Cross-platform IL code manipulation library for runtime instrumentation of .NET applications

master thesis subject for
Markus Gaisbauer (0256634)
in cooperation with
dynaTrace software GmbH

July 5, 2007

Abstract

The .NET Runtime is based on a virtual machine that executes an intermediate language (IL) which is independent of a specific platform or source language. IL code manipulation is a well known technique for instrumenting existing applications in order to enable debugging, profiling or aspect oriented programming. Objective of this master thesis is to develop a library that allows easy-to-use manipulation of .NET IL code at runtime.

Motivation

Code instrumentation of binaries has been around for more than a decade and became very popular with the introduction of Java. The small and easy-to-understand instruction-set of the Java Virtual Machine (Java Bytecode) considerably simplified the manipulation of binaries. Java technologies such as custom class loaders and HotSwap further increased the popularity of the technique as they made it possible to delay instrumentation until load or even runtime.
.NET is based on the same principles as Java and uses a very similar intermediate instruction set, called Common Intermediate Language (CIL) [1]. .NET binaries conform to the open ECMA standard 335 [2] and therefore can be read and manipulated by third party applications. The .NET Runtime supports runtime instrumentation via the so called Profiling API. Similar to Java, classes and methods can be instrumented while loading them into memory.
There are a number of existing libraries for .NET IL code instrumentation, but none of them can currently be considered satisfactory for runtime instrumentation. Among the problems with existing libraries are:

- **No support for runtime instrumentation via the Profiling API:** At runtime, metadata can only be manipulated incrementally, and identities for existing parts of an assembly (tokens) must not be recalculated after an assembly has been loaded into memory.

- **Lacking support for .NET features**: Different libraries support different things, but none of them fully supports all the features defined in ECMA 335. Among the features missing from current libraries are: reading and writing of assemblies containing unmanaged code (mixed-mode) or multiple modules as well as support for CLR 2.0 features such as generics.
- **Running on top of .NET:** Not all tools that analyze or manipulate .NET assemblies are necessarily based on .NET themselves or are running on operating systems where a .NET framework is available. This is especially true for tools supporting binaries of multiple platforms such as Java and .NET.
- **Performance:** Runtime instrumentation is often part of interactive tools such as monitoring or diagnosis tools. Speed and memory consumption is especially important in this context.

Objective

The objective of this master thesis is the development of a cross-platform library, that enables load time manipulation of .NET IL with low overhead and a simple, high level API. The library should especially satisfy all of the following requirements:

Functional Requirements

1. Reading, manipulating and writing of .NET Assemblies in-memory and on hard disk.
2. Support of all .NET Framework versions (1.1, 2.0, 3.0), including features like:
   - Multi-Module Assemblies
   - Mixed-Mode Assemblies
   - x64 Assemblies
3. Support .NET Metadata parsing and manipulations
   - High-performance metadata extraction
   - Adding, deleting, replacing of .NET Metadata, like Local Variables, Methods, Fields, Attributes
4. Support .NET IL Code parsing and manipulations
   - Adding, deleting, replacing of arbitrary IL Instructions
   - Reordering of IL instructions
   - Exception Handling
5. Integration with .NET Profiling API
   - No modification of unaffected metadata of the original assembly (especially tokens)
   - Support IL Code and .NET Metadata as used in .NET Profiling API

6. Different granularity of manipulations:
    - Method
    - Class
    - Module
    - Assembly

Non-functional requirements

Performance is critical because the library will be used by software that is interacting with human users. The library should achieve a good trade-off between speed, memory consumption and ease of use.

Approach

To achieve the objectives mentioned above, I will proceed in the following order:

1. **Research:** Before designing and implementing the library I will have to do research about the following topics:
    - **IL Code and existing instrumentation libraries:** What information is stored inside .NET assemblies and how is it encoded in binary files? How do existing libraries approach manipulation of assemblies?
    - **.NET Profiling API:** How can the API be used to instrument code at runtime? What is required from an instrumentation library to simplify this task?
    - **dynaTrace Diagnostics [10]:** What are typical use cases for IL manipulation? What are the major drawbacks of existing instrumentation libraries in this context?
2. **Develop a set of test cases:** To ensure the quality of the instrumentation library, I will develop an extensive automated test suite that will cover:
    - Processing different assembly formats (multi-module, x64...)
    - Processing assembly content (unmanaged code, embedded resources...)
    - Processing metadata
    - Processing IL code
3. **Implement assembly and metadata processing:** Determine which approach promises to deliver the best trade-off between speed, memory consumption and ease of use.
4. **Implement IL code processing:** This is the core of any code manipulation library. For the user of the library it should both be easy to manipulate IL code directly as well as build additional, higher level layers on top of the library for common usage patterns.

5. **Functional, performance and integration testing:** Finally I will have to ensure that the library fulfills all its functional and non-functional requirements and is ready to be used in production.

Related Work

1. **Mono Cecil** (C#): Cecil [3] is an existing open-source IL code manipulation library for .NET and is currently used by dynaTrace Diagnostics for all .NET related instrumentation and metadata extraction. It was designed for offline instrumentation (at compile or link time) of assemblies. Assemblies are represented internally using an abstract and unfortunately quite memory hungry object model.
2. **PostSharp** (C#) : PostSharp [4] is a very recent open-source project that enables aspect oriented programming in .NET. All IL code transformations are based on its own transformation library. Assemblies are represented in memory using lightweight data structures on top of the original binary.
3. **RAIL - Runtime Assembly Instrumentation Library** (C#) : RAIL [5] was one of the first public code manipulation libraries for .NET and was developed as a research project part of Microsoft ROTOR. It is based on Mono code for reading assemblies and Reflection.Emit for writing assemblies. It is now old and unsupported.
4. **.NET Profiling API and SSCLI** (COM, C++) : The Profiling API [6] can be used to exchange IL code and modify metadata at load time [7]. It works directly on internal data structures of the runtime. Shared Sources Common Language Infrastructure (SSCLI) [8] is an implementation of the CLR and is available with source code from Microsoft.
5. **BCEL - Byte Code Engineering Library** (Java) : BCEL [9] is a powerful and currently also the most popular instrumentation library for Java. It is used in a wide range of applications, including dynaTrace Diagnostics. It can be used as a reference for a library that is both easy-to-use and performant.

References

[1] Serge Lidin. *Expert .NET 2.0 IL Assembler*, APress, 2006

[2] *Ecma 335 Common Language Infrastructure (CLI)*
http://www.ecma-international.org/publications/standards/Ecma-335.htm

[3] *Mono Cecil*
http://www.mono-project.com/Cecil

[4] *PostSharp, a .NET post-compiler: AOP and more*
http://www.postsharp.org/

[5] *RAIL: a Runtime Assembly Instrumentation Library*
http://rail.dei.uc.pt/

[6] *Under the Hood - The .NET Profiling API and the DNProfiler Tool*
http://msdn.microsoft.com/msdnmag/issues/01/12/hood/

[7] *Rewrite MSIL Code on the Fly with the .NET Framework Profiling API*
http://msdn.microsoft.com/msdnmag/issues/03/09/NETProfilingAPI/default.aspx

[8] *Microsoft Shared Source Common Language Infrastructure*
http://research.microsoft.com/sscli/

[9] *BCEL - Byte Code Engineering Library (BCEL)*
http://jakarta.apache.org/bcel/

[10] *dynaTrace Diagnostics*
http://www.dynatrace.com