15 Symphony Fallstudie: Hierarchische Reflexion Modelle

Rainer Koschke
Daniel Simon
University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
{koschke,simondl}@informatik.uni-stuttgart.de

15.1 Einleitung

In diesem Bericht stellen wir eine Fallstudie vor, die dem Prozessmodell für Symphony [3] folgt. Wir führen in diesem Zusammenhang eine Sichten getriebene Architekturrekonstruktion durch, bei der das hierarchische Reflexion Modell [4] zum Zuge kommt. Im folgenden werden die einzelnen Schritte von Symphony für die Validierung von zwei Open-Source Compiler gegen eine Compiler Referenzarchitektur aus der Literatur mit konkretem Inhalt gefüllt.

15.2 Symphony Designebene

Problem Elicitation Als erster Schritt wird bei Symphony die Aufgabe des Reengineering Projekts definiert. In unserem Fall ist dies der Vergleich einer Compiler Referenzarchitektur mit den Architekturen der beiden Open-Source C Compiler `sdcc` [8], einem Compiler für Mikrocontroller, und `cc1` [2], einem Teil der GNU Compiler Collection ist. Die Beteiligten an dem Prozess waren wir selbst; als Nebeneffekt erhofften wir uns eine Evaluation unserer Erweiterung des Reflexion Modells.

Concept Determination Bei der Bestimmung der benötigten Konzepte für die Architekturvalidierung wurden als nützliche Sichten die Modulsicht auf den jeweiligen Compiler, die konzeptuelle Sicht mit der hypothetischen Architektur sowie die Reflexion Modell Sicht ausgewählt. Der gewählte *Target Viewpoint* war in der ersten Iteration die Kombination aus Modul und Reflexion Sicht, in der zweiten Iteration haben wir die Modulsicht mit der hierarchischen Reflexion Sicht kombiniert. Die *Source* und `Target Viewpoints` sind in Abbildung 15.1 zusammengefasst. Bemerkenswert ist im Zusammenhang mit Symphony, dass bei der Anwendung der Reflexion Modelle auch explizit hypothetische Sichten benutzt werden.

Source Viewpoint	**Target Viewpoint**
dir *contains* dir	module *convergence* module
dir *contains* module	module *divergence* module
module *contains* declaration	module *convergence* module
declaration *depends-on*$_a$ declaration	module *contains* module
	module *depends-on*$_a$ module
	module *depends-on*$_h$ module

Abbildung 15.1: Viewpoints bei der Compiler Validierung

Iterationen Auf der Designebene von Symphony führten die Mängel des ursprünglichen Reflexion Modells zu einer Iteration, in der wir unsere Erweiterung des Reflexion Modells zum Einsatz brachten.

Hierarchisches Reflexion Modell Das ursprüngliche Reflexion Modell, das von Murphy and Notkin [5, 6, 7]

vorgeschlagen wurde, erlaubt es dem Reengineer, ein abgeleitetes oder vorgeschriebenes Architekturmodell eine Softwaresystems gegen ein automatisch aus den Quelltexten abgeleitetes Modell zu validieren. Dabei werden zunächst die Entitäten im Quellmodell auf das hypothetische Architekturmodell abgebildet. Im Anschluss daran werden die Diskrepanzen zwischen Quell- und Architekturmodell automatisch berechnet und können dem Benutzer in geeigneter Weise präsentiert werden.

Da das Modell keine hierarchischen Architekturen unterstützt, ist es für die Untersuchung großer Systeme nur bedingt geeignet. In [4] haben wir das Modell von Notkin u.a. um die Möglichkeit erweitert, auch mit hierarchischen Architekturen umzugehen.

15.3 Symphony Ausführungsebene

Data Gathering Das Aufsammeln der Daten gestaltet sich in diesem Fall als Extraktion von statische Abhängigkeiten aus den Quellen der Compiler. Die Extraktion wurde automatisiert mit Hilfe der Bauhaus Toolsuite [1] durchgeführt; unter anderem haben wir globale Deklarationen (von Typen, Variablen und Routinen), Module und Verzeichnisse sowie die in Abbildung 15.2 aufgezählten Beziehungen automatisch ermittelt.

Knowlegde Inference und Information Interpretation Sowohl die Berechnung des hierarchischen Reflexion Modells als auch der Abgleich der gewonnen Informationen über die Quellen kann interaktiv mittels der graphischen Benutzeroberfläche von Bauhaus durchgeführt werden. Das Reflexion Modells lässt sich auf natürliche Art durch einen Graph visualisieren.

Iterationen Auf der Ausführungsebene von Symphony gab es eine ganze Reihe von Iterationen. Diese wurde zum einen verursacht durch die Tatsache, dass beim Erstellen der Referenzarchitektur Fehler gemacht wurden. Als Konsequenz wies das Reflexion Modell unerwartete Diskrepanzen auf; Ergänzungen der Referenzarchitektur behoben diese Problem.

Weitere Iterationen rührten von der Tatsache, dass die Abbildung von Quellentitäten auf hypothetische Module zum Teil von Dateien auf kleinere Einheiten verfeinert werden musste. Eine Iteration verfeinerte das Quellmodell, in dem zusätzlich zu direkten Funktionsaufrufen auch Aufrufe von Funktionen über Zeiger einfließen. Dazu ist allerdings eine Zeigeranalyse notwendig, deren (notwendigerweise konservative) Ergebnisse noch manuell geprüft werden mussten.

15.4 Erfahrungen mit dem Reflexion Modell

Das Validieren einer Referenzarchitektur macht bei großen System nur Sinn, wenn man hierarchische Reflexion einsetzt. Das manuelle Erstellen der Referenzarchitektur ist aufwändig, ebenso braucht man viel Erfahrung mit dem untersuchten System, um die Abbildung von Quellentitäten auf Modellentitäten durchzuführen. Beides sind allerdings Aufgaben, die sich viel einfacher gestalten, wenn sie bereits bei der Entwicklung der Softwaresysteme mit berücksichtigt werden.

Die anschließende Anwendung der Reflexion zum Zwecke der Validierung und Kontrolle der Implementierung ist automatisierbar und bietet auch Möglichkeiten, Fortschritte über mehrere Quellversionen zu verfolgen.

Referenz Typ	Beschreibung
static call	statically bound call of function
dynamic call	call through function pointer
access	use, set, or address-taken of a variable or record component
r-access	address-taken of a function
signature	type occurs in function signature
of-type	type of a variable or record component
local-var-of-type	function has local variable of type
based-on-type	one type uses another type for its declaration

Abbildung 15.2: Extrahierte Referenz Typen.

Literaturverzeichnis

[1] Bauhaus. `http://www.bauhaus-tec.com/`, Mai 2004.

[2] Free Software Foundation. The GNU Compiler Collection. `http://gcc.gnu.org/`.

[3] R. Koschke. View-Driven Software Architecture Reconstruction. In *Proc. of the WSR*, May 2004.

[4] R. Koschke and D. Simon. Hierarchical Reflexion. In *Proc. of the WCRE*, pages 36–45, Victoria, B.C., Nov. 2003.

[5] G. C. Murphy and D. Notkin. Reengineering with Reflexion Models: A Case Study . *IEEE Computer*, 30(8):29–36, Aug. 1997.

[6] G. C. Murphy, D. Notkin, and K. Sullivan. Software Reflexion Models: Bridging the Gap Between Source and High-Level Models. In *Proc. of the Third ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 18–28, New York, NY, 1995.

[7] G. C. Murphy, D. Notkin, and K. J. Sullivan. Software Reflexion Models: Bridging the Gap between Design and Implementation. *IEEE TSE*, 27(4):364–380, Apr. 2001.

[8] *sdcc*, a C Compiler for small devices. `http://sourceforge.net/projects/sdcc/`, June 2003.