Metaobjektprotokoll

Offene Software- und Metaebenen-Architekturen

Wir kennen alle das Problem, welches über kurz oder lang während des Gebrauchs einer Software auftritt, seien es Anwenderprogramme, wie Textsysteme, Software zur Unterstützung der Erstellung von Anwenderprogrammen, wie Graphik- oder Fenstersysteme, oder Programmiersprachen: die Grenzen der Software werden durch die Anforderungen der jeweiligen Anwendung erreicht. Softwareentwickler begegnen diesem Problem häufig durch ausufernde Funktionalität eines Programms. Dies konfrontiert den Anwender mit einer großen Zahl von Möglichkeiten und Optionen, die er im Regelfall nicht benötigt. Eine solche Software muß komplex sein und ist daher schwer handhabbar. Werden die Anforderungen jedoch nicht erfüllt, kommt es häufig dazu, daß Anwendungsprogrammierer selbst eine für ihre Dienste zugeschnittene Software schreiben.

Eine andere Möglichkeit, den variablen, sich ständig ändernden Anforderungen an Software gerecht zu werden, besteht darin, die Implementation einer Software zu öffnen. Software dieser Art spezifiziert neben dem eigentlichen Funktionsumfang die *internen* Strukturen und Abläufe. Die in konventionellen Entwürfen meist verborgene Implementation wird dabei in *festgelegter* Form dem Anwendungsprogrammierer zugänglich gemacht. Es kommt zu einer Auflösung der strikten Trennung zwischen Systementwicklern und Systembenutzern, da auch letztere durch die geöffnete Implementation Eingriffsmöglichkeiten in die Software bekommen.

Der Entwurf eines solchen Systems kann mittels einer *Metaebenen-Architektur* geschehen. Bei dieser Form von Systemarchitektur wird während der Entwicklung eines Systems neben der Basisebene, die den Funktionsumfang des Systems festlegt, eine Metaebene eingeführt, die der Beschreibung der Basisebene dient. Die Metaebene ist auch in konventionell entworfenen Systemen implizit vorhanden (z.B. wird eine Instanz einer Klasse eines C++ Programms durch eine interne Struktur etwa eines C-Programms repräsentiert), wird jedoch nicht expliziert und bleibt dem Programmierer verschlossen. In offenen Metaebenen-Architekturen wird die Metaebene dem Programmierer zugänglich gemacht. Wird die Metaebene durch Objekte im objektorientierten Sinn realisiert, werden diese auch *Metaobjekte* genannt. Eine vom Programmierer mittels einer objektorientierten Programmiersprache (OOP) definierten Klasse wird z.B. durch ein Klassenmetaobjekt repräsentiert, welches dem Programmierer zugänglich ist. Die Struktur und das Verhalten eines Systems wird durch *Protokolle* spezifiziert. Protokolle legen meist durch wenige Funktionen einen festumrahmten, funktionalen Teil eines Systems fest. Die Vereinigung von Protokollen, die das Verhalten von Metaobjekten beschreiben, wird *Metaobjektprotokoll* genannt.

Ein wesentlicher Vorteil eines so entworfenen Systems besteht darin, daß es für eine spezielle Anwendung anpaßbar ist. Metaobjektprotokolle bilden Sprachschnittstellen, die nicht nur die Möglichkeit liefern, Programme in der Sprache zu schreiben, sondern inkrementell das Sprachverhalten und die Sprachimplementation anzupassen.

Metaebenen-Architekturen werden bei der Entwicklung unterschiedlicher Systeme eingesetzt, wie z.B. für den portablen Benutzerschnittstellenmanager von Common Lisp CLIM (Common Lisp Interface Manager) [6]. Im Bereich von objektorientierten Sprachen sind Metaobjektprotokolle z.B. für TELOS [1] und Smalltalk [6] angegeben. Sie unterscheiden sich in der Granularität der Spezifikation. Eine bis ins Detail genau spezifizierte Realisierung eines Metaobjektprotokolls ist während der Entwicklung der Programmiersprache CLOS (Common Lisp Object System [3]) durchgeführt worden. CLOS spiegelt die Vorgehensweise, Ziele, Vorteile und Nachteile einer Sprachentwicklung mit einem Metaobjektprotokoll wieder.

Das Metaobjektprotokoll von CLOS - das MOP

Die Entwicklung von CLOS [2, 4] hatte die drei zunächst widersprüchlich erscheinenden Ziele: Kompatibilität zu bereits vorhandenen, objektorientierten Erweiterungen von LISP, Erweiterbarkeit der Sprachdefinition und Effizienz. Das gemeinsame Problem dieser drei Anforderungen liegt darin begründet, daß der Entwurf einer Programmiersprache jeweils spezifische Bedürfnisse verfehlt: Kompatibilität erfordert die Berücksichtigung von Besonderheiten vorhandener Software und Sprachen, Erweiterbarkeit ist durch eine spezifische Forderung der aktuellen Anwendersoftware begründet, und die effiziente Bearbeitung der unterschiedlichsten Benutzerprogramme erfordert spezifische Implementationsstrategien. Damit liegt die Schlußfolgerung nahe, daß *eine* Sprache *allein* nie die vorhandenen und kommenden Anforderungen erfüllen kann, egal wie elegant der Entwurf auch sein mag. Durch die Einführung des MOP liefern die Entwickler von CLOS daher nicht einen bestimmten Punkt im Sprachraum, sondern einen Bereich von möglichen Sprachen. Dadurch ist es möglich, CLOS einfach zu halten und gleichzeitig die drei oben genannten Ziele durch Wahl des aktuell passensten Punktes, d.h. einer gewünschten Sprachausprägung, zu erfüllen.

Wodurch wird ein solches Vorgehen ermöglicht? Wie erreicht man es, daß eine Sprache es erlaubt, sich selbst zu verändern? Der Sprachentwurf von CLOS basiert auf reflektiven und objektorientierten Techniken. *Reflektion* liefert die Möglichkeit, der Betrachtung (introspection) und Änderung (intercessory) des Programmverhaltens zur Laufzeit [5]. Dabei wird die Sprache geöffnet, ohne Implementationsdetails oder Portabilität preiszugeben. Portabilität heißt dabei, daß Änderungen einer Implementation, die für eine Anwendung auf der Metaebene gemacht werden, auch mit anderen Implementationen kompatibel sind.[1] Objektorientierte Implementierung von CLOS bedeutet, daß jedes Konstrukt von CLOS (d.h. Klasse, Methode und generische Funktion [2]) mittels Klassen dargestellt wird. Instanzen dieser Klassen repräsentieren aktuelle Konstrukte eines CLOS-Programms - die Metaobjekte. Diese Klassen heißen daher *Metaobjektklassen*.

---

[1] Im Falle des MOP liegt zwar keine Standardisierung vor, jedoch eine Akzeptanz der vorhandenen Spezifikation des MOP [4] bei den unterschiedlichen Herstellern.

[2] Generische Funktion in CLOS ist ein erweitertes Konzept der polymorphen Operation, welche in anderen Sprachen nur mit Methoden und Botschaftenaustausch ausgedrückt wird.

Das Verhalten von CLOS ist durch Protokolle für Metaobjektklassen spezifiziert. Ein solches Protokoll ist z.B. das Initialisierungsprotokoll, welches durchlaufen wird, nachdem eine Instanz erzeugt wurde. Ein anderes Beispiel ist das Instanzstrukturprotokoll, welches Funktionen festlegt, die beim Slotzugriff ausgeführt werden.[3] Durch Unterklassenbildung der Metaobjektklassen und Spezialisierung des vorgegebenen Protokolls ist eine gezielt einsetzbare Erweiterung oder Änderung des Sprachverhaltens von CLOS möglich.
Eine Anwendung des MOP liegt in der Bereitstellung von persistenten Objekten. Durch eine Metaobjektklasse und die damit verbundenen Spezialisierungen des Protokolls wird die Verwaltung von Instanzen und Slotwerten in objektorientierten Datenbanken geleistet. Kommerzielle Anwendungen dieser Erweiterung auf der Basis des MOP sind z.B. in den Systemen Itaska und AllegroStore realisiert. Eine weitere Anwendung bilden Erweiterungen von CLOS, die die Implementation eines wissensbasierten Systems vereinfachen. Ein Beispiel für solche Erweiterungen sind Slots, die in unterschiedlichen Kontexten verschiedene Werte haben können. Der Zugriff auf solche Slots wird durch eine Metaobjektklasse kontextabhängig verwaltet. Dies dient z.B. dazu, dem Benutzer verschiedene Sichten auf ein Konzept zu ermöglichen. Ein Überblick über weitere Anwendungen liefert [6].
Ein Problem bei der Verwendung des MOP besteht darin, die zu ändernde Stelle im Protokoll zu finden. Dazu muß die Implementation überschaut werden können. Durch die Lokalität, Kürze und Objektorientiertheit der einzelnen Protokolle sind Änderungen selbst jedoch nur kleine Eingriffe.

Protokolle und Metaebenen als Implementationsprinzip
Das wesentliche Prinzip bei der Verwendung von Protokollen liegt darin begründet, daß Struktur und Verhaltensweisen nicht in einen festen Algorithmus gepreßt werden. Stattdessen wird der Entwurf eines Systems bereits auf die zukünftigen Anforderungen und Änderungsmöglichkeiten ausgerichtet und die Einstiegspunkte in die Vorgehensweise in einem Protokoll festgehalten. Die Öffnung der Implementation einer Programmiersprache oder einer Anwendersoftware erscheint gewagt. Implementationsdetails sollten den Programmierer nicht von seiner eigentlichen Aufgabe, eine Anwendung zu realisieren, abhalten. Wie weit eine Implementation geöffnet werden sollte, ist jedoch eine schwierige Frage. Einerseits soll nicht zu viel verborgen bleiben, damit sinnvolle Erweiterungen möglich werden. Andererseits dürfen dem Implementator einer Sprache nicht zu restriktive Regeln auferlegt werden, um eine effiziente Implementation zu gewährleisten. Die neuralgischen Punkte eines Systems werden daher erst durch einen langwierigen Prozess gefunden, der den Entwurf einer Metaebenen-Architektur im Vergleich zu konventionellen Systemen verlängert. Ist dieser Prozess jedoch abgeschlossen, liegen lokale, präzise formulierte Punkte vor, die Ansätze für portable Änderungen geben.

Lothar Hotz, Universität Hamburg

[3] Slots bezeichnen Felder in Instanzen, sie werden in anderen Sprachen auch *Instanzvariable* genannt.

Literatur

[1] H. Bretthauer, H. Davis, J. Kopp und K. Playford. Balancing the EULISP Metaobject Protokol. *Lisp and Symbolic Computation, An International Journal, 6:1/2, 119-138, 1993.*

[2] R.P. Gabriel. Book Review. G. Kiczales, J. des Rivières, and D. G. Bobrow, The Art of the Metaobject Protocol. *Artificial Intelligence*, 61: 331-342, 1993.

[3] S.E. Keene. *Object-Oriented Programming in Common Lisp: A Programmers Guide to CLOS.* Addison-Wesley. 1989.

[4] G. Kiczales, J. des Rivières und D. G. Bobrow. *The Art of the Metaobject Protocol*. The MIT Press, Cambridge, MA, 1991.

[5] P. Maes und D. Nardi, Hrsg. *Meta-Level Architectures and Reflection.* North-Holland, 1988.

[6] A. Paepcke, Hrsg. *Object-Oriented Programming, The CLOS Perspective*. The MIT Press, Cambridge MA, 1993.

Published: Metaobjektprotokolle, KI-Lexikon. KI-Zeitung 8(4), 1994, 35-36