ATOSj:
Ein Werkzeug für den automatisierten Regressionstest oberflächenbasierter Java-Programme

Volker Janetschek & Nicos Tegos

Regressionstest

- Software ist häufigen Veränderungen unterworfen
- Überprüfung der Programmfunktionalität nach jeder Änderung
- Verringerung des Testaufwands durch Automatisierung → ATOS

Motivation für eine Java-Erweiterung

- Schnellere Hardware macht GUI-basierte Java-Programme konkurrenzfähiger
- Standardbibliothek Swing emuliert graphische Elemente
- SWT aus dem Eclipse-Projekt verwendet Components des Betriebssystems, ähnlich dem AWT
- SWT ist schnell und fördert die Nutzerakzeptanz auf Grund des vertrauten Aussehens

SWT

Swing

Windows

Mac

ATOSj - Ein neues System

- Vorgänger ATOS ist Windows-basiert
- Abbildung der unabhängigen Skriptsprache HTS auf das Windows-spezifische ATS
- Windows-Programm nicht geeignet plattformunabhängige Java-Programme zu testen

ATOSj - Ein neues System

- Integration einer Java-Unterstützung in ATOS würde ca. 81% der Implementation berühren

Komponente	*LOC*	*Anteil in %*
GUI-Komponenten	11000	36
Klasse ATOS	7500	25
Klasse ATSParser	6000	20
Klasse HTSParser	3900	13
Klasse CTEParser	1700	6
Summe	**30100**	**100**

- Komplizierte Kommunikation mit Java-Testobjekten über JNI
- Neues System komplett in Java → ATOSj

Vorlagen

- Ausgangspunkt: Erweiterung von ATOS um Java - Funktialiät
- Zuvor: Suche nach ähnlichen Implementationen
  - Einige kommerzielle Produkte:
    - Meist kleinere unbekannte Projekte
    - Sehr teuer
    - Detaillierte Informationen kaum beschaffbar
  - Marathon
- ATOSj: Marathon in ATOS?

Marathon

- Open Source Projekt
- Ebenfalls Capture & Replay
- GUI ähnlich der von ATOS
- Scriptsprache: Phython
- Einschränkung: Nur Swing
- Zum Zeitpunkt der Analyse komplett, leicht fehlerbehaftet

Entwicklungsmodelle

- Neues System, dennoch
  - Stärken beider Programme vereinen
  - Bekannte Schwächen durch eigene Ideen ersetzen
  - Erweiterung um zusätzliche, neue Funktionen

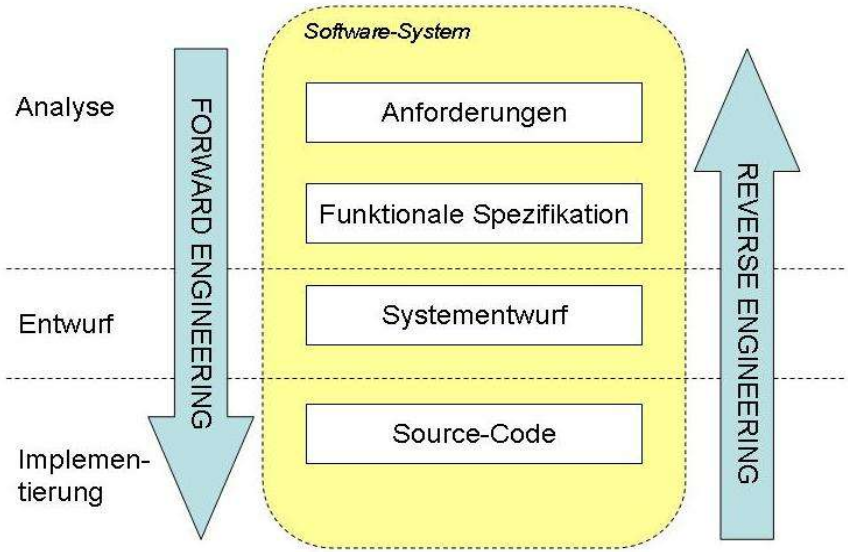
→ Ergänzendes Entwicklungsmodell: Reverse Engineering

  - Identifikation von Systemkomponenten
  - „Von der Implementation zum Entwurf“

Reverse Engineering bei ATOSj

- Systematische Analyse beider Vorlagen
- Vergleich & Evaluierung der jeweiligen Teillösungen

- Anschliessend
  - Idee umsetzten oder
  - Eigenen Ansatz entwickeln



Entwicklungsparadigma

- Weiteres Problem zu Beginn:

  kaum Erfahrungen auf dem betreffenden Gebiet

  - → Aufwandsabschätzung schwer möglich
  - → Wünschenswerter Funktionsumfang des Zielsystems schwer bestimmbar

- → Prototyping
  - Abschnittweise Entwicklung
  - Abschluss eines Abschnitts durch lauffähige Version
  - Darauf basierend Evaluation und neue Zielsetzung

Entwicklungsvorgang

- Ingesamt 6 Prototypen entstanden
- 1. und 2. Prototyp fast ausschliesslich bestehend aus mittels Rev. Eng. gewonnenen Ideen von ATOS und Marathon
- Durch 3. und 4. Prototyp fast sämtliche Komponenten durch Implementationen eigener, verbesserter Ideen ersetzt
- In 5. und 6. Prototyp neue Komponenten hinzugefügt
  - ATOSj hat deutlich mehr Features als Marathon und ATOS zusammen

Was bleibt von den Vorlagen?

- Von ATOS übernommen:
  - Basisaufbau GUI
  - Grundidee der Test-Skripte und -Pakete
  - Skriptsprache HTS
- Von Marathon übernommen:
  - Basisidee für Kommunikation mit den Testobjekten
  - Grundprinzip der Hierarchie der Klassen
- Vorlagen ohne Vorwissen kaum noch erkennbar

Resümee zur Entwicklung

- Zwischenzeitlich einfache Kopie von Ideen
- In Endversion sämtliche verwendete Ideen durch eigene Ansätze ersetzt
- Reverse Engineering war dennoch sehr wertvoll
  - Saubere Struktur der Implementation, da Referenzlösungen bereits in A&D – Phase vorhanden
  - Anfängliches Kopieren von Fremdideen diente Einstieg und besserem Verständinis der Thematik
- Erhöhter Zeitaufwand durch Prototyping (?)

Neue Features (1)

- Unterstützung des SWT
- Automatische Generierung von TEST-Kommandos während des Aufzeichnens
- Plugin-Mechanismus zur Ertstellung eigener Custom-Components

Neue Features (2)

- automatische Überdeckungsanalyse während des Ablaufs der Testsequenzen
- PDF-Report-Generierung
- Erhöhter Nutzerkomfort der GUI (z.B. Copy & Paste von Befehlen)

Wahl der Skriptsprache

- Skriptsprache für Strukturierte Speicherung von Testfällen
  - Einfachheit
  - Ausrichtung auf das Gebiet Oberflächentest
- **XML**
  - \+ Strukturierte Datenspeicherung
  - \+ Freie Semantikdefinition
  - \- Schlechte Lesbarkeit/Bearbeitbarkeit durch Strukturinformationen

Wahl der Skriptsprache

- **Python**
  - \+ Mächtige Skriptsprache (Kontrollkonstrukte, mathematische Operationen, funktionales und objektorientiertes Paradigma)
  - \+ Interpreter vorhanden
  - \- Komplex → schwer erlernbar
  - \- Keinerlei Restriktionen → reduziert allgemeine Verständlichkeit

Wahl der Skriptsprache

- **HTS**
  - \+ Bewährt im Einsatz (XCTL-Projekt)
  - \+ Wenige übersichtliche Kommandos
  - \+ Eingrenzung auf den Bereich Oberflächentest
  - \- Nicht durch den Tester erweiterbar

Überarbeitung von HTS

- Erweiterung um Standard-Typen – Tabelle, Baum, Karteikarte
- **ACTION**
  - Parameterumstellung:
    Alt: `ACTION, "Fenster", EDITBOX, EDIT, "Dr.", "Titel"`
    Neu: `ACTION, "Fenster", EDITBOX, "Titel", EDIT, "Dr."`
  - Einheitliches Konzept für den Zugriff auf zusammengesetzte Components:
    `ACTION, "Fenster", TABLE, "kunden", SELECT, SUBITEM, 0`
  - Neue Aktionsarten `PRESSKEY` und `CLICK`

Überarbeitung von HTS

- **READ**
  - Auslesen der Elementanzahl:
    ```
    READ, "Kunde", LIST, "lst", ITEMCOUNT, "numCustomers"
    ```
- **TEST/Compare**
  - Beseitigung von Inkonsistenzen durch Explizierung des booleschen Vergleichs:

    Alt:
    ```
    READ, MAIN, EDITBOX, ENABLESTATE, var, "Name"
    TEST, MAIN, EDITBOX, DISABLED, "Name"
    COMPARE, var, NUM, VAL, 0
    ```

    Neu:
    ```
    READ, MAIN, EDITBOX, "Name", ENABLESTATE, "var"
    TEST, MAIN, EDITBOX, "Name", ENABLESTATE, FALSE
    COMPARE, "var", BOOL, VAL, FALSE
    ```

Überarbeitung von HTS

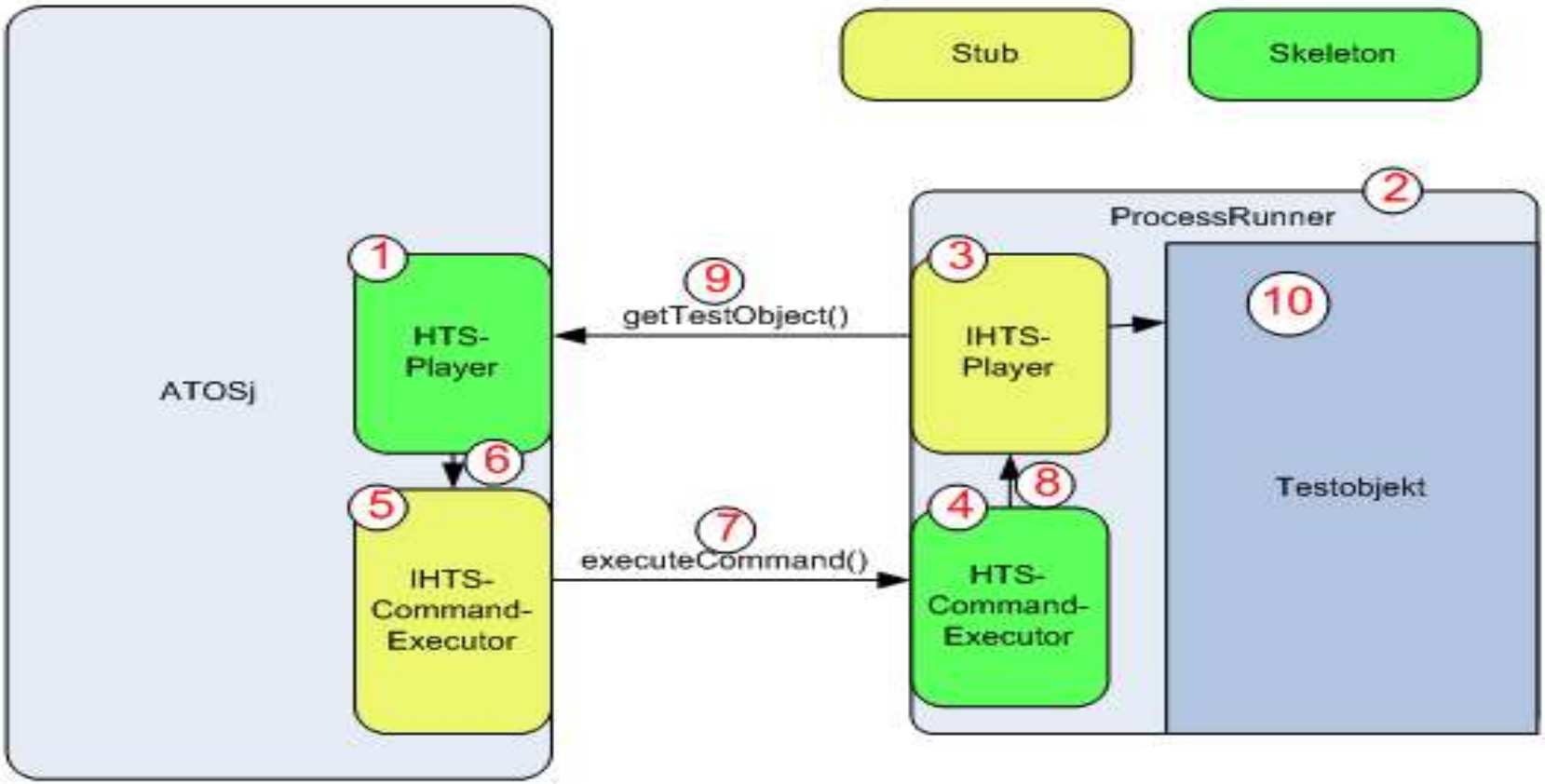
- **WHILE**
  - Unbedingte Schleife, dient zur Vermeidung von Synchronisationsproblemen
  - Synchronisationsprobleme entstehen, wenn mit dem Test nicht fortgefahren werden kann bevor nicht ein bestimmter Zustand erreicht wurde

```
READ, "Datei verschicken", BUTTON, "Beenden", ENABLESTATE, "senden beendet"
WHILE, "senden beendet", BOOL, VAL, FALSE
READ, "Datei verschicken", BUTTON, "Beenden", ENABLESTATE, "senden beendet"
ENDWHILE
```

Kommunikation mit dem Testobjekt

- ATOS -Lösung mittels Windows-Nachrichten für Java ungeeignet
- Plattforumunabhänigigkeit nur sichergestellt, wenn reine Java – Löung (keine JNI-Erweiterungen)
- ATOSj startet Testobjekt in neuem Prozess
- nutzt RMI zur Interprozesskommunikation
- Vor Start der Testsequenz paarweises Aufbauen von Kommunikationsinterfaces

Kommunikation mit dem Testobjekt



Identifikation Grafischer Elemente

- ATOS: Analyse der Resource-Dateien
- Für ATOSj nicht vorhanden
  - → Daher: Elementfindung zur Laufzeit
- Capture: Namenszuordnung bei Nutzaktion
- Replay: Elementfindung anhand des generierten Namens
  - → Beide Vorgänge müssen korrespondieren

Aufzeichnung von Nutzeraktionen Capture

- Schnelle Erzeugung von Testskripten, durchschnittliche Zeitersparnis 57,6%
- Syntaktische und logische Korrektheit
- Realisierung mit Javassist und Jaccess
- Erstellen einer globalen Überwachungsinstanz, die alle auftretenden Ereignisse weiterleitet
- Für jedes Component eine eigenen Klasse zur Erstellung von Kommandos, Implementation von Ereignisempfängern

Aufzeichnung von Nutzeraktionen Capture

- 2 Arten von Kommandos können aufgezeichnet werden: ACTION und TEST
- Für die Generierung von TEST-Kommandos kann aus einer Liste aller gültigen Zustandsprüfungen ausgewählt werden
- Alle generierten Kommandos sind kontextsensitiv, ein analoger Modus wird nicht unterstützt

Grundprinzip Capture

- Elementfindung anhand von Java-Events
- Betroffenes grafisches Element ist bekannt
- Generierung einer ID mittels Benamungsstrategie
- Erstellung des HTS-Kommandos anhand der Charakteriska des aufgetretenen Events

→ Benötigt wird globaler Listener-Mechanismus

Grundprinzip Replay

- Ermittlung des Zielelements
- Rekursives Durchsuchen aller existierenden Fenster
- Generierung einer ID für das aktuelle Element
- Abbruch falls gernierte ID gleich der ID des gesuchten Elementes
- → Gleicher Vorgang wie beim Capture
- → Benötigt Informationen über alle existierenden Fenster

ID Generierung

- 3 Bestandteile einer Component-ID:

```
"Hauptfenster",BUTTON,"OKButton"
```

1) Titel des umgebenden Fenster, wie in der Titelleiste angezeigt
2) Typ des Components, zB `BUTTON, TABLE, TREE, EDITBOX` oder Typ eines Custom Components
3) mittels Benamungsstrategie generierter Name des Components

Benamungsstrategie

- vierstufiges Modell:

1) Vom Programmierer festgelegt

- Swing: `Component.setName("Name")`
- SWT: `Widget.setData("ATOSJ_COMPONENT_NAME_KEY", "Name")`

2) Generiert in Abhängigkeit von Eigenschaften des Elements

- Beziehung zu einem Label
- Text auf einem Button

Benamungsstrategie

3) Position und Größe des Elements in seinem Fenster

- zB: [x=4,y=22,b=100,h=38]

4) Ergebnis der `Object.toString()`-Funktion des Elements

Entwickler muss sicherstellen, dass der Name pro Fenster und Elementtyp eindeutig ist.

Benötigte Javaerweiterungen

- Globale Informationen über Oberflächenelemente benötigt
- AWT/Swing: Java Accessibility Utilities
- SWT: ???
  - Eigenimplementation nötig
  - Idee: Elemente melden sich bei Erstellung bei Überwachungsmonitor
  - ATOSj erfragt alle Informationen bei diesem Monitor
  - ➔ Erfordert Veränderung der SWT-Quellen

SWTEventMonitor

- 2 Möglichkeiten:

1) Statische Anpasung der SWT-Quellen
   - unflexibel

2) Dynamische Veränderung beim Laden der Klassen
   - Hilfsmittel: Javassist

Architektur - Componentklassen

- 2-Schichten-Architektur sichert den Zugriff auf Components des Testobjekts
- Interface für jeden HTS-Typ
  - Methoden zur Simulation von Nutzeraktionen dem Kommando ACTION entlehnt
  - Methoden zum Auslesen von Zustandswerten dem Kommando READ entlehnt
- Identifikation eines Components für jeden HTS-Typ
- Bibliotheksspezifische Implementation der Typ-Interfaces

Plug-In Mechanismus

- Viele GUIs verwenden nutzerdefinierte graphische Elemente (Custom-Components)
- Zweck Annäherung an die Realwelt durch Verwendung von Methaphern z.B. Desktop
- Arten von Custom-Components sind mannigfaltig, es wird ein universeller Mechanismus zur Unterstützung benötigt

Plug-In Mechanismus

- Erweiterung des Testsystems durch den Tester
  - Definition von Eigenschaften (property)
  - Definition von Formaten für Eigenschaftswerte als String
  - Setzen von Eigenschaftswerten = Simulation von Nutzeraktion
  - Auslesen von Eigenschaftwerten = Ermitteln von Zuständen
  - Definition einer Wrapperklasse für das Custom-Component

Plug-In Mechanismus

- Erweiterung der Kommandos ACTION, READ, TEST für die Arbeit mit Custom-Components:

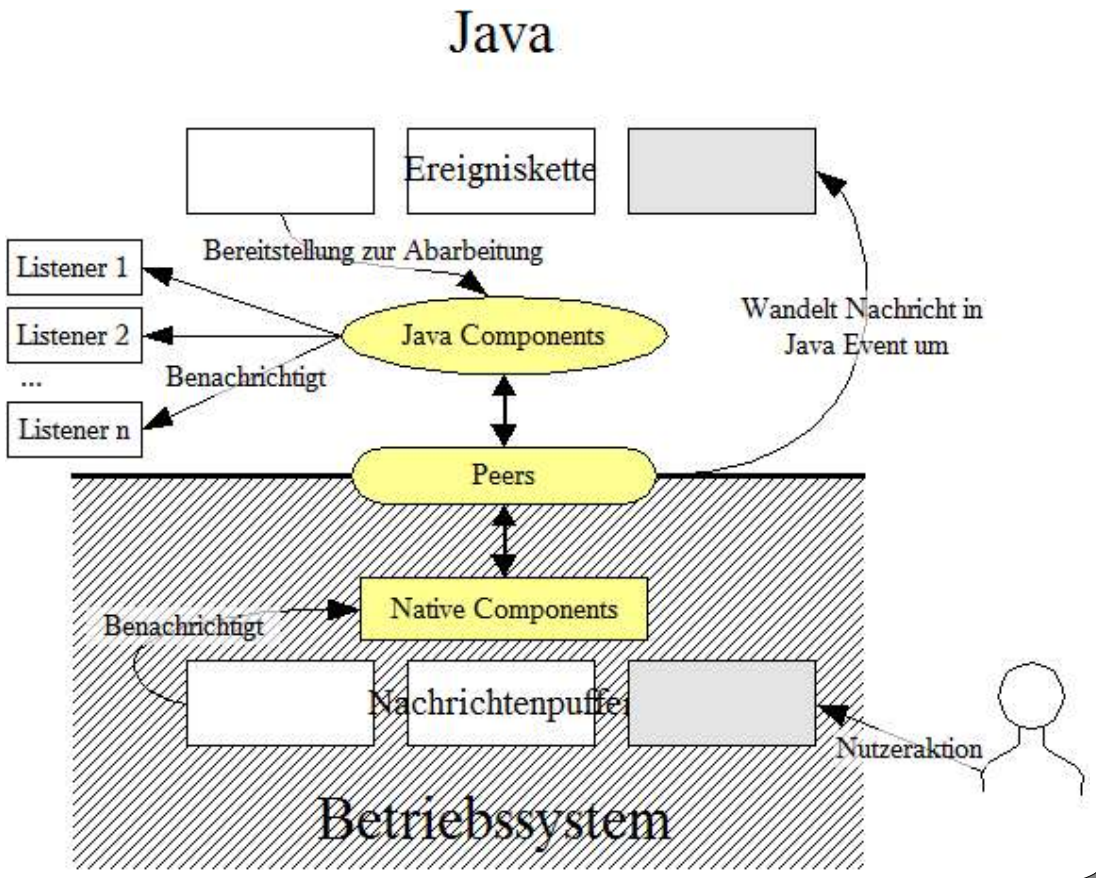
```
ACTION, "Kunde*", CALENDAR, "Datum", PROPERTY,
  "date", "24.06.1980"
```

Simulation von Nutzeraktionen

- Kommunikation des Betriebssystems mit Anwendungen über Nachrichten
- In Java informieren Ereignisquellen (Event Sources) Ereignisempfänger (Event Listener) über das Auftreten von Ereignisssen (Events) → Delegation Event Model
- Zwei Kategorien von Ereignissen
  - Elementarereignisse: Mausoperation, Tastatureingabe
  - Semantische Ereignisse: Treten in Folge von Elementarereignissen auf, sind Component-spezifisch z.B. die Änderung der Auswahl in einer Tabelle

Simulation von Nutzeraktionen

- In AWT werden Ereignisse in einer speziellen Ereigniskette (Event Queue) gesammelt



Simulation von Nutzeraktionen

- In SWT fehlt eine Ereigniskette auf Java-Seite, es wird direkt der Nachrichtenpuffer des Betriebssystems genutzt
- 2 Möglichkeiten zur Simulation von Nutzeraktionen
  - Der programmatische Ansatz
  - Der ereignisbasierte Ansatz

Simulation von Nutzeraktionen – Der programmatische Ansatz

- Direkte Manipulation von Components durch Aufruf von Methoden der jeweiligen Schnittstelle
- Beispiel:
  - Editieren eines Textfeldes zur Eingabe von Stückzahlen
  - Ein Ereignisempfänger verhindert die Eingabe aller Zeichen die keine Ziffern sind
  - `ACTION, "Bestellung", EDITBOX, "stueck", EDIT, "1000a"`
    `TEST, "Bestellung", EDITBOX, "stueck", TEXT, "1000"`
  - Nutzung der Funktion `setText(String)`
  - Problem: Ereignisempfänger werden nicht benachrichtigt

Simulation von Nutzeraktionen – Der ereignisbasierte Ansatz

- Dekomposition einer komplexen Nutzeraktion in Elementarereignissse
  - Falls das Textfeld nicht den Eingabefokus hat, Anklicken des Textfeldes, damit es den Fokus erhält.
  - Vollständige Auswahl des bereits enthaltenen Textes, um diesen zu überschreiben.
  - Eingabe der einzelnen Zeichen des Textes „1001a“.
- **Swing**
  - Einfügen generierte Elementarereignisse in die Ereignisskette

Simulation von Nutzeraktionen – Der ereignisbasierte Ansatz

- **SWT**
  - Umwandeln vo
    Betriebssystems
  - Zusätzlich prog
    betriebssystemabh
    Textauswahl in manc
  - Problem: sprachunab gige Simulation der Eingabe von Zeichen, SWT-Bug

Problem: The Display.post(Event) method is very handy to simulated user input. Everything is fine for single-key events. However, I would like to simulate typing, which is hard because for every character I want to input, I have to compute the corresponding key sequence, which depends on the keyboard layout being used.
Antwort: This limitation that I'm not sure we can work around portably. Anyhow, low priority for now.
https://bugs.eclipse.org/bugs/show_bug.cgi?id=71488

Datenbasis über die Components des Testobjekts

- Sammlung aller eindeutigen Bezeichner zu den Components des Testobjekts (Component-ID)
  - Name
  - Typ
  - Fenstertitel
- Dient der Überprüfung der referentiellen Integrität
- Dient als Unterstützung bei der manuellen Testkripterstellung

Datenbasis über die Components des Testobjekts

- Im Unterschied zu C-Programmen haben Java-Programme kein Ressource-File für Dialogelemente
- Automatische Generierung der Datenbasis durch Ausführen des Testobjekts
- Manuelle Bearbeitungsmöglichkeiten
  - Erstellen/Löschen/Ändern von Component-IDs
- Refactoring-Mechanismus übernimmt geänderte IDs in allen Testsequenzen

Überdeckungsanalyse

- Messung der Güte der spezifizierten Testfälle
- Verschiedene Überdeckungsmaße aus dem White-Box-Test z.B. Anweisungsüberdeckung
- Oberflächentest ist jedoch ein Black-Box-Test und sehr grobgranular
- Test an Hand funktionaler Anforderungen
- Bestimmung der Funktionsüberdeckung durch Approximation

Überdeckungsanalyse

- 3 neudefinierte Maße:
  - Methodenüberdeckung

$$C_{-1} = \frac{Anzahl\ durchlaufene\ Funktionen\ pro\ Klasse}{Anzahl\ Funktionen\ pro\ Klasse\ insgesamt} \cdot 100$$

  - Paketüberdeckung

$$C_{-2} = \frac{Anzahl\ überdeckte\ Klassen}{Anzahl\ Klassen\ insgesamt} \cdot 100$$

  - Systemüberdeckung

$$C_{-3} = \frac{Anzahl\ überdeckte\ Pakete}{Anzahl\ Pakete\ insgesamt} \cdot 100$$

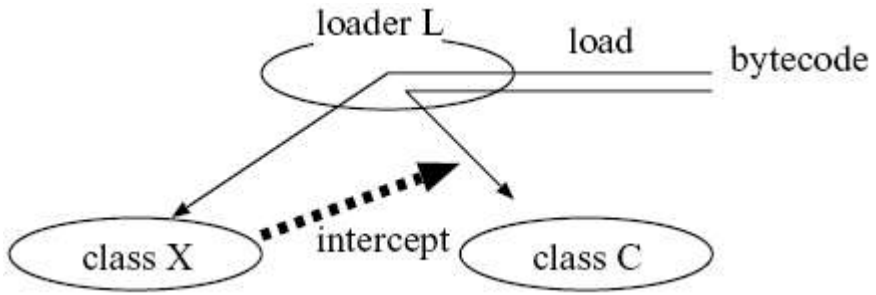
Überdeckungsanalyse

- Neues Werkzeug für die Berechnung der Überdeckungsmaße C_{-1}, C_{-2} und C_{-3}
- XML-Datenbasis zur Speicherung von Informationen zu den Methoden der Zielanwendung
- Laufzeitinstrumentierung der Zielanwendung, Einfügen eines Aufrufzählers in jede Methode
- Aktualisierung der Datenbasis nach jedem Testlauf und Visualisierung in der Ergebnisse in HTML

Javassist und AOP

- Änderung von Klassendefinitionen zur Lauf- oder Compilezeit
- Änderung des Bytecodes auf hohem Abstraktionsniveau
- Erweiterung der Introspektion – Hinzufügen neuer Methoden oder Felder zu Klassendefinitionen uvm.

```
CtClass clazz = ClassPool.getDefault().get("java.util.Vector ") ;
CtMethod.make(
" public String getStr(int index)" +
"{" +
" return (String) get(index); " +
"}" ,
clazz);
```



Javassist und AOP

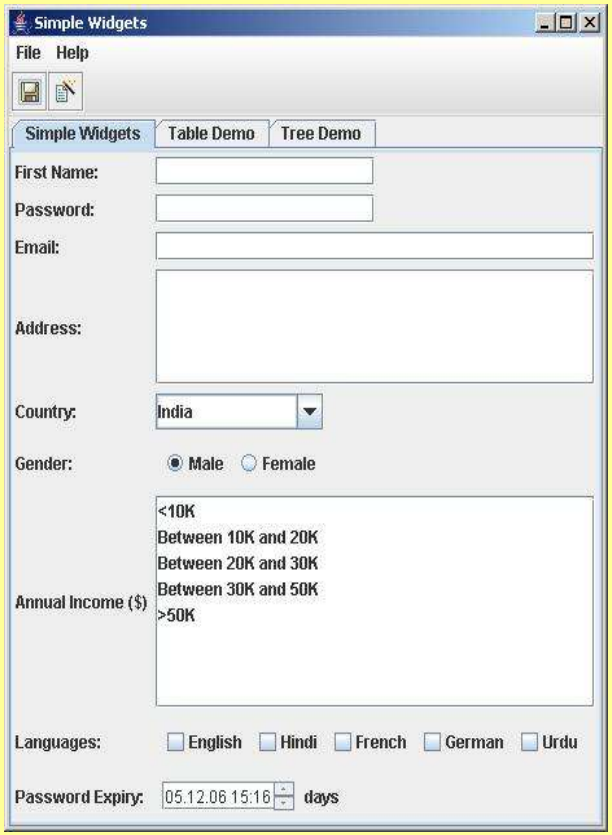
- Bestimmter Anforderungstyp (Cross-Cutting Concern) läßt sich nicht von anderen Anforderungen separieren
- Cross-Cutting Concern hebt die Kapselung von Funktionen auf und ist über viele Komponenten verstreut
- Beispiel: Logging
- AOP erweitert bisherige Programmierparadigmen und kapselt Cross-Cutting Concerns
- Erstmals 1997 von Gregor Kiczales vorgestellt

Javassist und AOP

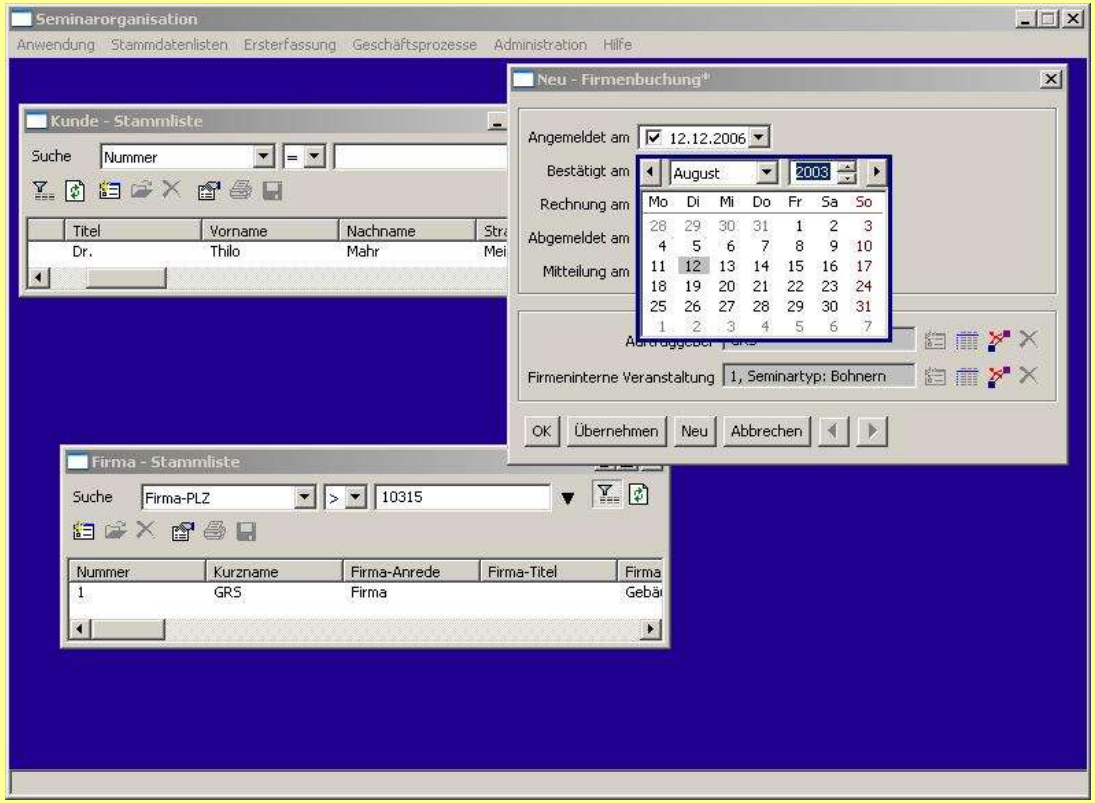
- Aspektdefinition erfolgt getrennt von Basiskomponenten in einer Aspektsprache (aspect language)
- Aspekt-Code wird an wohldefinierten Punkten (join points) mit den Basiskomponenten "verwoben"
  - Methodeneintritt/-austritt, Werfen einer Ausnahme usw.
- Anwendung in ATOSj z.B. bei Überdeckungsanalyse

Verwendete Testobjekte

Swing



SWT



Verwendete Testobjekte

- Swing: Simple Widgets
  - Zu Marathon mitgeliefertes Testobjekt
  - Enthält alle Swing Standardelemente
- SWT: Seminarorganisation
  - Verwendet in der Software-Engineering Vorlesung von Prof. Bothe
  - Lehrstuhl wünschte eigene Implementation
  - Entwicklung im Rahmen einer Tutorentätigkeit am Lehrstuhl

Testobjekt Seminarorganisation

- System für das Managment einer Firma, die Seminare als Dienstleitung anbietet

Typische Verwaltete Objekte:

  - Personen und Firmen
  - Seminartypen und Veranstaltungen
  - Beziehungen dieser Objekte untereinander

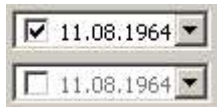
- Verwaltung mittels Datenbank

Testobjekt Seminarorganisation

- Zukünfige Verwendung in der Lehre
- Entwicklung parallel mit ATOSj
- Gegenseitiger Funktionstest

Custom-Component CalendarControl

- Auswahl eines Tages



- Aufklappen eines Kalenders zur Einstellung des Datums


```
ACTION,"Neu-Firma*",CALENDAR,"CpBirthDayCalendar",PROPERTY,"open","true"
ACTION,"Neu-Firma*",CALENDAR,"CpBirthDayCalendar",PROPERTY,"date","26.03.1955"
```

- Syntax ohne Custom-Control deutlich komplizierter

Ausblick

- Trotz Erreichtem weitere Erweiterungen denkbar:
  - Weitere Erweiterung von HTS zur Nutzung von Variablen in Nutzeraktionen

```
READ,"Neu-Kunde*",EDITBOX,"NumberText",NUM,"ClientId"
ACTION,"Neu-Dozent*",EDITBOX,"ClientIdText",EDIT,"ClientId"
```

  - Unterstützung der von Eclipse verwendeten eigenen Components (Paket swt.custom)

Ausblick

- Trotz Erreichtem weitere Erweiterungen denkbar:
  - Unterstützung weiterer Sprachen durch Sprachdateien
  - Hinzufügen der in ATOS implementierten Funktionen
    - Spezielle Erweiterung für Windows
    - ATOS als Bibliothek, die von ATOSj genutzt wird
    - Nutzung mittel Java-Native Interface
  - Umstellung der SWT Implementation auf Javassist

Erreichte Ziele

- Gestellte Aufgabe vollständig gelöst
- Darüber hinaus diverse neue Möglichkeiten
- Eingängliche und gut strukturierte Implementation ist gute Basis für weitere Entwicklungsmöglichkeiten
  - JavaDoc