PEAC: Ein partieller Evaluator für eingebettete Software

Michael Jung* Sorin Alexander Huss* Matthias Deegener+

*Integrierte Schaltungen und Systeme
Technische Universität Darmstadt
{mjung | huss}@iss.tu-darmstadt.de

+GM Powertrain Europe
Product Eng. Control Systems
matthias.dr.deegener@de.gm.com

Zusammenfassung

Die Wiederverwendung erprobter Software-Komponenten stellt einen pragmatischen Ansatz dar, um eine hohe Zuverlässigkeit, wie sie insbesondere im Bereich der Automobilindustrie benötigt wird, zu erreichen. Bei der Entwicklung eingebetteter Software-Komponenten haben sich Code-Generatoren bewährt, um die für die Wiederverwendbarkeit notwendige Flexibilität mit der Effizienz spezialisierter Software zu kombinieren. Die Implementierung entsprechender Generatoren erhöht als zusätzlicher Schritt jedoch die Komplexität des Entwurfsprozesses. Partielle Evaluierung ist eine Technik, die unter anderem ein gewisses Maß an Automatisierung bei der Entwicklung von Generatoren ermöglicht. Im Rahmen des Workshops *Zuverlässigkeit in Eingebetteten Systemen* wurde über *PEAC* berichtet, einen partiellen Evaluator für ANSI C, der unter Berücksichtigung der besonderen Anforderungen eingebetteter Software entworfen wurde.

Schlüsselwörter: Generatives Programmieren, Partielle Evaluierung, Reuse, Eingebettete Software.

1 Einleitung

Die Wiederverwendung von Software-Komponenten, die bereits ausgiebig getestet wurden und sich eventuell bereits im Einsatz bewährt haben, hat einen positiven Effekt auf die Zuverlässigkeit und Qualität des Produktes und die Effizienz des Entwicklungsprozesses. Damit ein Software-Modul in Systeme unterschiedlicher Charakteristik integriert werden kann, muss es einen hohen Flexibilitätsgrad aufweisen. Die in der Praxis übliche Umsetzung dieser Qualität beansprucht jedoch *zur Laufzeit* Resourcen des eingebetteten Systems. So muss z.B. die anwendungsspezifische Konfiguration der Software-Module in irgendeiner Form gespeichert und typischerweise während des Systemstarts angewandt werden.

Die bei eingebetteten Systemen oft üblichen hohen Stückzahlen resultieren in einem hohen Druck die Kosten pro Einheit zu minimieren. Dadurch verursacht finden wir heute die Situation vor, daß im Design eingebetteter Software selten Techniken zur Wiederverwendung von Software-Modulen berücksichtigt werden. Eingebettete Software ist oft in hohem Maße auf die gegebene Anwendung und Hardware spezialisiert, was im Umkehrschluss die Wiederverwendung in einem anderen Kontext behindert.

Trotzdem vertreiben Firmen erfolgreich Software-Komponenten für eingebettete Systeme, die charakteristischerweise nicht im Quelltext, sondern oft in Form von Code-Generatoren ausgeliefert werden. Im Forschungsgebiet *Generative Programming* (siehe [1]), das sich mit dem Generator-basierten Software-Entwurf beschäftigt, wird neben einer naheliegenden Methodik, die auf Code-Schablonen basiert, unter anderem die Verwendung einer Meta-Sprache vorgeschlagen. Nachteilig an diesen Ansätzen ist zum einen die Tatsache, daß der Generator in einer anderen Sprache implementiert wird als das zu generierende Software-Modul. Zum anderen muss bereits sehr früh im Entwicklungsprozess festgelegt werden, welche Parameter als statisch, also für eine gegebene Applikation als konstant anzusehen sind und welche Parameter dynamisch, also zur Laufzeit variabel sein werden. *Partielle Evaluierung* (siehe [3]) ist eine Technik, die angewandt auf das Gebiet des generativen Programmierens Lösungsansätze für diese Probleme bietet.

2 Partielle Evaluierung

Partielle Evaluierung (auch Programm Spezialisierung), die in [2] erstmals als Ansatz in der Informatik vorgeschlagen wurde, ist eine Transformations-Technik, die auf folgender Idee basiert: Sind ein Program p sowie ein Teil d_s seiner Eingaben gegeben (die *statischen* Parameter), so kann ein *Residualprogramm* p_{res} abgeleitet werden, welches sich im Bezug auf den verbleibenden Teil der Eingaben d_d (die *dynamischen* Parameter) identisch zu p verhält.

$$p_{res} = PE(p, d_s) \Rightarrow \forall d_d . p(d_s, d_d) = p_{res}(d_d) \qquad (1)$$

Da prinzipiell alle Berechnungen, deren Ergebnisse ausschließlich von den statischen Parametern d_s abhängig sind, bereits während der partiellen Evaluierung ausgewertet werden können, beansprucht das Residualprogramm p_{res} potentiell weniger Speicher- und Rechenresourcen als das Original.

Sogenannte Offline-Evaluatoren wenden ein zweistufiges Verfahren an (siehe Abb. 1). In einer ersten Stufe wird mittels abstrakter Interpretation jeder

Ausdruck des Programs als *statisch* oder *dynamisch* markiert, je nachdem ob er ausschließlich anhand der statischen Parameter auswertbar ist. Aus dieser sogenannten *Binding-Time-Annotation* lässt sich ein Generator ableiten, der hier *Generating-Extension* genannt wird (vgl. [6]).

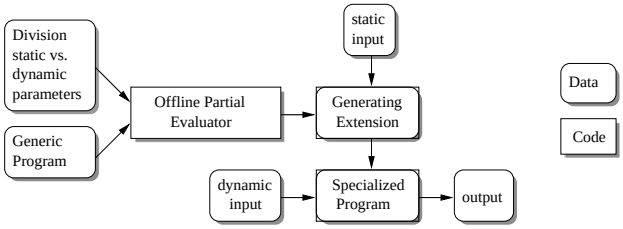

Abbildung 1: Partielle Evaluierung (nach [3])

Die zweite Stufe der partiellen Offline-Evaluierung ist die Ausführung des Generators, dem die Werte der statischen Variablen als Eingabe übergeben werden. Während statische Ausdrücke und Anweisungen bereits zur Laufzeit ausgewertet werden, werden dynamische Konstrukte mittels Pretty-Printer als Teil des Residualprogramms ausgegeben. Werte statisch auswertbarer Ausdrücke erscheinen im Residualprogramm in ihrer textuellen Repräsentation als Konstanten.

3 Spezielle Anforderungen

Bevor an unserem Institut mit dem Design eines partiellen Evaluators begonnen wurde, der die Anforderungen im Hinblick auf eingebettete Software berücksichtigt, wurde eine Fallstudie durchgeführt (s. [5]). Auf diese Weise sollte festgestellt werden ob der Ansatz prinzipiell sinnvoll ist und welche Schwachstellen sich im Kontext eingebetteter Software zeigen. Als partieller Evaluator wurde C-Mix/II verwendet (s. [7]). Als Fallbeispiel wurde die Implementierung eines Generator-basierten eingebetteten Betriebssystems gewählt, das der OSEK Spezifikation entspricht (s. [8]).

Die Fallstudie hat im Ergebnis gezeigt, dass ein mit Hilfe partieller Evaluierung erstellter Generator in der Qualität des erzeugten Codes einem manuell implementierten nicht nachstehen muss. Allerdings wurden einige Probleme deutlich, die sich durch die besonderen Anforderungen im Bereich eingebetteter Software ergeben.

Code Bloat - Im Allgemeinen ist es das Ziel partieller Evaluierung den größtmöglichen Teil eines gegebenen Programms bereits vor der Laufzeit auszuwerten, um einen maximalen Geschwindigkeitszuwachs zu erreichen. Dazu ist es notwendig zuzulassen, daß ein Programmsegment sich im Residualprogramm in mehreren spezialisierten Varianten wiederfindet. Somit ist das entsprechende Speicherabbild eventuell wesentlich umfangreicher als das des Originalprogramms. Da für eingebettete Systeme Programmspeicher genauso knapp ist wie Rechenleistung, gilt es dies zu verhindern. Im Beitrag [4] wird über zwei Ausprägungen dieses Problems sowie über entsprechende Lösungsansätze berichtet.

Partielle Cross-Evaluierung - Tatsächlich wird im Kontext eingebetteter Software "Partielle *Cross*-Evaluierung" benötigt: Während der partielle Evaluator und die Generating-Extension auf dem Rechner des Enwicklers ausgeführt werden, wird das Residualprogramm auf der Zielplattform ausgeführt. Dies ist insbesondere deshalb ein Problem, weil die Semantik von ANSI C nur im Kontext einer bestimmten Zielplattform eindeutig definiert ist. So ist zum Beispiel die Bit-Breite des `int`-Datentyps oder die Wahl der Binärdarstellung eines Wertes (Big-/Little-Endian) mit gutem Grund nicht standardisiert.

Deshalb ist es notwendig eine zusätzliche Abstraktionsebene einzuführen, die die Semantik der Zielplattform zur Laufzeit der Generating-Extension auf dem Entwicklungsrechner nachbildet.

4 PEAC

PEAC ist ein partieller Evaluator for ANSI C, der sich seit Ende 2003 am Fachgebiet ISS der TU Darmstadt in Entwicklung befindet und der die besonderen Anforderungen eingebetteter Software berücksichtigt.

An einem Beispiel wird in unserem Beitrag [4] gezeigt, wie generischer Code der den Kommunikationsmechanismus einer Peripherieanbindung abstrahiert und somit austauschbar macht, mittels PEAC spezialisiert werden kann. Das *text*-Segment des Kompilats der spezialisierten Variante hat im Beispiel nur 40% des Speicherbedarfs im Vergleich zum Kompilat des generischen Quelltextes.

Literatur

[1] Czarnecki, Krzysztof ; Eisenecker, Ulrich: *Generative Programming: Methods, Tools, and Applications.* Addison-Wesley, 2000. – ISBN 0–201–30977–7

[2] Futamura, Yoshihiko: Partial Evaluation of Computation Process - An Approach to a Compiler-Compiler. In: *Systems, Computers, Controls* 2 (1971), Nr. 5, S. 45–50

[3] Jones, Neil D. ; Gomard, Carsten K. ; Sestoft, Peter: *Partial Evaluation and Automatic Program Generation.* Prentice Hall, 1993. – ISBN 0–13–020249–5

[4] Jung, Michael ; Huss, Sorin A. ; Deegener, Matthias: PEAC: Ein partieller Evaluator für eingebettete Software. In: *Zuverlässigkeit in eingebetteten Systemen.* Aachen : Shaker Verlag, Oktober 2005. – ISBN 3–8322–4522–7, S. 9–21

[5] Jung, Michael ; Laue, Ralf ; Huss, Sorin A.: A Case Study on Partial Evaluation in Embedded Software Design. In: *3rd IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems*, IEEE Computer Society, 2005, S. 16–21

[6] Kutzmann, Daniel: *Design und Implementierung einer Binding Time Analyse für C*, Technische Universität Darmstadt, Diplomarbeit, Oktober 2004

[7] Makholm, Henning: Specializing C - An Introduction to the the Principles behind C-Mix/II / DIKU Copenhagen. 1999. – Forschungsbericht

[8] WG, OSEK O.: *OSEK/VDX Operating System Version 2.2.2.* `http://www.osek-vdx.org`, July 2004