

Entwurfsmuster-Erkennung mit BAUHAUS

Eduard Wiebe, Steffen Keul, Stefan Staiger, Gunther Vogel,
Andreas Haufler und Wolfgang Scherer
Universität Stuttgart
bauhaus@informatik.uni-stuttgart.de

Abstract: In diesem Artikel stellen wir ein interaktives Werkzeug zur Erkennung von Entwurfsmustern in Java-Programmen vor. Unser Werkzeug ermöglicht die Spezifikation von zu suchenden Mustern als UML-Klassendiagramme, sucht nach Instanzen dieser Muster in vorliegendem Bytecode, gewichtet die Vorkommnisse anhand des Übereinstimmungsgrades und ermöglicht eine manuelle Überprüfung anhand des zugrunde liegenden Quelltextes.

1 Einleitung

Die Wartung von Softwaresystemen ist eine herausfordernde Tätigkeit. Durch Veränderungen an der Software wird gegen Entwurfsideen verstoßen; häufig werden dabei auch neue Ideen eingebracht, so dass die Wartbarkeit des Systems im Laufe seines Lebens abnimmt. Statische Analysen zur Architektur-Erkennung aus dem Quelltext helfen, einer Verschlechterung der Wartbarkeit entgegen zu wirken. Solche Techniken und Werkzeuge werden im BAUHAUS-Projekt¹ [RVP06] entwickelt. Um auch objekt-orientierte Programme besser zu unterstützen, wurde die in diesem Artikel beschriebene Entwurfsmuster-Erkennung zu BAUHAUS hinzugefügt.

Entwurfsmuster sind einfache und elegante Lösungen für wiederkehrende Entwurfsprobleme [GHJV95]. Die Information, dass die Instanzen bestimmter Klassen sich gemäß eines Entwurfsmusters verhalten, kann einem Wartungsingenieur wertvolle Hinweise zum Verstehen des Programms geben. Auch lassen sich Verstöße gegen den ursprünglichen Entwurf während der Wartung automatisiert erkennen.

Im Rahmen des BAUHAUS-Projekts wird das Werkzeug GRAVIS [CES02] zur Rekonstruktion der Architektur eines Softwaresystems eingesetzt. Diese Funktionalität von GRAVIS wurde mit der hier vorgestellten Entwurfsmuster-Erkennung ergänzt. Im Gegensatz zu existierenden Ansätzen handelt es sich dabei um ein interaktives Reengineering-Werkzeug, bei dem die Entwurfsmuster-Erkennung in eine umfassende Infrastruktur integriert ist. Dieser Artikel beschreibt nach einer kurzen Diskussion verwandter Arbeiten neben der Funktionsweise unseres Ansatzes auch einige praxisrelevante Aspekte der Interaktionsgestaltung.

¹<http://www.bauhaus-stuttgart.de>

2 Verwandte Forschung

Die Entwurfsmuster-Erkennung ist noch ein relativ junges Arbeitsgebiet. Als Vorläufer kann die *Clichés*-Erkennung [Wil92] gelten, die sich in erster Linie um die Erkennung der vorliegenden Abstraktionen im Allgemeinen, sei es auf Daten- oder Algorithmen-Ebene, bemüht. Die nach unserer Kenntnis am weitesten zurückreichende Arbeit, die sich mit Mustern nach [GHJV95] beschäftigt, stammt von Krämer und Prechelt [KP96]. Für die Programmiersprache Java berichten Seemann und von Gudenberg in [SvG98] von einem Verfahren zur Entwurfsmuster-Erkennung. Antoniol et al. [AFC98] entwickelten Werkzeuge, mit denen anhand von Metriken eine Filterung der Eingangsdaten durchgeführt wird. Die Arbeit von Tonella und Antoniol [TA99] geht einen anderen Weg, indem sie nicht im Quelltext nach Instanzen einer bestimmten Menge bekannter Muster sucht, sondern stattdessen aus dem Quelltext verwendete Strukturen extrahiert, die dann vom Benutzer auf ihre Relevanz weiter untersucht werden können. Heuzeroth et al. [HHHL03] zeigen, wie die Kombination einer statischen und dynamischen Analyse zur Erkennung von Verhaltens-Entwurfsmustern eingesetzt werden kann. Kaczor et al. [KGH06] formulieren die Suche als Pattern-Matching-Problem auf Bitvektoren.

3 Konzeption von EMMU

Obwohl also bereits vielfältige Ansätze erdacht wurden, ist uns dennoch kein *interaktives* Reengineering-Werkzeug für diese Aufgabe bekannt. Diese Lücke soll unser System „Erkennung und Visualisierung von Entwurfsmustern mittels UML“ (EMMU) ausfüllen. Auf der Grundlage unserer Anforderungen an die Entwurfsmuster-Erkennung entwickelte ein Studienprojekt das Werkzeug als Erweiterung von GRAVIS. Die Entwurfsmuster-Erkennung ist damit in eine umfassende Infrastruktur für das Software-Reengineering integriert.

In GRAVIS wird ein Resource Flow Graph (RFG) [Kos00, S. 58 ff.] des Quelltextes als Zwischendarstellung verwendet. Die Integration erlaubt es, die Ergebnisse der Entwurfsmuster-Erkennung weiterzuverwenden und mit anderen Analysen zu kombinieren, z. B. mit der in GRAVIS bereits implementierten Reflexionsmethode [KS03].

Die Eingabe der zu suchenden Muster (genannt *Schablonen*) erfolgt grafisch. Der Benutzer erstellt ein UML-Klassendiagramm der Schablone in EMMU. Er hat die Möglichkeit, in der Schablone Klassen und Interfaces mit Attributen und Methoden anzulegen. Diese können mit den in UML üblichen Kanten wie Vererbung, Assoziation und Aggregation, sowie mit weiteren für die Mustererkennung wichtigen Kanten wie der Creates- und Delegates-Kante oder auch jeder anderen Kante, die im RFG vorkommt, in Beziehung zueinander gesetzt werden (Abb. 1, „UML-Editor“).

An jedem Element der Schablone kann entweder annotiert werden, dass es in einem Erkennungsergebnis (*Kandidat*) zwingend enthalten sein muss, dass es erwünscht oder unerwünscht ist oder nicht enthalten sein darf. In Abbildung 1 wird in dem Dialogfenster „Edit Edge“ die Auswahl der Attribute für die Creates-Kante dargestellt. Mit einem Zahlenwert können erwünschte und unerwünschte Elemente gewichtet werden. Anhand dieser Ge-

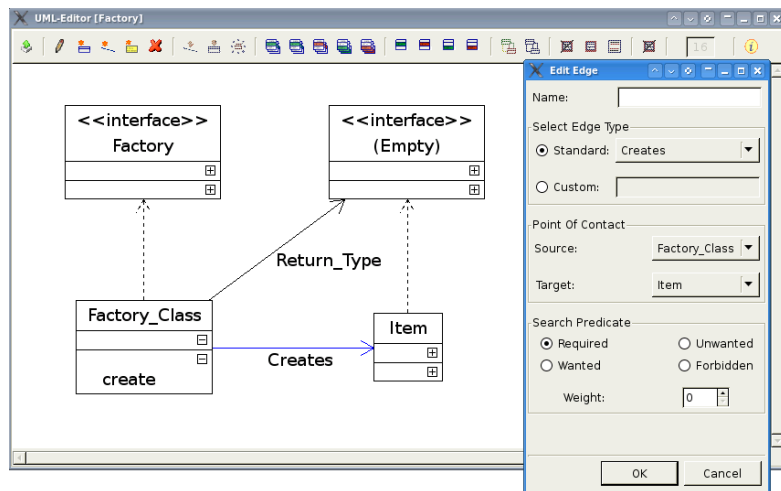


Abbildung 1: UML-Editor der Factory-Schablone

wichtung kann EMMU heuristisch die Übereinstimmungsqualität eines Kandidaten mit der Schablone abschätzen. EMMU unterstützt außerdem die Einschränkung von Feld- und Klassennamen über reguläre Ausdrücke. Der Suchraum kann durch Ausnutzung von Namenskonventionen oftmals stark eingeschränkt werden.

Nach Durchführung der Mustersuche zeigt EMMU eine nach Übereinstimmungsqualität sortierte Liste der Kandidaten. Diese werden als UML-Diagramm visualisiert, wobei das Layout der vom Benutzer erstellten Schablone beibehalten wird. Ist die Übereinstimmung nicht vollständig, so wird dies durch farbliche Hervorhebung der gefundenen und fehlenden Elemente verdeutlicht.

EMMU bietet dem Benutzer die Möglichkeit, die Ergebnisse der Mustersuche zu bewerten und gegebenenfalls zu präzisieren. Korrekt erkannte Kandidaten markiert der Benutzer in einer Checkbox in der Kandidatenliste. Es können aber auch weitere Elemente aus dem RFG zum Kandidaten hinzugefügt oder aus diesem entfernt werden. Dies kann notwendig sein, wenn durch die Analyse nur ein Teil der tatsächlichen Ausprägung des Musters erkannt wurde oder wenn zusätzliche Beziehungen zwischen den Klassen eines Kandidaten dokumentiert werden sollen.

4 Analyse

Die in EMMU eingesetzte Analyse beruht auf der Teilgraph-Suche von [SvG98] und besteht aus zwei Phasen: der Anreicherung des Graphen um zusätzliche Informationen und der eigentlichen Mustersuche. Die Implementierung trennt diese Phasen strikt.

Die Anreicherungsphase dient zur Herleitung weiterer Kanten und stellt sicher, dass der Graph die von den Schablonen erwartete Struktur aufweist. So werden zum Beispiel die

| Name | Initial | | Anreicherung | | Analyse | | |
|----------|---------|-------|--------------|-------|---------|-------|---|
| | SLoc | RFG | RFG | Zeit | RFG | Zeit | #Muster verifiziert/gefunden Art |
| ant | 104k | 4,4MB | 5,2MB | 5m13s | 5,4MB | 2m19s | -/28 A -/109 B 5/9 D 1/1 P 1/1 Si |
| antr | 50,5k | 2,0MB | 2,3MB | 1m34s | 2,6MB | 0m50s | -/15 A -/187 B -/12 D |
| catalina | 72,5k | 5,0MB | 5,6MB | 4m58s | 5,8MB | 6m04s | -/93 A -/60 B 3/6 D 1/2 Fd 0/5 P |
| cocoon | 232k | 6,0MB | 6,9MB | 5m25s | 7,7MB | 4m23s | -/74 A -/100 B 6/24 D 0/1 Fd 1/5 Fc 3/18 P |
| jasper | 27k | 2,0MB | 2,4MB | 1m11s | 2,6MB | 0m50s | -/29 A -/96 B 7/39 D |
| jedit | 91k | 5,2MB | 6,2MB | 7m15s | 6,5MB | 1m24s | -/75 A -/124 B -/62 D 0/2 P 1/2 Si |
| mina | 26k | 1,6MB | 1,8MB | 0m31s | 1,9MB | 0m32s | -/26 A -/9 B 0/2 D 3/6 Fc 0/2 Si |
| velocity | 31k | 1,9MB | 2,2MB | 0m42s | 2,2MB | 3m38s | -/17 A 3/3 D 2/3 P 1/3 Si |

Tabelle 1: Messergebnisse, teilweise durch Inspektion verifiziert.
 Legende: Adapter, Bridge, Decorator, Proxy, Abstract Factory, Facade, Singleton

transitiven Hüllen der Vererbungsrelation sowie die Propagierung der Aufruf-, Assoziations- und Delegationsbeziehung bis auf die Klassenebene berechnet. Die in Abbildung 1 gezeigte Creates-Kante wurde auf diese Weise propagiert.

Da die Effizienz der Propagierungsstrategie und Aufbereitung der Zusatzinformation stark von der zu untersuchenden Software abhängt, wurde die Anreicherung als Python-Skript realisiert, das über die Skript-Schnittstelle von GRAVIS aufgerufen wird. So können hier bei Bedarf projektspezifische Skripte angefertigt und ausgewählt werden.

Danach bestimmt die Mustersuche alle Teilgraphen, die die Vorgaben einer Schablone erfüllen. In Breitensuch-Reihenfolge werden alle Knoten des RFG besucht und zunächst überprüft, ob im RFG Entsprechungen für die in der Schablone mit „Required“ markierten inzidenten Kanten existieren, und ob die mit „Forbidden“ markierten Kanten keine Entsprechungen finden. Sind diese Bedingungen erfüllt, wird der Teilgraph als Kandidat erkannt und anhand der weiteren Attribute der Schablone gewichtet.

Die Suchmöglichkeiten von EMMU beschränken sich nicht auf Entwurfsmuster; die Teilgraphüberdeckung kann beispielsweise auch dazu genutzt werden, Anti-Pattern zu finden und somit vor Einführung unerwünschter Muster zu warnen.

5 Ergebnisse

Wir haben eine Bibliothek von Schablonen für bekannte Entwurfsmuster erstellt, unter anderem für Factory, Bridge, Adapter, Decorator, Facade, Proxy, Singleton und Strategy [GHJV95]. Diese Schablonen haben wir auf einige bekannte Open-Source-Projekte angewendet. Über die Größe der einzelnen Projekte (gemessen mit `sloccount`²) und die Anzahl der gefundenen Entwurfsmuster gibt die Tabelle 1 Auskunft. Man erkennt, dass EMMU auch auf größere Projekte anwendbar ist. Eine manuelle Auswertung einiger Ergebnisse lieferte dabei wie in der Tabelle gezeigt eine stark von der Software und dem Muster abhängige Rate an falsch-positiven Ergebnissen.

²<http://www.dwheeler.com/sloccount/>

6 Ausblick und Schluss

EMMU hat sich als ein praktisch einsetzbarer Beitrag auf dem Gebiet der Entwurfsmuster-Erkennung herausgestellt. Durch unsere Tests konnten wir noch weitere Ansatzpunkte für künftige Forschung mit diesem Werkzeug identifizieren. So erhoffen wir uns eine Verbesserung der Analyse durch die Ausnutzung von Kontroll- und Datenfluss-Informationen in der Anreicherung.

Bei manchen Kandidaten fällt es dem Benutzer schwer, die durch transitiven Schluss hergeleiteten Beziehungen im Quelltext nachzuvollziehen. Hier und speziell für die Integration in die in GRAVIS implementierte Reflexionsmethode [KS03] wollen wir die Benutzerführung für mehr Benutzungskomfort ausbauen. Ein Benutzer könnte z. B. in einer Soll-Architektur seine Vermutung über vorhandene Entwurfsmuster darlegen und dann regelmäßig die Soll- gegen die durch EMMU erkannte Ist-Architektur vergleichen.

Literatur

- [AFC98] G. Antoniol, R. Fiutem und L. Cristoforetti. Using Metrics to Identify Design Patterns in Object-Oriented Software. In *Int. Symp. on Software Metrics*, Seiten 23 – 34, 1998.
- [CES02] J. Czeranski, T. Eisenbarth und D. Simon. Softwarevisualisierungstool Gravis. In *4. Workshop Software-Reengineering*, Seiten 25–26, 2002.
- [GHJV95] E. Gamma, R. Helm, R. Johnson und J. Vlissides. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley, 1995.
- [HHHL03] D. Heuzeroth, T. Holl, G. Högrström und W. Löwe. Automatic Design Pattern Detection. In *11th International Workshop on Program Comprehension*, Seiten 94–103, 2003.
- [KGH06] O. Kaczor, Y.-G. Gueheneuc und S. Hamel. Efficient Identification of Design Patterns with Bit-vector Algorithm. In *10th European Conference on Software Maintenance and Reengineering*, Seiten 10–19, 2006.
- [Kos00] R. Koschke. *Atomic Architectural Component Recovery for Program Understanding and Evolution*. Dissertation, Universität Stuttgart, 2000.
- [KP96] C. Krämer und L. Prechelt. Design Recovery by Automated Search for Structural Design Patterns in Object-Oriented Software. In *WCRE*, Seiten 208–215, 1996.
- [KS03] R. Koschke und D. Simon. Hierarchical Reflexion Models. In *10th Working Conference on Reverse Engineering*, Seiten 36–45, 2003.
- [RVP06] A. Raza, G. Vogel und E. Ploedereder. Bauhaus – A Tool Suite for Program Analysis and Reverse Engineering. *Reliable Software Technologies, Ada Europe 2006 (LNCS 4006)*, Seiten 71 – 82, 2006.
- [SvG98] J. Seemann und J. W. von Gudenberg. Pattern-Based Design Recovery of Java Software. In *6th Intern. Symp. on Foundations of Software Engineering*, Seiten 10–16, 1998.
- [TA99] P. Tonella und G. Antoniol. Object Oriented Design Pattern Inference. In *International Conference on Software Maintenance*, Seiten 230–239, 1999.
- [Wil92] L. M. Wills. *Automated Program Recognition by Graph Parsing*. Dissertation, 1992.