Reengineering mit Sniffalyzer

Walter Bischofberger
Wind River Software Inc.
Brüttenweg 11
CH-8052 Zürich
wbischofberger@acm.org

Silvio Löffler
Software Systems Engineering Research Group
Technical University Cottbus, Germany
sl@informatik.tu-cottbus.de

Einleitung

Um bestehende Softwaresysteme erfolgreich umzubauen und zu erweitern, ist es wichtig, dass man die Mikro- und Makroarchitektur der Systeme versteht, dass man versteht wo die Qualitätsprobleme liegen und, dass man verhindern kann, dass beim Umbau die Architektur zerstört wird.

Die Dokumentation eines Softwaresystems ist nur sehr selten gut genug, um zu verhindern, dass man viele relevante Informationen direkt aus dem Quelltext ableiten muss. Programmierumgebungen liefern wertvolle Informationen, die das Verständnis großer Softwaresysteme auf der Mikroebene erleichtern. Diese Informationen sind aber vielfach auf einem zu tiefen Abstraktionsniveau, um beim Verstehen der Makroebene von entscheidendem Nutzen zu sein.

Informationen über die Makroebene können nur selten standardmäßig aggregiert und dargestellt werden. Erfahrene Architekten verwenden zum Verstehen größerer Zusammenhänge normalerweise verschiedene auf ihren Denkmustern und Erfahrungen basierende Prozesse. Je mehr dieser Prozesse wir verstehen und als Ganzes unterstützen können, desto besser unterstützen wir das Reengineering. Wir bezeichnen die abstrakte, wiederverwendbare Beschreibung solcher Prozesse als Code Comprehension Patterns. Unser Ziel ist es, parallel eine Plattform sowie generische und spezifisch Werkzeuge zu Entwickeln, die es massiv erleichtern, große Softwaresysteme zu verstehen. Diese Werkzeuge werden wir einsetzen, um in realen Anwendungsszenarien typische Code Comprehension Patterns zu entwickeln und zu explorieren. Auf diesen basierende Reengineering-Prozesse sollen dann durch spezialisierte Werkzeuge und Wizards unterstützt werden.

In diesem Kurzpapier geben wir zuerst einen kurzen Überblick über die Architektur unserer Werkzeuge, die Sniffalyzer-Plattform. Der Fokus liegt aber darauf zu beschreiben, wie man mit unseren Werkzeugen das Verstehen von großen Softwaresystemen und das Finden von Qualitätsproblemen in diesen unterstützen kann. Der Architekturüberblick liefert die Informationen, die es einem erlauben, die Anwendungsbeispiele in einen technischen Kontext zu stellen.

Sniffalyzer

Im Rahmen des Sniffgate-Projekts, einer Kooperation der Firma Wind River Systems (www.windriver.com) und dem Lehrstuhl Software-Systemtechnik der BTU Cottbus (http://www.software-systemtechnik.de), arbeiten wir seit fast zwei Jahren an der Sniffalyzer-Plattform.

Die Sniffalyzer-Plattform besteht aus einer relationalen Datenbank, die momentan über ein API aus der Programmierumgebung SNiFF+ gefüllt wird. Das könnte aber genauso gut über einen separaten Parser oder aus irgend einer anderen Quelle geschehen. Diese Datenbank enthält weitgehendvollständige Strukturinformationen für Java und C und weitgehend vollständige Informationen für C++[1]. Über den Code in Methoden und Funktionen wissen wir, was benutzt und aufgerufen wird, wir haben aber keinen abstrakten Syntaxbaum (d.h. wir haben typische Crossreferencing-Informationen).

Daneben haben wir eine Menge von Frameworks, die es uns erlauben, erste vollfunktionale Versionen neuer Werkzeuge in sehr kurzer Zeit (normalerweise ein bis zwei Wochen) zu implementieren. Konkret sind das Frameworks, die implementieren wie die Werkzeuge zusammenarbeiten, wie die gesamte Menü- und Shortcut-Behandlung funktioniert, wie man Daten generisch in Tabellen und Graphen visualisiert, und wie man aus den generischen Darstellungen wieder alle auf den spezifischen Typen verfügbaren Befehle absetzen kann.

Basierend auf dieser Plattform gibt es unterschiedlich spezifische Werkzeuge. Das allgemeinste Werkzeug ist der QueryDeveloper, ein generisches Abfragewerkzeug mit der Möglichkeit, eine Menge von SQL-Abfragen zu

---

[1] Im Rest dieses Papiers verwenden wir immer die Java spezifische Terminologie. Alle beschriebenen Analysen können aber genauso für C++ durchgeführt werden.

erstellen, verwalten und auszuführen. Dieses Werkzeug kann mit minimal erweitertem SQL umgehen, um Abfragen auf bestimmten Entitäten ausführen zu können, ohne die SQL-Abfrage ändern zu müssen. Häufig entstehen beim Arbeiten mit dem QueryDeveloper Abfragefamilien. Das sind Abfragen die grundsätzlich den selben Zweck haben aber sich in Details unterscheiden. Für das Beschreiben solcher Abfragefamilien haben wir eine eigene Sprache entwickelt. Diese Sprache wird vom QuerySniffer ausgeführt, der dynamisch ein GUI zum Einstellen von Abfrageparametern erzeugt und aufgrund der eingestellten Werte eine konkrete SQL-Abfrage generiert. Daneben gibt es eine Reihe konkreter Werkzeuge, die weiter unten im Rahmen ihrer Anwendung kurz umrissen werden. Die mittelfristige Vision ist es, ganze Analyseprozesse, die mit verschiedenen Werkzeugen durchgeführt werden, mit Code Comprehension Patterns basierten Wizards zu unterstützen.

Zur Unterstützung der Mikroanalyse auf Quelltextebene gibt es eine generische IDE-Anbindung, die momentan für SNiFF+ implementiert ist und die es ermöglicht, jederzeit für jede Entität direkt in die Programmierumgebung springen.

Wir setzten Sniffalyzer bereits bei der eigenen Entwicklung wie auch zum Studium anderer grosser Java-Anwendungen ein. Es fehlen aber noch einige Bausteine bis die gesamte hier skizzierte Funktionalität zur Verfügung steht. Die wichtigsten fehlenden Bausteine sind das Wizard-Framework und die Wizards, SniffalyzerAccess, d.h. der externe Zugriff auf die Sniffalyzer-Funktionalität, das Menu-Framework und das Graph-Framework.

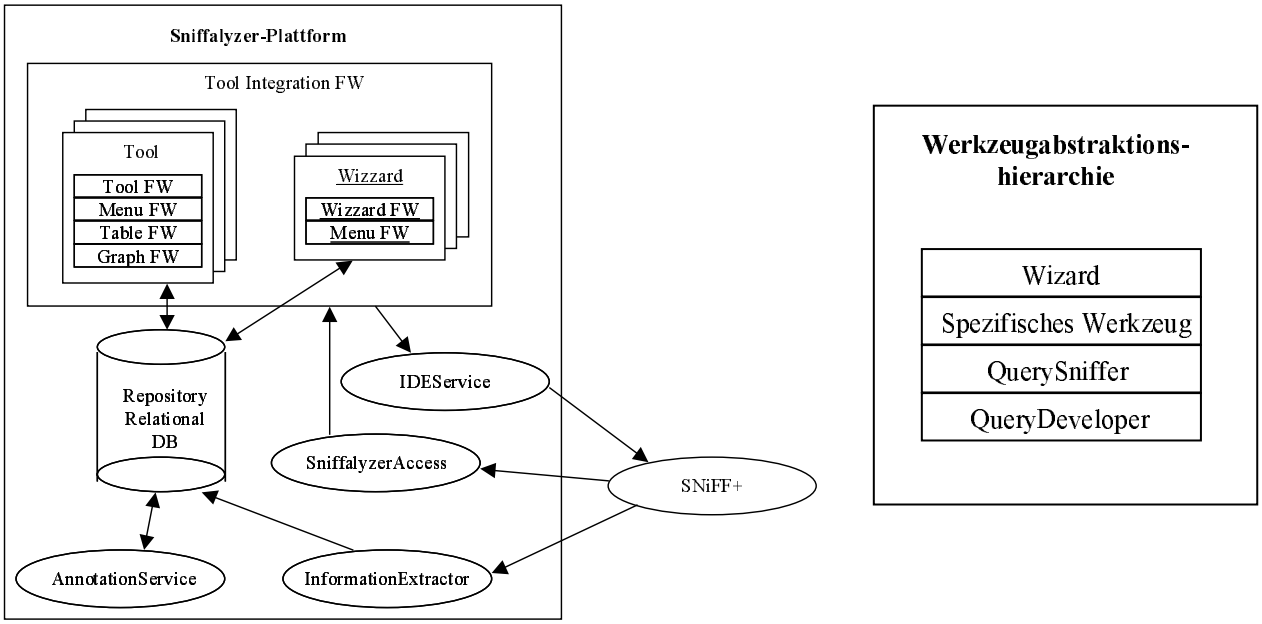

Abbildung 1: Architekturübersicht und Werkzeugabstraktionshierarchie

Verstehen von objektorientierten Systemen mit Sniffalyzer

Top-Down-Analyse

Typische Code Comprehension Patterns für die Top-Down-Analyse sind das Suchen und Verstehen von zentralen Frameworks, das Verstehen, wie man ein Framework erweitert und das Suchen und Verstehen der Benutzungsschnittstelle, der am häufigsten verwendeten Packages eines Systems.

Eine mögliche Instanziierung für das Pattern zum Suchen und Verstehen von zentralen Frameworks mit Sniffalyzer läuft in den folgenden Schritten ab:

- Suchen der Packages, von denen am meisten geerbt wird (d.h. die Klassen enthalten unter denen tiefe und/oder breite Vererbungsbäume hängen.)
- Anschauen ihrer Position in der Packagehierarchie und der Position der Klassen in der Vererbungshierarchie. Dabei kann man aufgrund der Position und der Namen von nahegelegenen Knoten häufig schon erste Rückschlüsse ziehen.

- Analyse der vererbungsrelevanten Package-Schnittstelle. Z.B. welche Methoden werden von Unterklassen typischer Klienten-Packages überschrieben oder von diesen aufgerufen.
- Suche von Gruppen von eng kooperierenden Klassen. Klassengruppen die sich gegenseitig verwenden oder über Instanzvariablen kennen. Damit findet man häufig den Kern von Frameworks. Studium dieser Gruppen z.B. im Kontext der Vererbung oder aufgrund der gegenseitigen Verwendungsschnittstellen.
- Studium der Klassenfamilien der zentralen Klassen des mutmaßlichen Frameworks aufgrund der Vererbungshierarchie und der Übersicht, welche Methoden, welcher Ebene wo überschrieben und benutzt werden. Aus diesen Informationen sind die zentralen Abstraktionen, die eine Klassenfamilie implementiert, meist direkt ableitbar.

Bottom-Up-Analyse

Bei der Bottom-Up-Analyse geht es darum, eine bestimmte Entität zu verstehen. Neben verschiedenen strukturellen Abfragen verwenden wir dazu häufig den XrefSniffer. Dieser bietet Crossreferencing auf beliebig mischbaren Abstraktionsebenen an. Man kann für beliebige Mengen von Beziehungsarten (Call, Read, Write, Containment, Inheritance, TypeAccess) die Beziehungen von einer Entität auf einer beliebigen Abstraktionseben zu allen in Beziehung stehenden Entitäten einer beliebigen anderen Abstraktionseben analysieren. Abstraktionsebenen sind Package, File, Class, Symbol. Zwischen den Abstraktionsebenen kann flexibel navigiert werden, und aggregierte Resultate (z.B. Beziehungen von einem Package) können jederzeit aufgelöst und im Quelltext angeschaut werden.

Für detailliertere Analysen konkreter Beziehungen springt man in die integrierte IDE.

Qualitätskontrolle von Objektorientierten Systemen

Sicherstellen der Einhaltung architektureller Restriktionen

Der ArchitectureSniffer erlaubt es, die Architektur eines Systems durch eine Anzahl Schichtenmodelle zu beschreiben und dann basierend auf diesen Schichtenmodellen zu prüfen, ob es nicht erlaubte Beziehungen im System gibt. Wir hoffen dadurch den typischen Zerfall von Architekturen während der Wartung eindämmen zu können, da man frühzeitig erkennen kann, wenn Wartungsingenieure unerwünschte neue Abhängigkeiten in die Codebasis einschleppen.

Qualitätskontrolle mit Softwaremetriken

Mit dem MetricsSniffer gibt es auch ein "konventionelles" Softwaremetrikwerkzeug auf der Sniffalyzer-Plattform. Bei der Entwicklung des MetricsSniffer haben wir versucht, die typischen Schwächen der auf dem Markt erhältlichen Metrikwerkzeuge auszumerzen. Konkret haben wir viel Zeit in die Browsing-Unterstützung der Messwerte investiert. Werte die nicht von Interesse sind, können auf verschiedenen Abstraktionsebenen ausgefiltert werden. Werte von Interesse können annotiert und aufgrund der Annotationstypen und deren Werten später wieder gezielt gefunden werden. Dadurch wird es einfach, Mengen von Werten zu verwalten, die z.B. noch genauer analysiert werden sollten oder aufgrund von denen Änderungen im Quelltext vorgenommen werden sollen.

Auffallende Werte können jederzeit mit den anderen Analysewerkzeugen oder in der eingebundenen Programmierumgebung studiert werden.

Qualitätskontrolle mit Analyseabfragen

Bei der Unterstützung dieses Gebiets stehen wir noch ganz am Anfang. Bis jetzt bietet der QuerySniffer verschiedene Abfragen zum Auffinden von nicht benutzten Entitäten an (Packages, Classes, Methods, Fields), die nie gebraucht werden.

Trendanalyse

Qualitätsanalysen einer bestimmten Version eines Systems sind nützlich. Während der Entwicklung ist es aber von größerem Interesse, zu wissen wie sich die Qualität über die Zeit verändert. Welche schlechten Meßwerte verschlechtern sich noch weiter? Welche illegalen Abhängigkeiten und welcher unbenutzte Code sind neu eingeführt worden? Die aktuelle Version von Sniffalyzer unterstützt diese Art von Analysen nicht. Wir planen aber sie in Zukunft zu implementieren.