

Self-contained .NET Assemblies für eingebettete Systeme

Bernhard Rabe
Fachgebiet Betriebssysteme und Middleware
Hasso-Plattner-Institut an der Universität Potsdam
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam
bernhard.rabe@hpi.uni-potsdam.de

Überblick

Die Softwareentwicklung für eingebettete Systeme erfordert eine Anpassung bestehender Frameworks an die speziellen Eigenschaften und begrenzten Ressourcen. Middleware-Technologie hat im Desktopbereich eine stetig wachsende Bedeutung in der Softwareentwicklung. An eingebettete Systeme angepasste Middleware-Frameworks sind im Funktionsumfang eingeschränkt um Ressourcenbegrenzungen wie Speicherplatz oder Prozessorleistung zu handhaben. Für eine optimale Speichernutzung kann das Verfahren des statischen Linkens aus der Compilertechnik, angewendet auf Middleware-Technologie, den minimalen Footprint für die Ausführung von Middleware-Anwendungen auf speicherbegrenzten eingebetteten Systemen verringern. Das Verfahren kann in verschiedenen Stufen abhängig von den Systembeschränkungen adaptiv angewendet werden. Die interne Organisation von .NET Assemblies lässt die Anwendung dieses Verfahrens prinzipiell zu. In diesem Beitrag sollen mögliche Anwendungsszenarien aufgezeigt und technische Realisierungen, die im Rahmen des Micro.NET Projekts entstehen, skizziert werden.

Einleitung

Die Klasse der eingebetteten Systeme zeichnet sich durch die eingeschränkten Möglichkeiten der Interaktion mit der Umwelt aus. Diese Systeme haben oft begrenzte Ressourcen: nicht nur begrenzten Speicher oder geringe Prozessorleistung, sondern auch Batterielaufzeit oder Netzwerkkommunikationsbandbreite. Viele dieser Einschränkungen haben gegenseitige Abhängigkeiten, so hat verstärkte Netzwerkkommunikation in der Regel auch einen höheren Stromverbrauch zur Folge. Die Produktzyklen für eingebettete Systeme werden immer kürzer und erfordern neue Wege der Softwareentwicklung für diese Klasse von eingebetteten Systemen. Middleware-Frameworks mit ihren umfangreichen Bibliotheken erlauben immer wiederkehrende Aufgaben schneller zu implementieren.

Für Desktop-Systeme sind Middleware-Frameworks wie .NET[Net06] oder Java[J2SE06] verfügbar. Für bestimmte Klassen von eingebetteten Systemen (PDA, Smartphone) sind angepasste Versionen z.B. Java Micro Edition (J2ME)[J2ME06] oder .NET Compact Framework verfügbar.

Diese Technologien erfordern in der Regel mehr Hardwareressourcen oder sind aus anderen Gründen für die Mehrzahl von eingebetteten Systemen nicht verfügbar.

Die Verwendung der virtuellen Maschinenteknologie für die Ausführung von Programmen vereinfacht die Wiederverwendung von Code und ermöglicht eine vorhersagbare sichere Ausführung. Diese auf Desktop-Systemen erreichten Eigenschaften lassen sich in der Regel nicht auf eingebettete Systeme übertragen. Das J2ME Framework und das .NET Compact Framework schränken die möglichen Anwendungen bzw. die erlaubten APIs der Standardbibliotheken stark ein.

Im Rahmen des Micro.NET Projekts wird durch Proof Of Concept Implementierungen untersucht wie .NET Framework Anwendungen für speicherbegrenzte eingebettete System verfügbar gemacht werden können.

Die Idee bei Micro.NET ist es die Funktionalität der Middleware-Framework nicht durch eine reduzierte Klassenbibliothek einzuschränken, sondern eine Größenreduzierung von Middleware-Anwendungen für eingebettete Systeme zu erreichen, in dem die benötigten Funktionen der Klassenbibliothek Teil der Anwendung werden. Das Verfahren ist mit dem statischen Kompilieren von nativen Programmen vergleichbar.

.NET Plattform

Die ECMA/ISO *Common Language Infrastructure* (CLI)[Cli03] spezifiziert die Architektur, das Typsystem, die Instruktionen sowie die Bibliotheken und Profile. Implementierungen dieses Standards sind durch das Microsoft .NET Framework, die SSCLI[J2ME06] Sun Microsystems, "Java 2 Platform, Micro Edition (J2ME)", <http://java.sun.com/j2me/> [Mic06] sowie durch das Mono Projekt verfügbar.

Die CLI ist ein auf virtuellen Ausführungsumgebung basierende Programmiersprachenunabhängige *stack*-basierte Middleware-Plattform. Der Standard enthält auch eine Beschreibung der Klassenbibliotheken die von den CLI Profilen unterstützt werden müssen. Im CLI sind zwei Profile, das *Kernel*-Profil und das *Compact*-Profil, sowie zusätzliche Bibliotheken die in keinem Profil enthalten sind. Die kleinste CLI-konforme Profile ist das *Kernel*-Profil, welches die *Base Class Libraries* (BCL) sowie die Laufzeitbibliothek umfasst. Der Umfang der Klassenbibliothek des Kernel-Profiles ist jedoch für viele eingebettete Systeme mit begrenzter Speicherausstattung zu umfangreich.

Der CLI-Standard unterscheidet Assemblies und Module als Container CLI-Inhalte.

Eine Assembly umfasst eine oder mehrere Dateien, die eine funktionale Einheit bilden. Eine Assembly kann Referenzen auf andere Assemblies oder Module enthalten. In der Regel enthält jede Assembly eine Referenz auf die Assembly **mscorlib**, da diese alle Basis-Typen des in der CLI spezifizierten Common Type System (CTS) enthält die direkt von dem *Virtual Execution System* (VES) unterstützt werden. Referenzen auf externe Assemblies/Module erfolgen auf Basis vom Klassenmethoden bzw. Feldern.

Ausführung von IL-Code

Um in einer Assembly enthaltenen IL-Code auszuführen, muss die virtuelle Maschine die Position des IL-Codes in der Datei ermitteln. Das erfordert mehrere Zugriffe auf die Metadaten zur Folge. Die Struktur der Metadaten in .NET Assemblies ist in Tabellen organisiert. Ausgewählte Tabellenfelder enthalten Verweise auf andere Tabelleneinträge, so dass mitunter mehreren Verweisen gefolgt werden muss, bis die gewünschte Information zur Verfügung steht. Am Beispiel eines Methodenaufrufs eines in einer externen Assembly definierten Typs sollen die notwendigen Schritte verdeutlicht werden.

Für die Ausführung einer externen Assembly definierten Methode (Abbildung 1) muss zunächst die Position des IL-Codes ermittelt und geladen werden. Das Sprungziel einen Methodenruf (*call*, *calli*, *callvirt*) wird durch ein Metadaten-Token bestimmt.

Liegt die Methode extern in einem Modul oder einer Assembly vor, so verweist das Token auf einen Eintrag in der *MemberRef*-Tabelle. Jede Zeile dieser Tabelle verweist in die *TypeRef*-, *ModuleRef*-, *Method*-, *TypeSpec*- oder *TypeDef*-Tabelle. Für das Beispiel werden nur *TypeRef*- und *ModuleRef*-Verweise betrachtet.

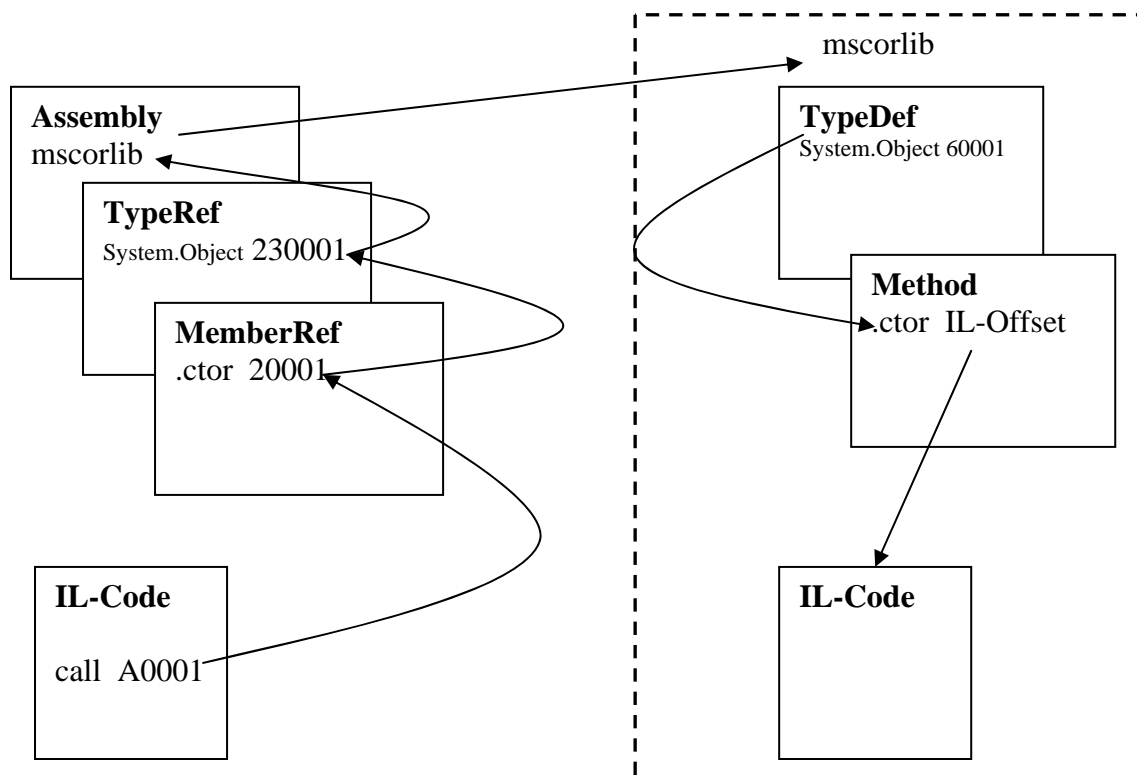


Abbildung 1: Aufruf einer externen Methode

Ein Eintrag in die *ModuleRef*-Tabelle benennt ein externes Modul, welches zur aktuellen Assembly gehört. Im Modul muss dann mit Hilfe des Namens aus der *MemberRef*-Tabelle die Position des IL-Codes in der Modul-Datei bestimmt werden.

Der Eintrag in der *TypeRef*-Tabelle enthält den Typnamen und den Namespace sowie einen Verweis auf einen *ModuleRef*-, *AssemblyRef*- oder *TypeRef*-Tabelleneintrag. Ein Eintrag der *AssemblyRef*-Tabelle enthält den Namen der Assembly, deren Versionsnummer, einen Public-Key. Mit diesen Informationen kann die referenzierte Assembly geöffnet werden, mit Hilfe des Typnamen und dem Methodennamen die Position des IL-Codes bestimmt werden.

Im einfachsten Fall, für eine Methode in einem externen Modul definiert ist, benötigt mindestens 4 Schritte bis die Position des IL-Codes im Modul bestimmt ist (*MemberRef*, *ModuleRef*, *TypeDef* im Modul, *MethodDef* im Modul). Im Gegensatz dazu erfordert der Aufruf einer Methode im aktuellen Modul erfordert einen Zugriff auf die *Method*-Tabelle um die Position des IL-Codes zu lesen (rechter Kasten im Abbildung 1).

Micro.NET

Im Rahmen des Micro.NET Projekts werden Möglichkeiten untersucht um .NET Programme auf kleinen eingebetteten Systemen auszuführen. Die aktuell verfügbaren CLI Implementierungen sind nicht geeignet um auf Systemen mit begrenzten Speicherressourcen und geringer Prozessorleistung eingesetzt zu werden. Middleware-Laufzeitsysteme für kleine Systeme in Form von J2ME und .NET Compact Framework erreichen die Größenreduzierung durch Beschränkung der zur Verfügung gestellten API. Programme nutzen häufig nur einen sehr kleinen Teil dieser API, so dass durch die nicht genutzte API unnötig Speicher belegt wird. Ein großer Teil von .NET Programmen durch die enthaltenen Metadaten verursacht wie in [Cost05] untersucht wurde.

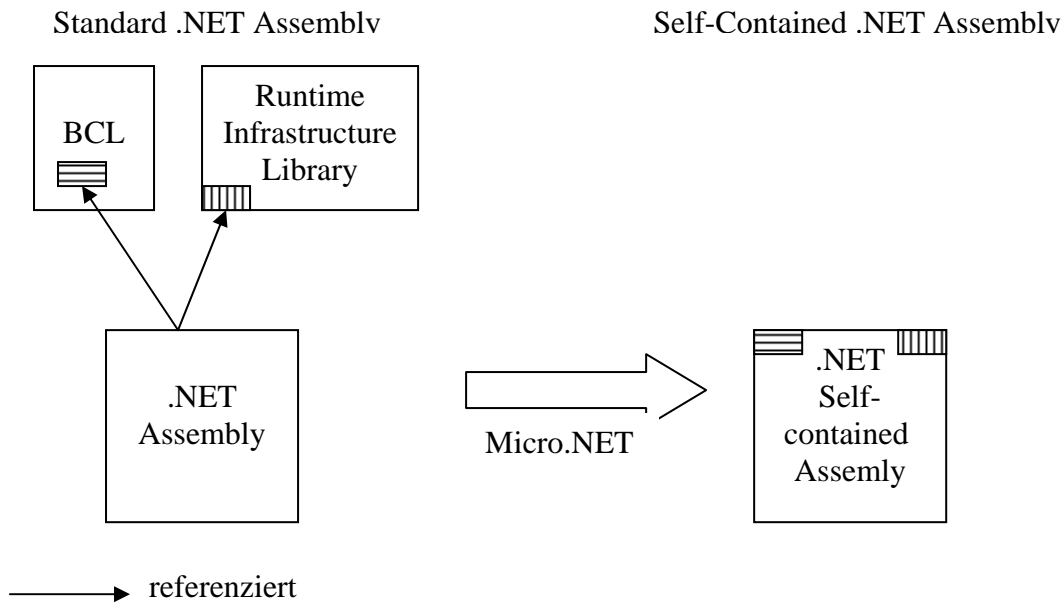


Abbildung 2: Self-Contained Assemblies

Die Idee des Micro.NET Projektes ist .NET Assemblies zu erzeugen die *Self-contained* (Abbildung 2) sind. Diese enthalten keine Referenzen auf externe Assemblies oder Module. Das trifft insbesondere auf die mscorlib zu. Die Architektur von .NET Assemblies und Modulen erlaubt es prinzipiell Assemblies ohne Einträge in *TypeRef*-Tabelle zu erzeugen. Das kann erreicht werden in dem die referenzierten Typen durch den Import dieser Typen in die Assembly importiert werden. Die so modifizierten Assemblies enthalten nur den unmittelbar notwendigen Typen.

Werkzeuge zur Erzeugen von *Self-contained* Assemblies werden im Rahmen des Micro.NET entwickelt. Das ist mit Hilfe von Bibliotheken zur Assembly-Manipulation z.B. PERWAPI[PER06] möglich. Dieses Verfahren verringert die Größe der Metadaten nur geringfügig. Da es keine externen Referenzen mehr vorhanden sind, ist es auch nicht notwendig die textuellen Typ-, Methoden- und Feldnamen zur Verfügung zu haben. Typen, Methoden und Typen können mit Zeichenketten mit einem Zeichen Länge beginnend umbenannt werden. Dieses Verfahren wird mit einem anderen Ziel z.B. zur Erschwerung der Disassemblierung benutzt. Als weitere Größenoptimierung kann eine vollständige Programmanalyse benutzt werden um nicht verwendete IL-Code zu identifizieren und zu entfernen.

Self-contained Assemblies

Die Reduzierung des für die Ausführung minimal notwendigen .NET Codes ist nur eine Eigenschaft der so optimierten Assemblies. Da die Verhaltensbeschreibung für alle IL-Codes in der CLI beschrieben ist, ist das Verhalten von *self-contained*-Assemblies durch den identischen IL-Code auf jeder CLI-konformen Ausführungsumgebung von Klassenbibliotheken unabhängig. Eine Ausnahme bilden die durch die virtuelle Maschine implementierten nativen Codes so wie Aufrufe von nativen Bibliotheken. Da interne Aufrufe innerhalb der Laufzeitbibliothek spezifisch für jede Implementierung einer virtuelle Maschine ist muss eine Abbildung von internen Klassenbibliotheksrufern auf native Funktionen getroffen werden. Experimentelle Untersuchungen müssen zeigen in welcher Größenordnung der Speicherplatzbedarf für *self-contained* .NET Assemblies durch Reduzierung der Indirektionen für externe Referenzen ausfallen.

Zusammenfassung

Existierende Middleware-Frameworks adressieren nicht die Anforderungen von Speicherbegrenzten eingebetteten Systemen. Einen großen Teil der Laufzeitumgebung nehmen die umfangreichen Klassenbibliotheken ein. Das Konzept der *Self-contained* Assemblies kann die Verfügbarkeit von .NET auf speicherbegrenzten eingebetteten Systemen erhöhen.

Die Größe der in .NET Assemblies enthaltenen Metadaten kann reduziert werden, da Methodenreferenzen nicht mehr auf Namen basieren, sondern auf einen Tabellenindex verweisen.

Eine weitere Eigenschaft der *self-contained* Assemblies ist die Vorhersagbarkeit des Verhaltens aufgrund des IL-Codes, welcher unabhängig von externen Bibliotheken ist. Es ist jedoch noch Arbeit zu leisten um die Vorteile durch die Reduzierung von Indirektionen bei Methodenaufrufen zu quantifizieren.

[Cli03] “Information technology — Common Language Infrastructure”, ISO/IEC 23271:2003(E) First edition

[Cop06] Microsoft Corporation, “.NET Compact Framework”, <http://msdn.microsoft.com/netframework/programming/netcf/>, 2005

[Cost05] Roberto Costa, Erven Rohou, “Comparing the size of .NET applications with native code”, Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, 2005

[J2SE06] Tim Lindholm, Frank Yellin, “The Java(TM) Virtual Machine Specification (2nd Edition)”, Addison-Wesley Professional, 1999

[J2ME06] Sun Microsystems, “Java 2 Platform, Micro Edition (J2ME)”, <http://java.sun.com/j2me/>

[Mic06] Microsoft Corporation “Shared Source Common Language Infrastructure”, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/Dndotnet/html/mssharsourcecli.asp>, 2006

[Mon06] The Mono Project Homepage, <http://www.mono-project.com>, 2005

[Net06] Microsoft Corporation “.NET Framework Developer Center”, <http://msdn.microsoft.com/netframework>, 2005

[PER06] Queensland University of Technology, PERWAPI, <http://www.plas.fit.qut.edu.au/perwapi/>, 2006