

A Microsoft .NET Front-End for GCC

Martin v. Löwis

Hasso-Plattner-Institut
für Softwaresystemtechnik GmbH
Postfach 900460
+49 331 5509 239

Martin.vonLoewis@hpi.uni-potsdam.de

Jan Möller

Hasso-Plattner-Institut
für Softwaresystemtechnik GmbH
Postfach 900460

Jan.Moeller@hpi.uni-potsdam.de

ABSTRACT

In the past, embedded systems developers have been severely constrained in their choice of programming languages. Recent advancements in processing power and memory availability allow for new techniques. We present an extension to the GNU Compiler Collection (GCC) that offers the expressiveness of all Microsoft .NET languages to embedded systems.

Keywords

Common Intermediate Language, GNU Compiler Collection, GCC.

1. INTRODUCTION

Embedded systems are known for the severe resource constraints in terms of memory size and clock speed. For that reason, developers traditionally use assembler language and C for such systems [Bar99]. Compared to current desktop and server programming languages such as Java, C#, Python, Visual Basic, and others, the typical development environment is tedious to use, and the development is less productive.

There are two primary aspects of the “desktop” programming languages that we consider interesting for embedded developers as well: object-orientation and safety. With object-orientation, the software may become more maintainable, as the encapsulation mechanisms allow for better modularization and abstraction.

By “safety”, we refer to the reliability aspects that are typically associated with interpreters: the run-time system of the language will make sure that invalid operations (such as out-of-bounds accesses to arrays) cause a well-defined program termination (typically through an exception), instead of causing undefined behavior (such as memory corruption). Safe programming languages reduce the number of bugs that remain in the software after testing, as errors are reliably detected. They also simplify the process of locating the source of a bug, as the error is often detected right after it occurred.

Unfortunately, both object-orientation and safety come at significant run-time cost. Interpreters execute program code much slower than similar compiled programs. Alternatively, just-in-time compilation is

used to speed-up execution [Kra98]. Unfortunately, just-in-time compilation is itself expensive and causes unpredictable run-time behavior. Furthermore, a just-in-time compiler needs to be developed for each new target architecture.

As an alternative, we present an approach which allows static compilation of .NET programs for embedded targets. We briefly discuss different aspects of this solution in the remainder of this paper.

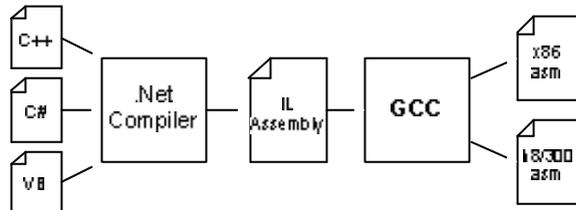
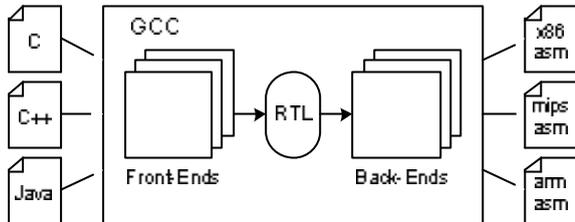
2. GCC

The GNU Compiler Collection integrates different programming languages (C, C++, Java, Ada, ...) for various microprocessor architectures [GS04]. Among the supported targets are many desktop and embedded processors; GCC is known for relatively easy extensibility to new architectures [Sta95]. While it originally focused on the C language only, it has recently been extended to object-oriented and safe languages, such as Java [Bot97].

In GCC, the source code of the input language is transformed into an intermediate representation¹, which is then processed in optimization passes. The result of the compilation is then output as an assembler source code file for the target machine. This assembler file is processed with assemblers, loaders, etc. for the target system to produce an executable program.

¹ More precisely, there are two internal representations: the *tree* structure, and the Register Transfer Language (RTL).

The design of GCC is engineered towards extensibility. Support for new microprocessors can be added relatively easy by describing the processor in a *machine definition*. Using this machine definition, the compiler can convert the internal representation (RTL) into assembler code of the target system. This assembler code is then further processed in an assembler to object files, and eventually combined with a linker into executable files and libraries.



4. Target Systems

In principle, it is possible to support all features of the .NET platform that don't require dynamic insertion of behavior. That is, all instructions of the intermediate language can be converted into sequences of assembler instructions of the target system. Through generation of data structures into the resulting assembler code, introspection of objects is possible, using the standard APIs. Even dynamic loading of assemblies is possible, as long as the assembly to be loaded was compiled using GCC in advance.

For the remaining features, we plan to support interoperability with the Mono software [DB04]. To achieve an integration of Mono, we need to use the same application binary interface (ABI) that mono uses, with respect to calling conventions, and representation of meta-data in memory.

At the same time, we also like to target embedded systems. At the moment, our primary target is the Lego Mindstorms hardware [Sat02], which uses the Renesas H8/300 processor [Ren03]. On this system, memory is limited. For our .NET implementation, this means primarily that we have to be very selective in the subset of the .NET library that we can support – the entire platform library just won't fit into 32k of main memory. In this environment, we may also have to accept further limitations. However, depending on the application's needs, we believe that all features of the virtual machine can be supported. The more challenging features are floating point computations (which require emulation in software, as the chip has no hardware floating point support), exception handling, and garbage collection. At this point, we cannot yet predict what costs in terms of memory and processor cycles these features will require.

In addition to the Lego Mindstorms, we also target Windows CE; in particular CE PC.

5. Current Status

Currently, only a small fraction of the CIL features are supported, namely

- primitive data types (bool, byte, short, int, float, double)
- classes, including static and instance attributes and properties, as well as inheritance,
- static and instance methods, including parameters, local variables, and constructors,
- arrays and strings,
- delegates
- arithmetic operations, and
- control flow operations (conditional and unconditional branch instructions).

Using this subset, we have been able to develop small control programs for the Lego Mindstorms platform.

On the Windows CE system, we were able to create control programs which meet hard real-time constraints.

Work to provide additional features, such as interfaces, and exception handling, is in progress. Our current implementation is available from <http://www.dcl.hpi.uni-potsdam.de/research/lego.NET/release.htm>.

6. Related Work

Cygnus Solutions (now Redhat) has developed a Java front-end [GCJ05], supporting both Java source code and byte code. The CIL front-end has taken much inspiration from the latter.

Microsoft currently develops the Phoenix framework [Lef04], which appears to be similar in architecture to GCC, and also appears to contain a .NET front-end. Very little information about Phoenix has been published so far.

7. REFERENCES

- [Bar99] M. Barr. Programming Embedded Systems in C and C++. O'Reilly, 1999.
- [Bot97] P. Bothner. A Gcc-based Java implementation. IEEE Compcon'97.
- [DB04] E. Dumbill, N.M. Bornstein. Mono: A Developers Notebook. O'Reilly, 2004.
- [Dot05] DotGNU Portable.NET. <http://www.dotgnu.org>
- [ECM02a] ECMA-335. Common Language Infrastructure, Partition III: CLI Instruction Set. Dec. 2002.
- [ECM02b] ECMA-334. C# Language Specification. Dec. 2002.
- [GCJ05] The GNU Compiler for the Java™ Programming Language. <http://gcc.gnu.org/java>
- [GS04] B.J. Gough, R.M. Stallman(Forword). An Introduction to GCC. Network Theory Ltd, 2004.
- [Kra98] A. Krall. Efficient JavaVM Just-in-Time Compilation. PACT, 1998.
- [Lef04] J. Lefor. Phoenix as a Tool in Research and Instruction. July, 2004.
- [Ren03] Renesas Technology Corp..H8/300 Programming Manual. 2003.
- [Sat05] J. Sato. Jin Sato's Lego Mindstorms. No Starch Press, San Francisco, 2002.
- [Sta95] R.M. Stallmann. Using and Porting GNU CC. Free Software Foundation, 1995.

