The .NET Framework:
What is it and How to Use it?

Drs.ir. Vadim V. Zaytsev

19 October 2004

History en Future of Cobol

- The lecture in week 45 will be **cancelled**!

- Replacement lecture will take place as a part of a bigger event:

- A symposium in honour of **Wim Ebbinkhuijsen**: 22 October 2004 (Friday), 13:00, Auditorium. Please read information at `http://www.automatiseringgids.nl/events/default.asp?page=hfcobol` and then register via a.luisman@wkths.nl (or contact Ralf).

Technical issues

- Up-to-date information about the course: requirements, suggestions, slides, papers, rescheduling issues, . . . — http://www.cs.vu.nl/~ralf/oo/lecture-2004/

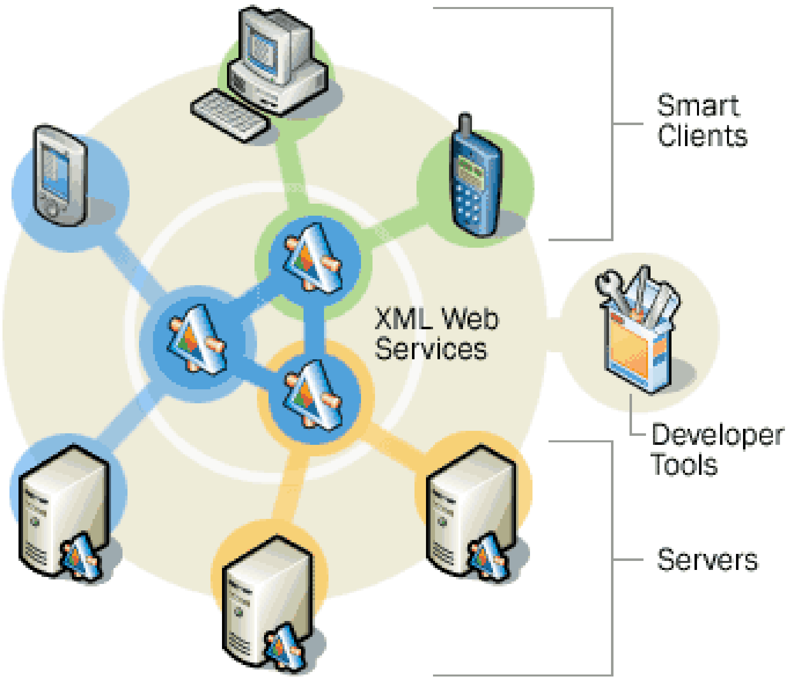
- These slides incorporate some of the work by Ralf Lämmel, Manuel Costa, Kai Rannenberg, Erik Meijer, Damien Watkins, Hanspeter Mössenböck & probably some others.

What is .NET?

- Microsoft .NET is a set of Microsoft software technologies for connecting information, people, systems, and devices. It enables a high level of software integration through the use of Web services—small, discrete, building-block applications that connect to each other as well as to other, larger applications over the Internet. (© *M$ website*)

- A development platform: interfaces, components and tools to develop software. The biggest change in the Microsoft platform since Windows NT replaced DOS. (© *Manuel Costa*)

The components of Microsoft .NET-connected software



Web services are small, reusable applications written in XML, a universal language for data exchange. They allow data to be communicated across the Internet (or internal intranet) between otherwise unconnected sources that are enabled to host or act on them, for example:

Client-to-client: "Smart" clients or devices can host and apply XML Web services that allow data to be shared anywhere, any time.

Client-to-server: XML Web services can share data from a server application to a desktop or mobile computing device via the Internet.

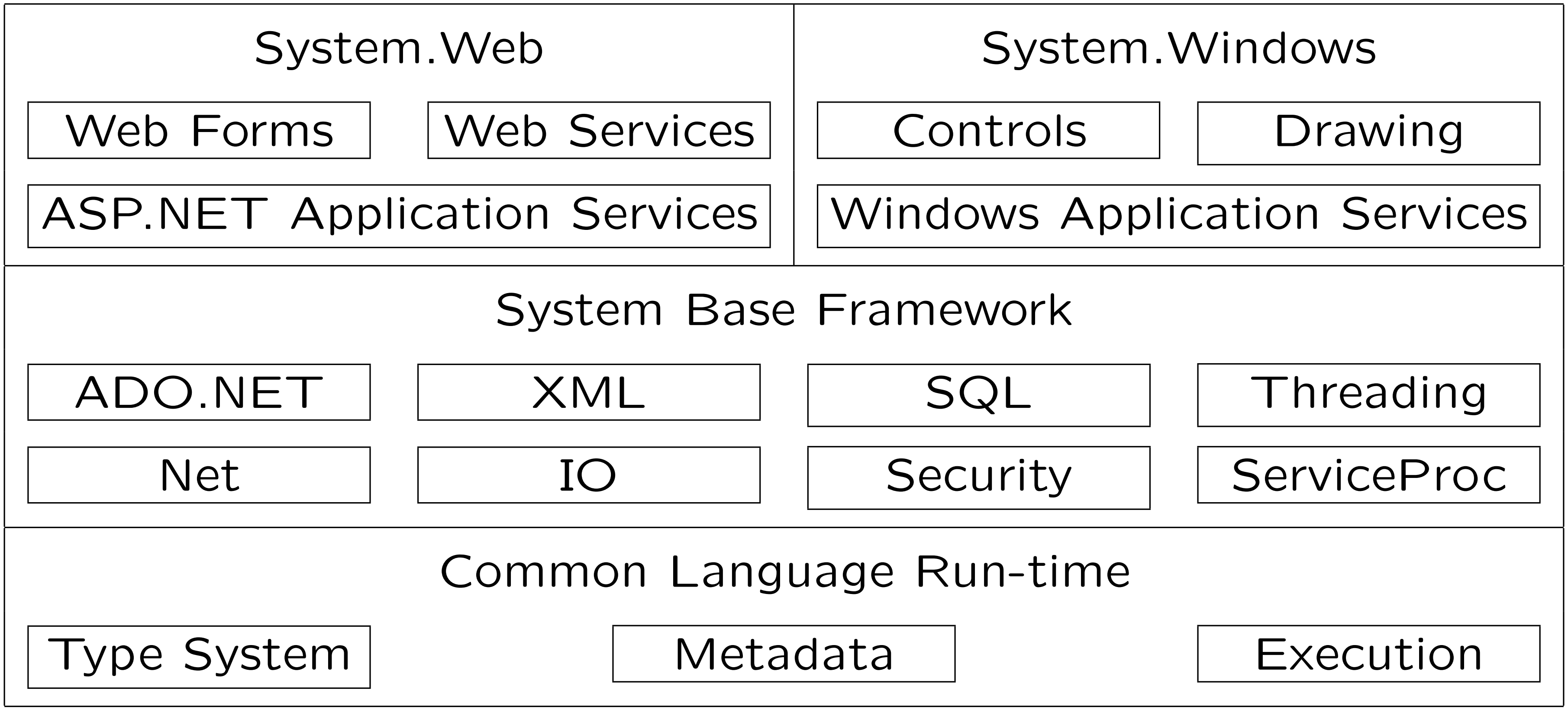
Server-to-server: XML Web services provide a common interface between existing applications within an environment of independent servers.

Service-to-service: XML Web services can work together in sequence to create a more complex data operation.

.NET framework principles

- Make Internet-scale distributed computing ubiquitous
- Seamless integration of multiple applications and devices
- Deliver software as a service
- Independent of any programming language

.NET framework as a *framework*



System.Web

Web Forms

Web Services

ASP.NET Application Services

System.Windows

Controls

Drawing

Windows Application Services

System Base Framework

ADO.NET

XML

SQL

Threading

Net

IO

Security

ServiceProc

Common Language Run-time

Type System

Metadata

Execution

Metadata

- Metadata generation is both mandatory and automatic
- Metadata is the essential bridge between language compilers and the execution system
- Metadata annotations are extensible via Attributes (explicitly specified by a programmer):

```
[STAThread]
static void Main()
{
    Application.Run(new MainForm());
}
```

Common Language Run-time (CLR)

- Multi-language support
- Common type system
- Simplified deployment
- Code Access Security

Corporation support

- Rich class libraries
  - Powerful and consistent programming model
  - Focus on code, not plumbing
- Tools
  - Support for design-time functionality
  - Debugging, profiling, instrumentation support

CLR design goals

- Simplify application development
- Simplify deployment and management
- Provide a robust and secure execution environment
- Support multiple programming languages

Simplified development (example)

Windows API (C++)

```
HWND hwndMain = CreateWindowEx(
    0, "MainWinClass", "Main Window",
    WS_OVERLAPPEDWINDOW | WS_HSCROLL | WS_VSCROLL,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    (HWND)NULL, (HMENU)NULL, hInstance, NULL);
ShowWindow(hwndMain, SW_SHOWDEFAULT);
UpdateWindow(hwndMain);
```

.NET Framework (C#)

```
Form form = new Form();
form.Text = "Main Window";
form.Show();
```

Simplified development

- **Organisation** — code organised in hierarchical namespaces and classes.

- **Unified type system** — everything is an object, no variants, one string type, all character data is Unicode.

- **Component-oriented** — properties, methods, events and attributes are first class constructs.

Simplified deployment & management

- **Assembly** — a unit of deployment, versioning and security; very much like a DLL, but self-describing.

- **Zero-impact install** — applications and components can be shared or private.

- **Side-by-side execution** — multiple versions of the same component can coexist, even in the same process.

Robust & secure

- **Automatic lifetime management** — all .NET objects are garbage collected; no stray pointers, no circular references.

- **Code correctness and type safety** — IL can be verified to guarantee type-safety; no unsafe casts, no uninitialised variables, no out-of-bounds array indexing.

- **Evidence-based security** — based on origin of code as well as user; extensible permissions possible.

Multi-language friendly

- All features of the .NET platform available to any .NET programming language.
- Application components can be written in multiple languages.
- Debuggers, profilers, code coverage analysers, . . . work for all languages.
- Available: *(on the next slide)*

Available languages under .NET

A# (Ada), Abstract IL (IL+OCaml), Active Oberon, ActiveState Python, ASNA Visual RPG, BETA, Boo (python), C#, Cω, Component Pascal, Delphi 2005, Delta Forth .NET, DotLisp, Dyalog APL, Eiffel, F# (ML+Caml), Glasgow Haskell, Haskell.NET, Hugs98 (Haskell), HotDog Scheme, IL (a.k.a. MSIL, CIL), ILX (functional IL), IronPython, JScript.NET (ECMAScript), Lahey Fortran, Lexico (educational), Mercury (Prolog, kinda), Mondrian, MonoLOGO, Nemerle (functional C#), NetCOBOL, Net Express (MicroFocus COBOL), Oberon, PerlNET, Python, Salford FTN 95 (Fortran), Scheme.NET, S# (Smalltalk 98), #Smalltalk, SML.NET (Standard ML), Tachy (Scheme-like), TMT .NET Pascal, Visual Basic, Visual C++, Visual J# (Java), Zonnon (Oberon trend).

This is fourty six!

Example: Visual C++ (Managed)

```
#using <mscorlib.dll>
using namespace System;
__gc public class HelloWorldCPP
{
  public:
  void SayHelloCPP()
  {
    Console::WriteLine("Hello World from C++!");
  }
};
```

Example: Visual Basic

```
Imports System
Imports HelloWorldCPP

Public Class HelloWorldVB
Inherits HelloWorldCPP
   Sub SayHelloVB()
      Console.WriteLine ("Hello World from Visual Basic!")
   End Sub
End Class
```

Example: COBOL

```
CLASS-ID. HelloWorldCOB INHERITS HelloWorldVB.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    CLASS HelloWorldVB AS "HelloWorldVB"
OBJECT.
PROCEDURE DIVISION.
METHOD-ID. SayHelloCOB.
PROCEDURE DIVISION.
    DISPLAY "Hello World from COBOL!".
END METHOD SayHelloCOB.
END OBJECT.
END CLASS HelloWorldCOB.
```

Example: C#

```
using System;
class HelloWorldCS: HelloWorldCOB
{
  public void SayHelloCS()
  {
    String message = "Hello World from C#!";
    Console.WriteLine(message);
  }
  public static int Main()
  {
    HelloWorldCS h = new HelloWorldCS();
    h.SayHelloCPP();
    h.SayHelloVB();
    h.SayHelloCOB();
    h.SayHelloCS();
    return 0;
  }
}
```

.NET availability

- Standardised by ECMA-335: CLI, ECMA-334: C#, ISO/IEC 23271:2003 IT–CLI, ISO/IEC 23270:2003 IT–C#.
- **.NET Framework SDK** — essential part, around 100 Mb, free to download, just CLR and basic tools.
- **Visual Studio .NET** — huge (all meanings), not quite free: $749–$2499.
- **Rotor: SSCLI** — shared source, free to download, working on Windows XP (of course!), FreeBSD, Mac OS X 10.2.
- **Mono** — comprehensive open source development platform based on the .NET framework, sponsored by Novell, free to download, works on Linux, not completed yet.

break

Security: Policy

- Defining **security goals**
  - What do I want to protect?
  - From whom?
  - How do I express it?
  - How do I know it is right?
- Different **parties** have different **interests** and different (maybe conflicting) **policies**
- Approaches:
  - Policy languages
  - User Interfaces Tools

As it is usually done: plumbing

- Implementing **security functionality**
  - Assuming I have a policy, how do I **implement** it?
    (Application security)
  - How do I **enable** implementation of the **widest** range of policies?
    (OS/Network security)
- Dealing with bugs
  - How do I **minimize** security holes in the plumbing?
  - How do I **cope** with them?
  - How do I **recover** from their effect?
  - Approaches include: filters, firewalls, code checkers, audition tools.

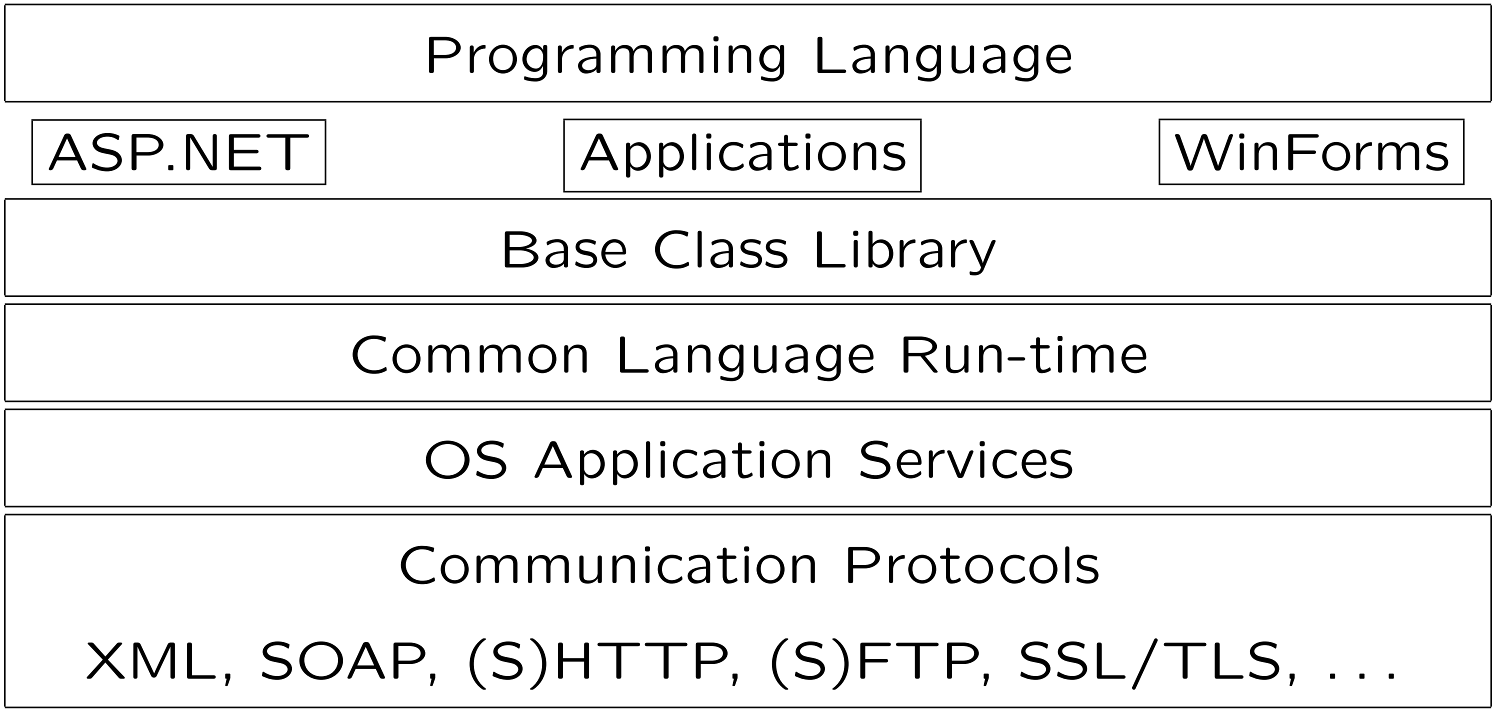
Distributed security

- The trust model is fantastically complex (partial or limited trust defined by policies, contracts, liability, educated guessing).

- The “Trusted Computing Base” is **exposed** (includes interfaces between the software and the system, network, user and other code)

- Security usually contradicts reliability or performance.

.NET framework



Programming Language

ASP.NET

Applications

WinForms

Base Class Library

Common Language Run-time

OS Application Services

Communication Protocols

XML, SOAP, (S)HTTP, (S)FTP, SSL/TLS, . . .

+IDE “Visual Studio .NET”

CLR security design goals

- Robust security system for **partially-trusted, mobile** code
  - OS security is based on *user* rights
  - CLR security (on top of OS security) gives rights to *code*
- Make it easier for. . .
  - **Developers** to write secure applications (standard libraries implement security checks for exposed resources; easy to perform security checks in user code)
  - **Administrators** to express their policies (fine-grained authorisation models; system is extensible)
  - **End users** to work securely (no run-time security decisions are to be made on the fly)

The four scenarios

	Trusted user	Untrusted user
Trusted code	should-be usual situation	limited database access
Untrusted code	virus or another malicious software	crystal clear. get out!

Permission

- A **permission** is a set (or subset) of capabilities
  - The right to access a particular resource
  - All permissions implement $\cup$, $\cap$, and $\subset$ operations

- Permission types are orthogonal (a demand for a permission of type A must be satisfied with a grant of a permission of type A)

- Permissions protect resources

- Assemblies need permissions

Policy

- **Policy** determines the set of **permissions** to grant to code based on **evidence**

- Classic trust management problem

- Solution?
  - End users write programs to express their policies?
  - Base on administrator's experience (evidence)?
  - . . . ?

How it is done in .NET

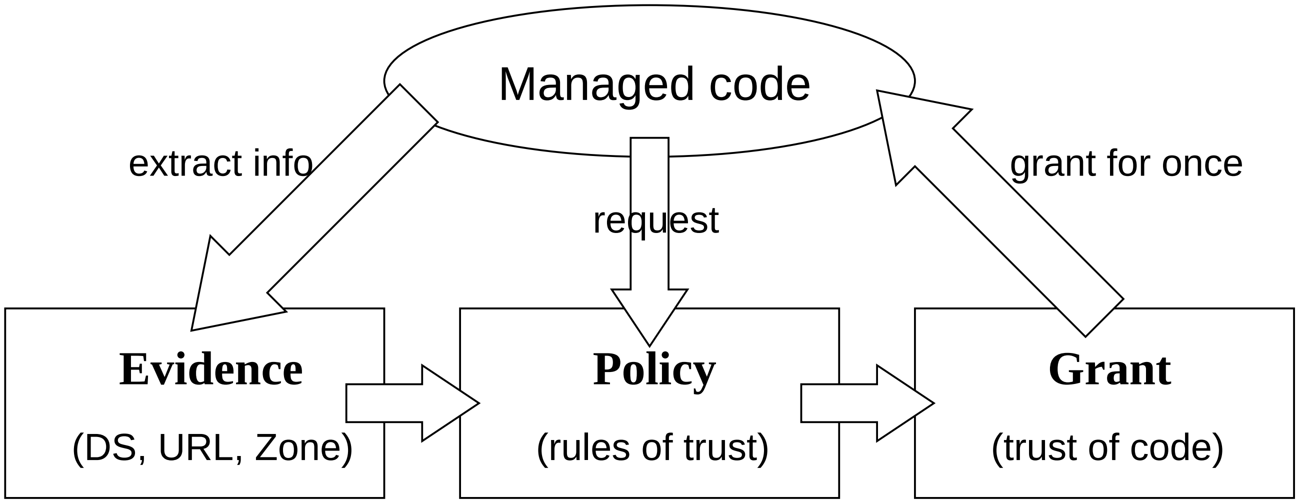
```
[SendMailPermission(
        SecurityAction.Demand,
        Sender="kair@microsoft.com")]
public static void SendMessage(...)
    {
        ...
    }
```

- Programmer defines `SendMailPermission` and decides when to demand it of callers

- Administrator decides what code should be granted `SendMailPermission`

Policy evaluation

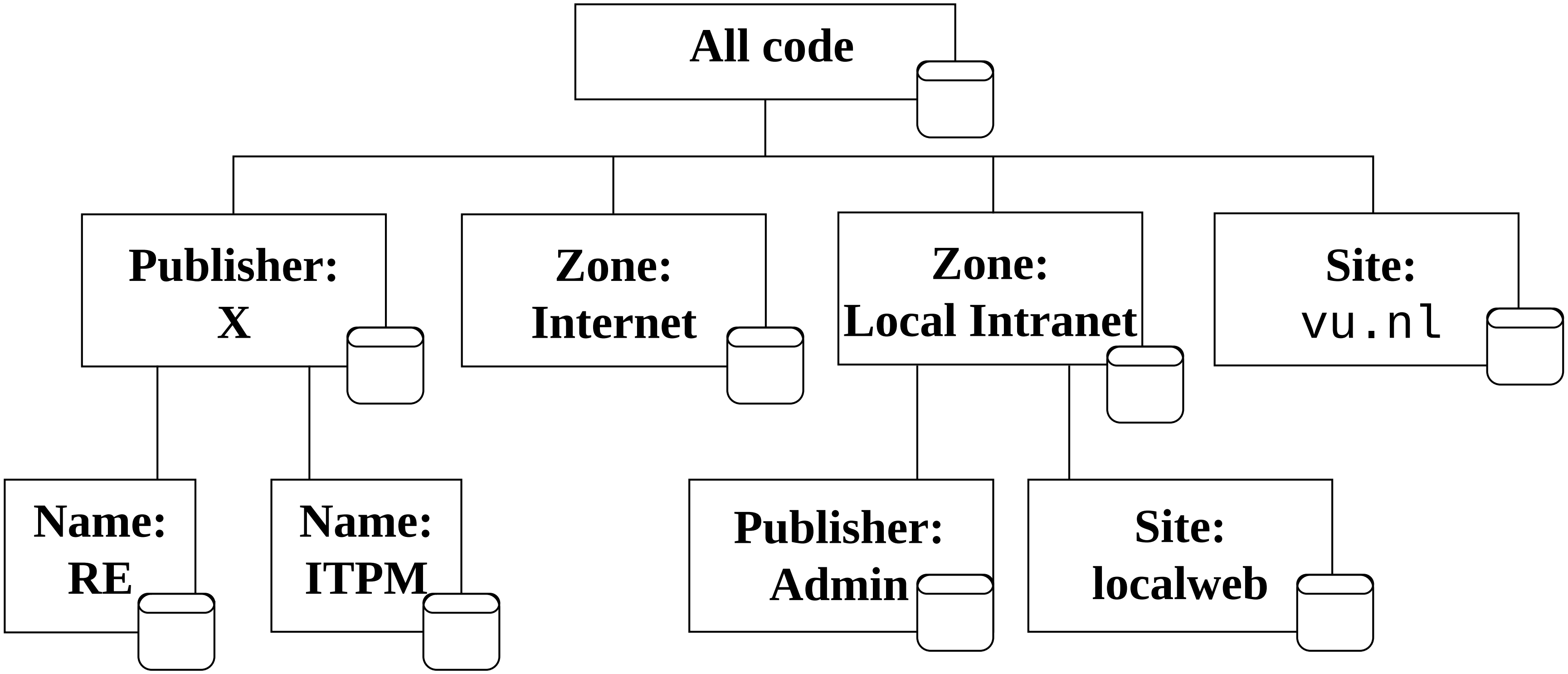
- process of determining the set of *permissions* to grant to code based on
  - **Evidence** known about that code
  - **Requests** from the code



Notions of **code group** and **policy level**

- Code group groups assemblies that should be granted similar permission
- Code groups are organised into a hierarchy
- Membership for each assembly is evaluated w.r.t. evidence
- A tree of code groups is a *policy level*.
- The permissions granted by a policy level for a given set of evidence are determined by evaluating the root code group of the tree.

Sample policy level



Evidence

- Evidence is the input to policy evaluation
- For example: information about assembly (strong names, publisher identity, original location), third-party certifications
- Evidence is extensible (any object can be a piece of evidence)

Assembly input: permission requests

- **Minimum** (must have to run)
- **Optional** (would like to have to run)
- **Refuse** (never need)

C#

- Made by Anders Hejlsberg, Scott Wiltamuth, Peter Golde

- 70% Java, 10% C++, 5% Visual Basic, 15% new (claimed)

- Mostly C++, Deplhi, Modula, Smalltalk

- Syntactically almost Java.

- Different points of view, see e.g. *C#: A language alternative or just J--?*.

C# features

- Object-orientation (no multiple inheritance)
- Interfaces
- Exceptions (+checking)
- Threads
- Namespaces (independent of file structure)
- Strong typing, unified type system
- Garbage collection **and** destructors
- Reflection, dynamic loading of code
- Method / operator overloading
- Pointer arithmetic in unsafe code
- Reference and output parameters, variable number thereof
- Comments in XML

C# features (cont'd)

- Objects on the stack (structs)
- Rectangular arrays
- Enumerations
- Visibility modifiers
- `goto`
- Versioning
- Component-based programming (properties, events)
- Delegates
- Indexers
- `foreach` statement
- Boxing/unboxing
- Attributes (metadata)

C# future features

- Generics (next step from C++ templates)
- λ-functions as “anonymous methods”
- Type inference!!
- Iterators (`foreach`+`IEnumerator`)
- Partial types
- Static classes
- Property accessor accessibility
- `#pragma warning`
- Nullable types

The End.