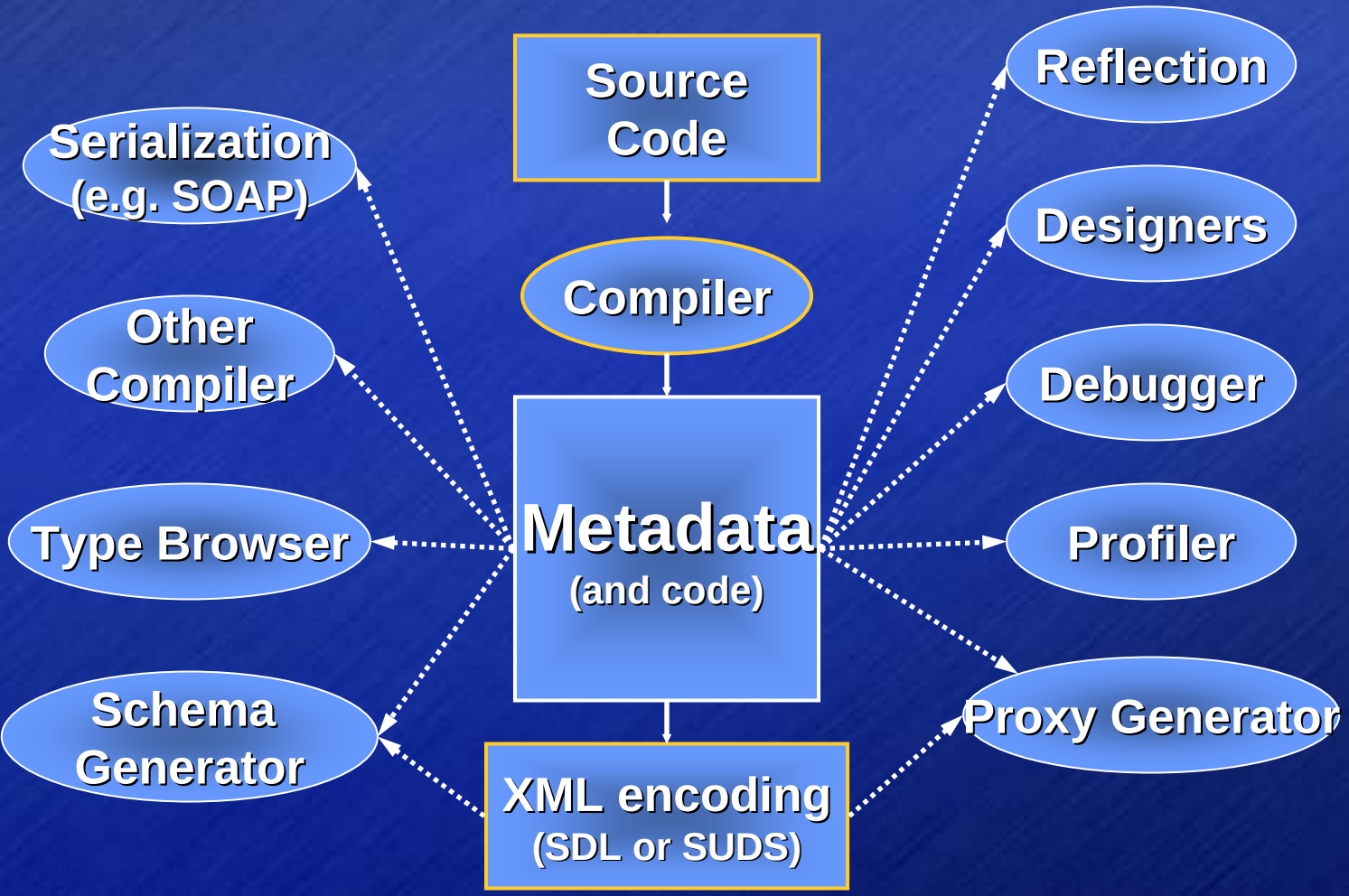
MSIL: Microsoft Intermediate Language

- MSIL is the CPU-independent instruction set generated by .NET compilers from .NET languages such as J#, C# or Visual Basic.NET. MSIL is compiled before or JIT compiled during execution of the program by a Virtual Execution System (VES), part of the Common Language Runtime (CLR).
- Covered in Serge Lidin's talk

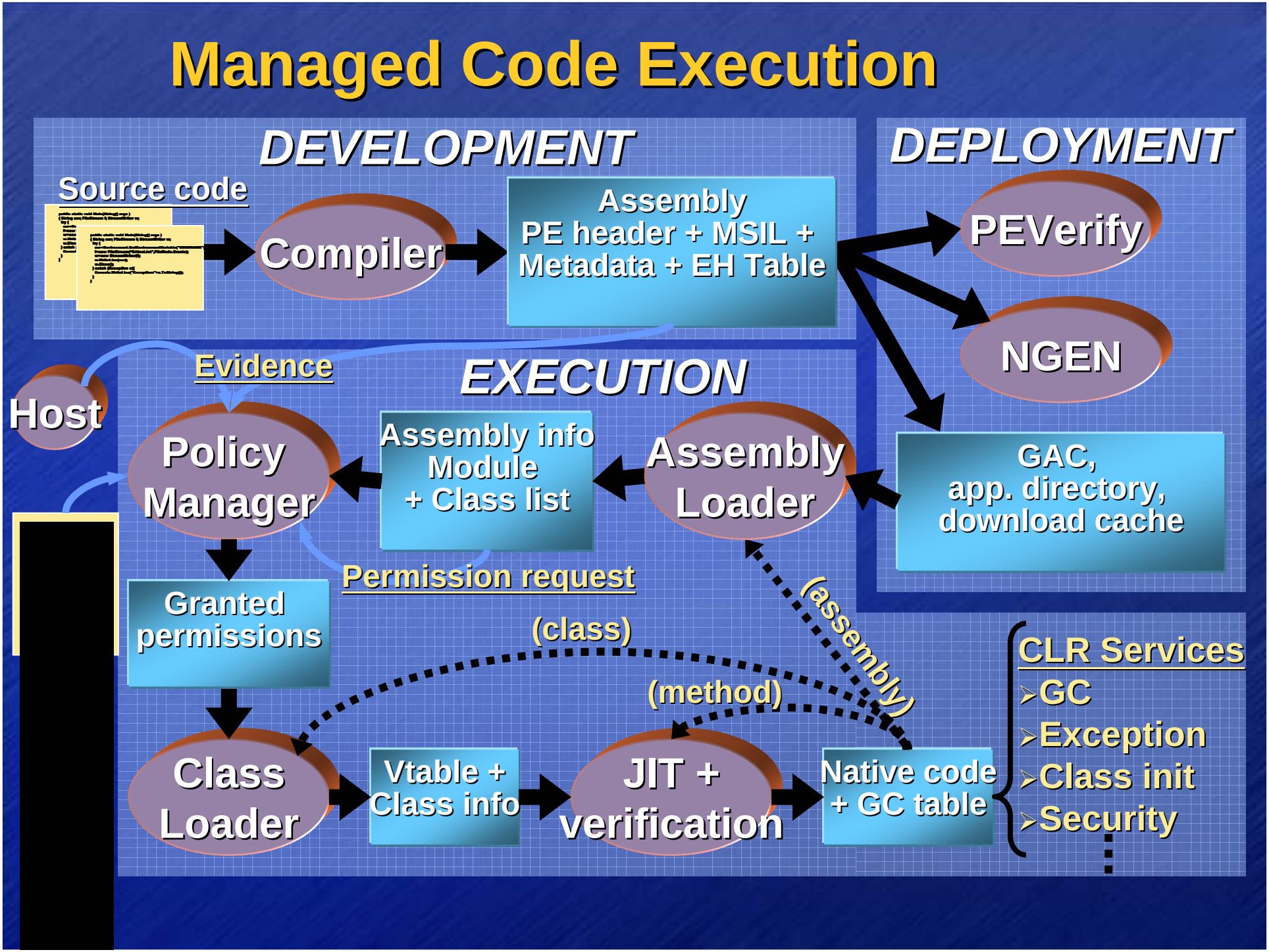
Metadata

- Key to simpler programming model
- Generated automatically
  - Stored with code in executable file (.dll or .exe)
  - Uses existing COFF format
    - Via existing extension mechanism
- Convertible to/from XML Schema

Metadata: Creation And Use



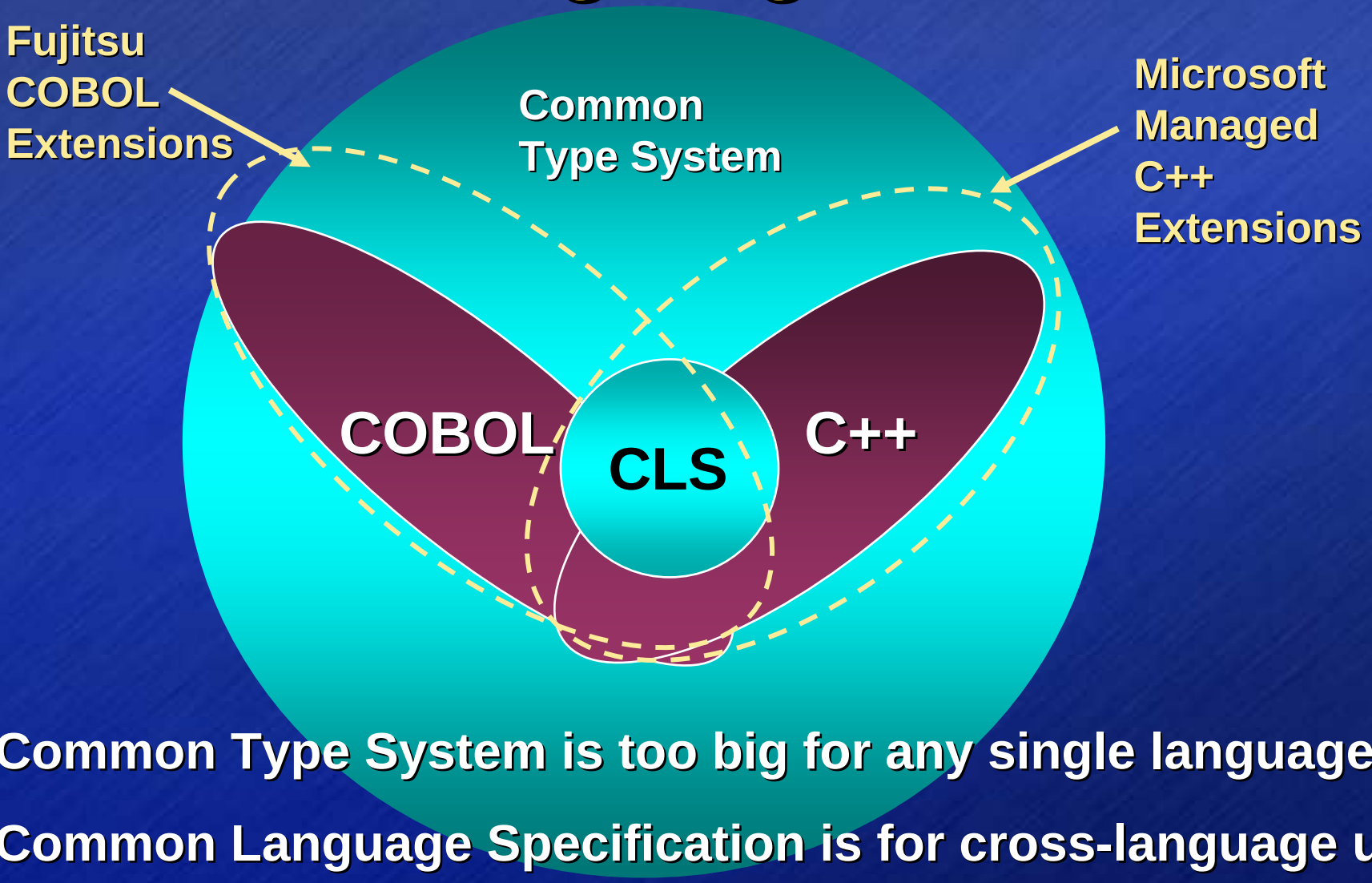
Managed Code Execution



The Common Language Specification

- CLS: Rules for “published” contracts
  - Doesn’t apply within a single assembly
- Full Common Type System for internal contracts and implementation
- Three classes of use:
  - Consumer tools
  - Extender tools
  - Frameworks
- Design and naming patterns

CLS: Designing For Reach



Common Type System is too big for any single language

Common Language Specification is for cross-language use

Some History: Project 7

- Project 7 - Invited independent parties writing language compilers for the CLR, prior to its announcement
- The project demonstrated the multi-language capabilities of the platform.
- Demonstrated that the CLR is a really comfortable environment in which to do compiler development.

Commercial Languages

- APL – Dyadic Systems
- Cobol – Fujitsu
- Delphi - Borland
- Eiffel – Interactive Software Engineering
- Fortran – Fujitsu, Salford Software
- RPG – ASNA
- Pascal – TMT Software
- Smalltalk – Quasar Knowledge Systems

Non-Commercial Language Implementations

- ML – MSR
- Mondrian/Haskell – Utrecht University
- Mercury - Melbourne
- Scheme – Northeastern, Indiana , Northwestern
- Oberon – ETH Zurich
- Component Pascal – Queensland Tech
- Ada – U.S. Air Force Academy

- Others: Pizza, Beta, IronPython, Ruby, lcc, …

Microsoft Languages

- C#
- VB.NET
- J#
- C++
- JScript

CLR Summary

- Component software: “traditional” assemblies and XML Web Services
- Simpler development and deployment
- Rich managed execution services
- Multi-language
- Side by side versioning
- Compiled execution
- Full interoperation with unmanaged code
- Integrated tools support

SSCLI:
The Microsoft *Shared Source* CLI

SSCLI Motivations & History

- Work began in the Fall 2000 on the Shared Source CLI

- Support ECMA/ISO standards efforts
- Validate multi-platform targetability of .NET
- Provide feedback/improvement to commercial offerings
- Create an open laboratory for future work

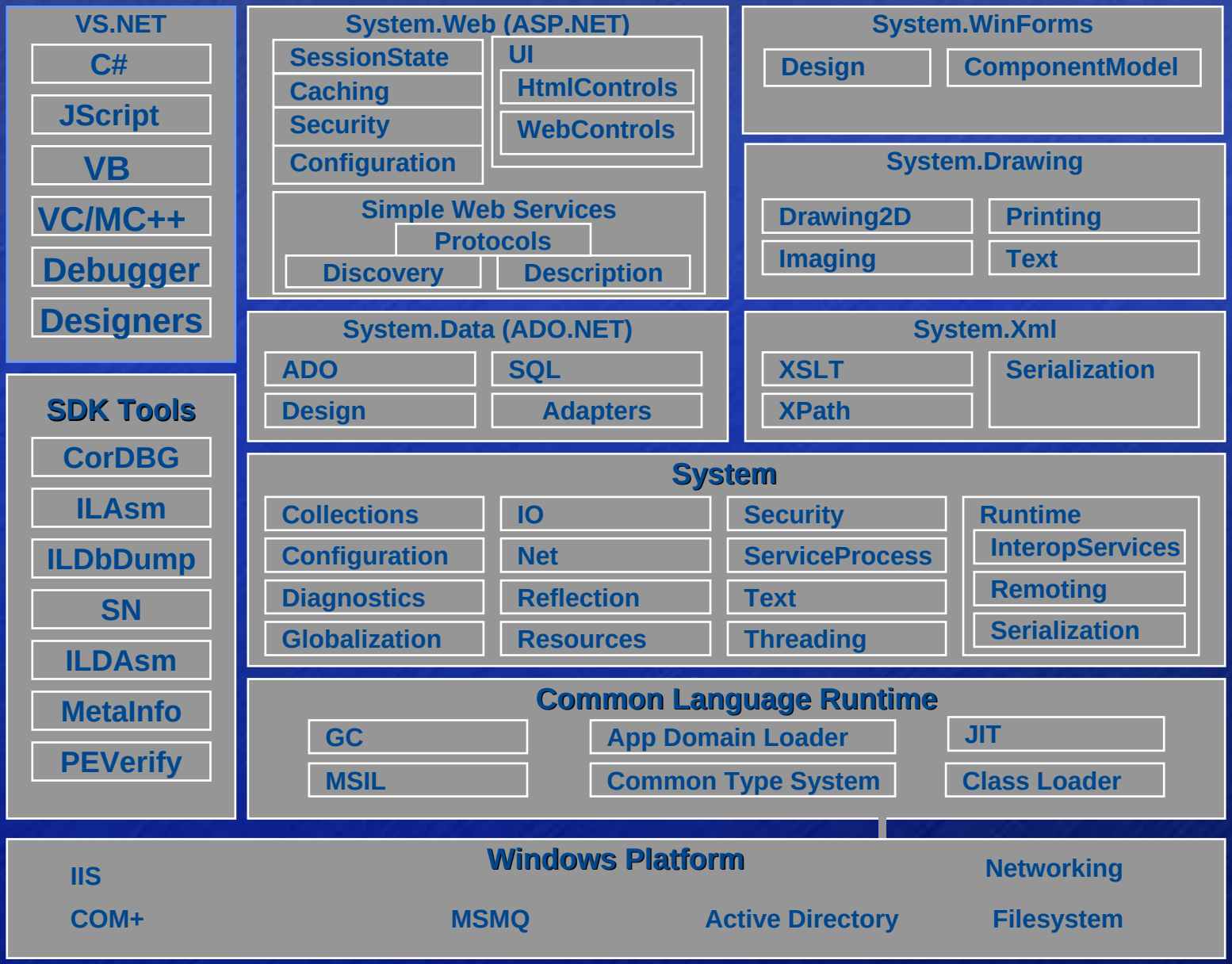
- Milestones
  - SSCLI 1.0 Released October 2001 at OOPSLA 2001
  - SSCLI 2.0 to follow soon after next version of Visual Studio.NET (“Whidbey”)

SSCLI: For Teaching and Research

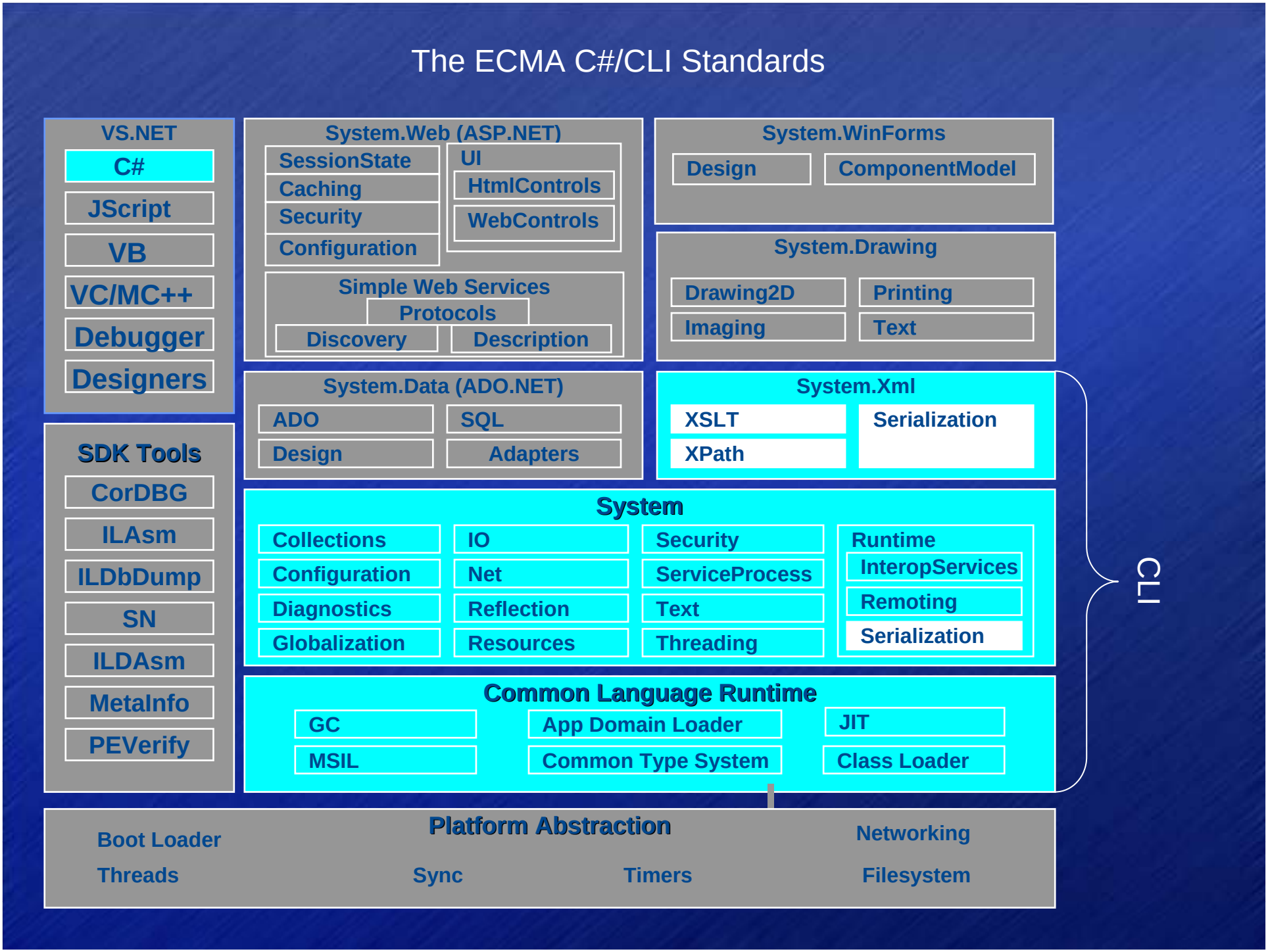
- Complete example to enable research in and support teaching of modern programming languages, compiler design, and runtime infrastructure.

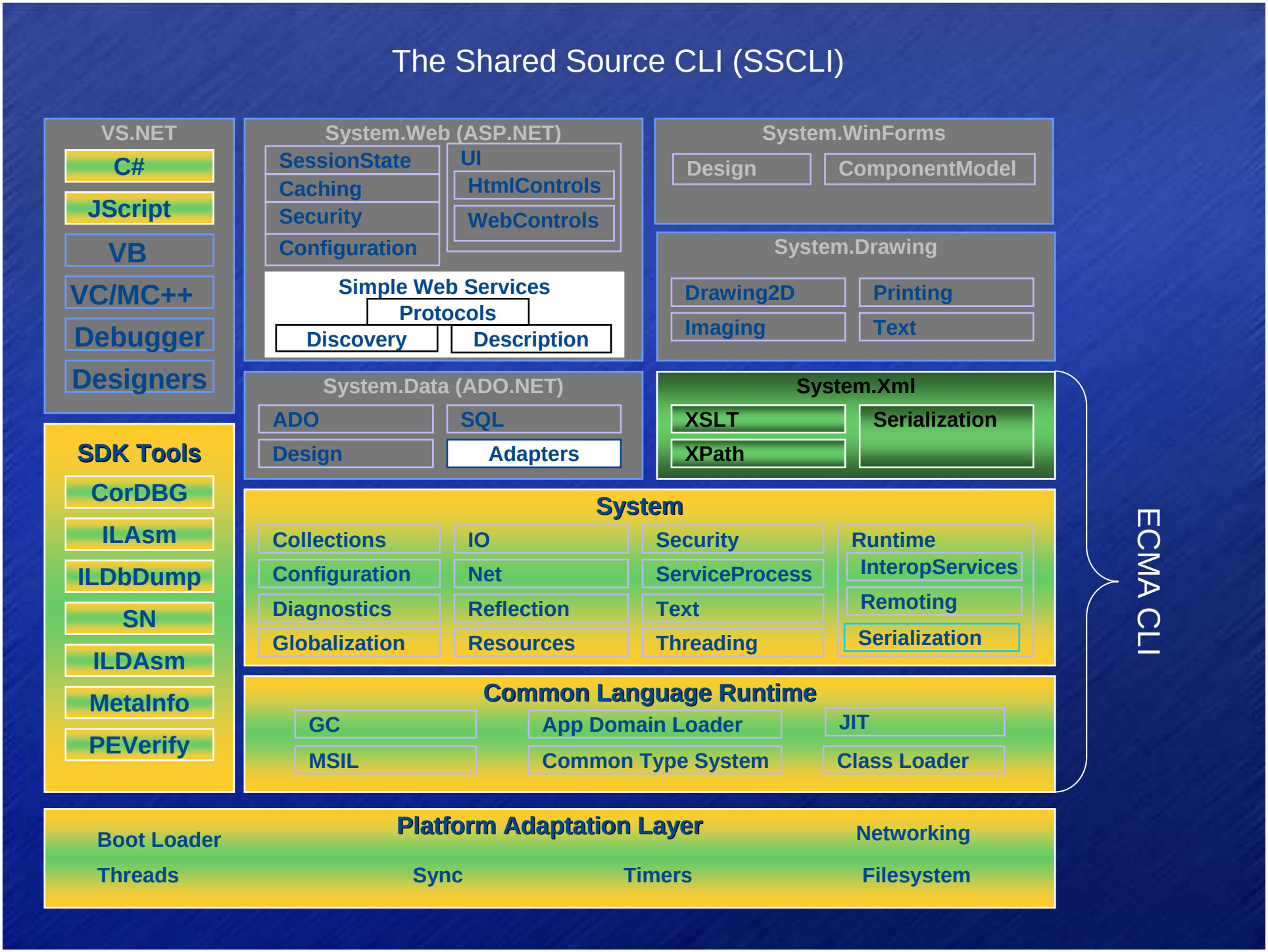
- SSCLI supports the ECMA standards with a real implementation
  - Commercial grade code (but documented for academia)
  - Demonstrates multiple platform support
  - SSCLI license allows for “safe” examination of code
- For compiler writers who want to target CLI:
  - JScript compiler shows dynamic techniques (in C#)
  - C# compiler shows nearly all runtime features
  - IL Assembler shows low-level API implementation and use
- SSCLI also supports .NET programmer learning

The .NET Framework World



The ECMA C#/CLI Standards




The Shared Source CLI (SSCLI)
VS.NET
C#
JScript
VB
VC/MC++
Debugger
Designers
SDK Tools
CorDBG
ILAsm
ILDbDump
SN
ILDAsm
MetaInfo
PEVerify
System.Web (ASP.NET)
SessionState
Caching
Security
Configuration
UI
HtmlControls
WebControls
Simple Web Services
Protocols
Discovery
Description
System.WinForms
Design
ComponentModel
System.Drawing
Drawing2D
Printing
Imaging
Text
System.Data (ADO.NET)
ADO
SQL
Design
Adapters
System.Xml
XSLT
XPath
Serialization
System
Collections
IO
Security
Runtime
InteropServices
Configuration
Net
ServiceProcess
Remoting
Diagnostics
Reflection
Text
Serialization
Globalization
Resources
Threading
Common Language Runtime
GC
App Domain Loader
JIT
MSIL
Common Type System
Class Loader
ECMA CLI
Platform Adaptation Layer
Boot Loader
Threads
Sync
Timers
Networking
Filesystem

Comparing SSCLI and the .NET Frameworks

- Both SSCLI and CLR implement complete ECMA
- SSCLI (slight) superset of ECMA, subset of commercial .NET Frameworks
- SSCLI derived from the main product tree and re-integrated
- SSCLI differences:
  - JIT and GC replaced with more portable implementations
  - Many Windows-specific features not included: COM interop, WinForms, and other integration
  - Commercial features not included, such as ADO.NET, enterprise services, NGEN (JIT-ahead), and ASP.NET

Stats for SSCLI Archive

- Packaged as single compressed file archive
- 1.9 million lines of code
  - 1.15M of C and C++
  - 625K of C#
  - 125K of CIL (intermediate language)
  - Smattering of assembly code
- 5900 source files (9700 total)
  - 2900 tests
- Build output
  - 1200 defined types
  - About 20 dynamically loadable libraries
  - About 22 executable programs

How SSCLI Is Organized

- Four major areas in source code
  1. Runtime “execution engine”
  2. Frameworks
  3. Compilers and tools
  4. Portability layer, tests, and build infrastructure
- Other important points of interest
  - License
  - Documentation
  - Samples

Area #1: Execution Engine

- Heart of component-oriented infrastructure
- Converts metadata, resources, and MSIL (on disk as PE/COFF image) into running code
- JIT compilation and IL verification
- Cross-language exception handling
- Language-agnostic, object-capable, type system
- Automatic heap and stack management
- Dynamic code loading
- Evidence-based security (code access security)

The SSCLI Garbage Collector

- Designed to do things fast – not always pretty
- Heavy use of macros instead of class abstractions or data abstractions
- Code is quite flexible and general with a number of “tuning knobs”
- Many code paths to similar results
- Replaceable

Metadata Facilities

- Metadata is heart of CLI execution model
  - Metadata interwoven with instruction set
  - Read-only PE pages shared between processes
  - Data stored in tables (heaps when variable length)
  - Optimized for load time at expense of generation complexity/speed; compact within these parameters
- Three levels of API
  - Reflection for managed code
  - Both internal and external API for C/C++, see `md` tool

Area #2: Frameworks

- A Toolkit for the 21st century programmer
- Base class library
  - Collections, arrays, strings, and other compound datatypes
  - Globalization and formatting
  - System services (threads, I/O, synchronization, etc.)
  - Security
- Floating point and extended arrays libraries
- Networking, regular expressions, and XML libraries
- Includes access to runtime infrastructure
  - Reflection and custom attributes
  - Remoting, unmanaged interop, serialization, and marshaling
  - AppDomains, Assemblies, GC, other execution engine features

Area #3: Compilers and Tools

- Full-featured C# compiler (also used in build)
- JScript compiler written entirely in C#
- Additional developer tools
  - clix, the shared source CLI program launcher
  - Assembly tools: resource compiler, assembly linker, metainfo metadata viewer, assembler, disassembler
  - Debuggers: cordbg managed debugger, plus a debug extension for working on managed code from C/C++

Area #4: Build, Port, & Test

- Platform Adaptation Layer (PAL) in `pal` directory
  - System resources all consumed via PAL
  - Missing system features layered into PAL on any given platform
  - “PAL runtime”: shared code for upper level tools
- Good specification can be found in pal_guide.html
  - Loading and teardown of context
  - Threading model, scheduling, and timers
  - Synchronous file I/O, asynchronous network I/O
  - Synchronization (critsec, mutex, semaphore, events)
  - Debugging and tool support
- Additional work for port:
  - JIT (very portable 3 layer macro emitter design)
  - Small amount of assembly code in execution engine

SSCLI Licensing Detail

- Part of Microsoft's "shared source" access program
  - Full access to code, including ability to modify and distribute modifications
  - Non-commercial use only (books, courseware, and/or websites that teach the CLI are OK in this)
  - No "contamination" issues for users: "*You may use any information in intangible form that you remember after accessing the Software. However, this right does not grant you a license to any of Microsoft's copyrights or patents for anything you might create using such information*."
- Long-term investment
  - To enable academic research and curriculum
  - To allow better understanding of commercial product
  - To improve technology feedback process by enabling deep input

The SSCLI Toolkit 2005

“demo”

Example Teaching Topics

- Component-based programming
- Comparative languages/infrastructure
  - Language integration
  - Type systems
- Compiler internals, runtime internals (e.g. code optimization, JIT, GC)
- Language topics: verification, security
- Industrial software engineering
  - Internationalization
  - Code structure and maintenance
  - Build and test harness

Rotor in Curriculum: Three Examples

- “.NET Master’s Degree in Distributed Systems Development” - University of Hull, UK
- “Teaching Software Security” - Katholieke Universiteit Leuven, Belgium
- “Compiler Design” – George Mason University, USA

  All will be in Curriculum Repository

Hull: .NET Master's Degree

- Give advanced coverage of modern distributed computing techniques
- Develop skills in working with large codebases
- To develop "active practitioners"
- Provide hands-on practical experience underpinned by advanced theoretical concepts

First 40 students started Fall 2003!

KU Leuven, Belgium

Extended existing course on software security with:

- Additional lecture notes and slides on .NET security
- Project assignments based both on the CLR and on Rotor

Rotor used as primary teaching example and laboratory for capstone projects

George Mason: Compiler Design graduate-level course

- Using C# compiler in SSCLI to illustrate various topics in compiler design. Mutation of internal data structures is observed using the VS.NET debugger.
- Areas covered:
  - Lexical Analysis
  - Parsing
  - Symbol Tables

Rotor Research Projects

- Memory management and GC
- JIT improvements, replacement
- Profile-driven optimization
- Security extensions
- Aspect-oriented programming
- New programming languages
- Concurrency
- Type system extensions

From Research To Curriculum

- 44 Research projects funded through Rotor research grants in 2002
- An additional 40 funded in 2004
- Rotor Research Capstone Workshop held last Fall in Redmond, USA
  - Researchers from over 20 countries
  - 16 refereed paper presented
  - IEE Journal: special Rotor research issue!
- Next step: Capture the experience and insight of researchers for reuse in the classroom

Potential Research Areas

- Proof-carrying code, reflective environments
- JIT-supported pre/post conditions
- Runtime layout, compaction, or rewriting of code/data
- High-level support for mobile/distributed computing
- Generics
- Garbage collection and other memory management techniques
- Distributed computing
  - XML-based
  - Asynchronous
  - Concurrency, memory models, locking
- Others? You Decide!

The Future

- Rotor will continue to evolve to reflect advances in the commercial .NET and ECMA/ISO standards
- Phoenix: a new extensible compiler backend architecture for code generation and optimization
- Microsoft is working to bring more technology, tools, and source code to teachers and researchers to support work in programming languages, compilers, tools, and runtimes
- Let us know what you need!

Questions?

Additional material

Some Rotor projects

Principal Investigator	University	Project
Bishop and Horspool	University of Pretoria (za) and University of Victoria (ca)	Views - Vendor Independent Event and Windowing Systems based on XML for Rotor
Egon Boerger	University of Pisa and ETH Zuerich	Abstract Operational Model for the Semantics of C#
Antonio Cisternino	University of Pisa, Italy	Supporting staged computations within the Common Language Infrastructure
Geoff Coulson	Lancaster University, England	A (reflective) meta-Architecture for .NET
Meng Chiau Er	Multimedia University, Melaka, Malaysia	Design and Implementation of Concurrent C#.NET
Paulo Ferreira	INESC, Portugal	Extending Rotor with Distributed Garbage Collection
Diaz Fondon	Oviedo University, Spain	Adding low-level pure capability-based protection to the CLI for supporting flexible and secure embedded and mobile applications
John Gough	Queensland University of Technology (QUT), Australia	Factorizing the JIT

More Rotor projects

Principal Investigator	University	Project
David Grey	University of Hull, England	Towards a Rotor-based computer science curriculum
Dirk Grunwald	University of Colorado, USA	Runtime Subordinate Threads: Using Simultaneous Multithreading To Improve Runtime Behavior
Jurg Gutknecht	ETH Zurich	Using C# on .NET as a Development Platform for Wearables
Tai-Yi Huang	National Tsing Hua University, Taiwan	Constructing Scalable Web Services using Microsoft Rotor Code
Giuseppe Iazeolla	University of Roma at TorVergata	Modeling the performance of Rotor applications
Roberto Ierusalimschy	Catholic University of Rio de Janeiro, Brazil	Integrating Lua with Rotor
Neil Lawrence	University of Sheffield, England	SIMPL: Simple implementation of a mathematical programming language
Keith Mannock	Birkbeck College, University of London	A simple Interactive Development Environment for C#

Some Rotor projects #3

Principal Investigator	University	Project
Bertrand Meyer	ETH Zuerich	SCOOP on .NET: Implementation of concurrency model using the .NET Remoting Library, based on Design by Contract principles
Christine Mingins	Monash University, Australia	Support for assertions in the Rotor runtime
Valery Nepomniaschy	Russian Academy of Science, Siberian Division, Russia	Towards C# program verification: methods and tools
Paddy Nixon	University of Strathclyde, Scotland	Thread Migration in Rotor
Enric Pastor	Technical University of Catalonia, Spain	Rotor in embedded processors
Nigel Perry	University of Canterbury, New Zealand	JIT Objects and Dynamic Typing
Igor Petrov	Moscow Institute of Physics & Technology, Russia	Pipelined processing of HTTP and SOAP requests
Benjamin C. Pierce	University of Pennsylvania	Xtatic: Regular Expression Types for C#

Some Rotor projects #4

Principal Investigator	University	Project
Frank Piessens	Katholieke Universiteit Leuven, Belgium	Rotor as a vehicle for teaching secure software development
Stefano Pinardi	Universita di Milano-Bicocca, Italy	Teaching with Rotor
Andreas Polze	Hasso Platner Institute, University of Potsdam	Object and Process Migration in .NET
Michel Riveill	University of Nice, France	Interaction Integration into the CLI
Babak Sadighi	Swedish Institute of Computer Science	Delegation and authorization management for middleware
Vladimir O. Safonov	St. Petersburg State University, Russia	Aspect.NET : a Framework for Aspect Oriented Programming
Manual Serrano	INRIA, Sophia, France	Porting Bee to Rotor
Alexander Shargin	St. Petersburg State Technical University, Russia	Run-time optimizations and self-modifying programs in Rotor

Some Rotor projects #5

Principal Investigator	University	Project
Luis Moura Silva	University of Coimbra, Portugal	RAIL: a Runtime Assembly Instrumentation Library
Y. N. Srikant	Indian Institute of Science, Bangalore, India	Profile-guided optimizations for .NET JIT compiler using Rotor
Darko Stefanovic	University of New Mexico, USA	Garbage Collection alternatives for Rotor
Andrey A.Terekhov	St. Petersburg State University, Russia	Course on Compiler Development for .NET platform
Evgeny Vigdorchik	St. Petersburg State University, Russia	Advanced Debugging in .NET
Werner Vogels	Cornell University, USA	High-Performance computing using Rotor
Bruce W. Watson	Technological University of Eindhoven, Netherlands and University of Pretoria, South Africa	Compiling and optimizing regular expression, state machine, and transducer domain-specific languages for CIL
Albrecht Woess	Johannes Kepler University Linz, Austria	Compiler generation tools for .NET

Books

- *Shared Source CLI Essentials* – Dave Stutz, Geoff Shilling, Ted Neward; O'Reilly (March 2003)
- *The Annotated CLI Standard* – Jim Miller, Susann Ragsdale; Addison Wesley (est. November 2003)
- *Distributed Programming Runtime Systems: Inside Rotor* - Gary Nutt; Addison Wesley (est. December 2003)
- *Compiling for the .NET Common Language Runtime (CLR)* – John Gough; Prentice Hall (2002)
- *Essential .NET, Volume I: The Common Language Runtime* – Don Box, Chris Sells; Addison Wesley (2002)

Online Community

- *news:microsoft.public.shared_source.cli*
- http://mailserver.di.unipi.it/mailman/listinfo/dotnet-sscli for mailing list and archive
- http://discuss.develop.com for dotnet-rotor mailing list and archive.

Additional Resources

- ECMA CLI and C# standards documents:

  http://msdn.microsoft.com/net/ecma

- Information on other Microsoft Shared Source offerings:

  http://www.microsoft.com/sharedsource

  (n.b. Windows CE!)

Compilers for .NET

Academic Projects

Ada - A# - http://www.usafa.af.mil/dfcs/bios/mcc_html/a_sharp.html
Beta - http://www.pervasive.dk/projects/langInter/langInter_summary.htm
Forth - Delta Forth - http://www.dataman.ro/dforth/
Haskell - Hugs98 for .NET - http://galois.com/~sof/hugs98.net/
lcc - lcc.NET - http://www.cs.princeton.edu/software/lcc/
Mixal - MixNet - http://sourceforge.net/projects/mixnet/
Mercury - http://www.cs.mu.oz.au/research/mercury/dotnet.html
Mondrian - http://www.mondrian-script.org/mondrian/index.html
Oberon - Active Oberon - http://www.oberon.ethz.ch/lightning/
Pascal - Component Pascal - http://www.citi.qut.edu.au/research/plas/projects/cp_files/cpnet.html
P# (Prolog/C# hybrid) - http://www.dcs.ed.ac.uk/home/jjc/psharp/psharp-1.1/dlpsharp.html
Ruby - Ruby/.NET Bridge - http://www.saltypickle.com/rubydotnet/
Scheme – Grand Larceny (Northeastern/Will Clinger) - <no URL yet>
Scheme - Hotdog Scheme (Northwestern) - http://rover.cs.nwu.edu/~scheme/
Scheme - Chez Scheme (Indiana) - <no URL yet>
Scheme - Tachy - http://radio.weblogs.com/0101156/stories/2002/03/19/tachy.html
Smalltalk - S# - http://www.smallscript.net/
Zonnon - http://www.bluebottle.ethz.ch/Zonnon/

More .NET Compilers, Tools

MSR Projects

Asml - Abstract State Machine Language - http://research.microsoft.com/fse/asml/

F# - http://research.microsoft.com/projects/ilx/fsharp.aspx

Gyro - Generics for C# and the .NET CLR - http://research.microsoft.com/projects/clrgen/

Polyphonic C# - (paper only) - http://research.microsoft.com/~nick/polyphony/

SML - http://research.microsoft.com/projects/sml.net/

Aspect-Oriented Programming Projects

Aspect.NET - http://research.microsoft.com/programs/europe/rotor/wshop-aspectnet.ppt

Aspect Weaver - http://www.devhood.com/tutorials/tutorial_details.aspx?tutorial_id=572

LOOM.NET - http://www.dcl.hpi.uni-potsdam.de/cms/papers/papers/isorc2002.pdf

WEAVE.NET - http://www.dsg.cs.tcd.ie/index.php?category_id=194

AspectC# - http://www.dsg.cs.tcd.ie/index.php?category_id=169

Compiler Writing Tools

Coco/R – Recursive Descent Compiler Generator - http://dotnet.jku.at/Projects/Rotor/

Details of Hull degree program

Hull: Degree Structure

Three stages:

- Certificate Stage
  - Skills to underpin the original work
- Diploma Stage
  - Advanced computing topics
- Master's Stage
  - Practical deployment of techniques

Hull: Certificate Stage

Introduction to C#	Introduction to .NET	Extensible Systems
Object oriented design and development Object oriented programming using C# Software engineering/ UML design Testing and debugging skills	Component based development Managed code environments Framework components Common Language Runtime Common Type System	Navigating, understanding, modifying and maintaining large-scale, commercial quality source code Testing and debugging strategies Commercial development skills Shared-source .NET as an example of large-scale commercial quality software

Hull: Diploma Stage

Distributed Programming	Virtual Machine Architectures	Trustworthy Computing
Architectures and paradigms for distributed computing .NET support for networking and distributed computing Remoting & Web services Mobile computing	Requirements for managed code environments Intermediate languages and their design Virtual Machine architecture and implementation Shared source .NET implementation	Security risks and the need for security Trust and authentication Security techniques and algorithms Language security Security in distributed systems .NET & Rotor security models

Hull: Masters Stage

- Large, individual project based on real-world problem using Rotor or other commercial-grade software
- Examples
  - FORTH implementation
  - Custom Garbage Collection
  - Custom Remoting / object mobility
  - Code profiling

Thank You!