Automatisierte Ermittlung von Subsystemschnittstellen

Worum geht es ?

- *Programme (z.B. XCTL-System) sind in Subsysteme aufgeteilt*
- *Schnittstellen beschreiben die (von anderen Subsystemen) nutzbaren Elemente (Funtkionen, Klassen, ...)*
- *bei Reverse-Engineering:*
  - *noch keine Subsysteme vorhanden*
  - *oder Subsystemeinteilung nicht optimal*
  - *-> Schnittstellen müssen gefunden werden*

Rückblick Studienarbeit:

- *Erstellen von Subsystem-Headerdateien:*
  - *Headerdatei als Subsystemschnittstelle*
  - *feste Vorgehensweise:*
    - *außerhalb des Subsystems #include-Anweisungen auskommentieren*
    - *versuchsweise compilieren*
    - *Fehlermeldungen auswerten*
    - *benötigte Deklarationen in neue Headerdatei verschieben*
  - *Aufwand:*
    - *umständlich*
    - *zeitaufwendig*
    - *fehleranfällig*

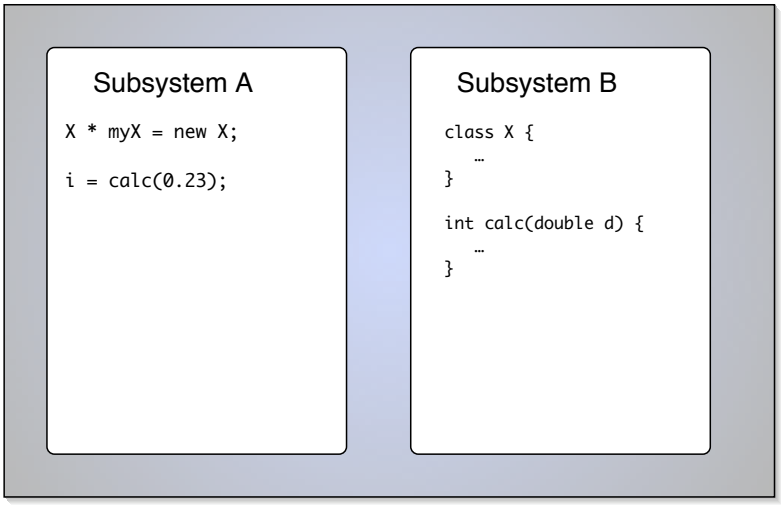
Automatisierung

- *Idee: Tool zur Bestimmung der Schnittstellen*
- *Vorteile:*
  - *schnelle Bestimmung der Schnittstellen*
  - *Vergleich von Subsystemeinteilungen möglich*

Ansatz

- *grundlegendes Problem: Referenzen finden*

  "Welche Symbole (Klassen, Funktionen, etc.) eines Subsystems werden in anderen Subsystemen benutzt?"



Ansatz

- *Sourcecode muss geparst werden -> zu umständlich*
- *Idee: vorhandenes Tool nutzen, welches diese Referenzen ermittelt*
- *Betrachtung von Sourcecode-Analyse-Tools:*

SNiFF+ 4.0

WIND RIVER

objectiF®

microTOOL
making IT better

objectiF

- *unterstützt großen Teil des Softwarentwicklungsprozesses*
  Use-Case-, Klassen-, Aktivitätsdiagramme, Dokumentation, Sourcecode
- *Reverse-Engineering Komponente*
- *Erweiterungemöglichkeiten:*
  COM-Interface für VisualBasic
- *ermittelt Referenzen*
- *aber:*
  - *Einlesen des XCTL-Systems mit Problemen verbunden*
  - *Informationen über Referenzen nicht von außen nutzbar*

SNiFF+

- *Cross-Referencer*
- *SNiFF-API:*
  - *Java Schnittstelle*
  - *liefert sämtliche vorhandene Informationen über:*
    - *das Projekt allgemein*
    - *Dateien*
    - *Symbole*
    - *Referenzen*

SNiFF-API

- *class SniffController:*
  - *stellt Verbindung zu SNiFF her:*

```
connect();
disconnect();
```

  - *startet Datenübertragung:*

```
getFileData();
getProjectData();
```

- *class SymbolHandler:*
  - *Funktionen, die Informationen empfangen:*

```
handleClass();
handleFunction();
handle...();
```

- *FileHandler, PreprocessorHandler, ProjectHandler, ReferenceHandler, ScopeHandler*

SNiFF+

- *Cross-Referencer*
- *SNiFF-API:*
  - *Java Schnittstelle*
  - *liefert sämtliche vorhandene Informationen über:*
    - *das Projekt allgemein*
    - *Dateien*
    - *Symbole*
    - *Referenzen*
- *problemloses Einlesen der XCTL-Quellen*

Toolentwicklung

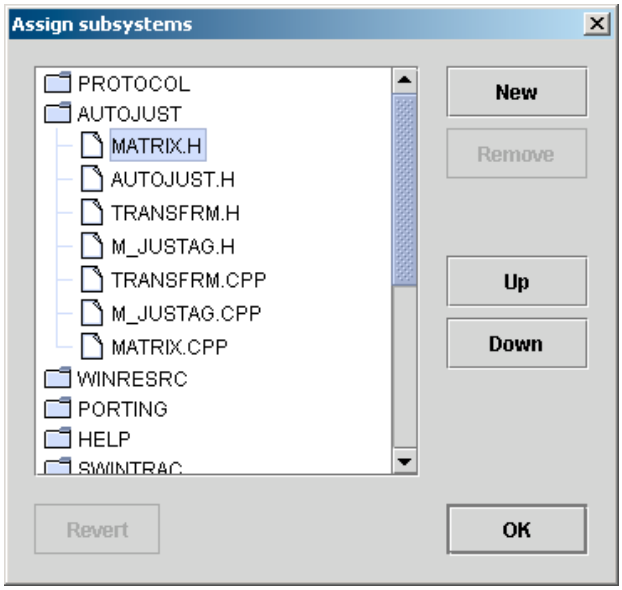
- *Arbeitsweise:*
  - *Verbindung zu SNiFF+ herstellen*
  - *Dateiinformationen übertragen*
  - *Zuordnung der Subsysteme*
  - *Symbol- und Referenzinformationen übertragen*
  - *Generierung der Schnittstellen*
  - *Ausgabe der Schnittstellen*

Anwendung

- *Programm starten*
- *Projekt öffnen (Sniff .proj-Datei)*
- *Subsystemzuordnung bearbeiten*
- *Berechnung abwarten*
- *Ergebnisse betrachten*

Subsystemzuordnung

- *jede Datei wird einem Subsystem zugeordnet*
- *erste Zuordnung wird von Verzeichnisstruktur bestimmt*
- *Dialog erlaubt Veränderung der Zuordnung*

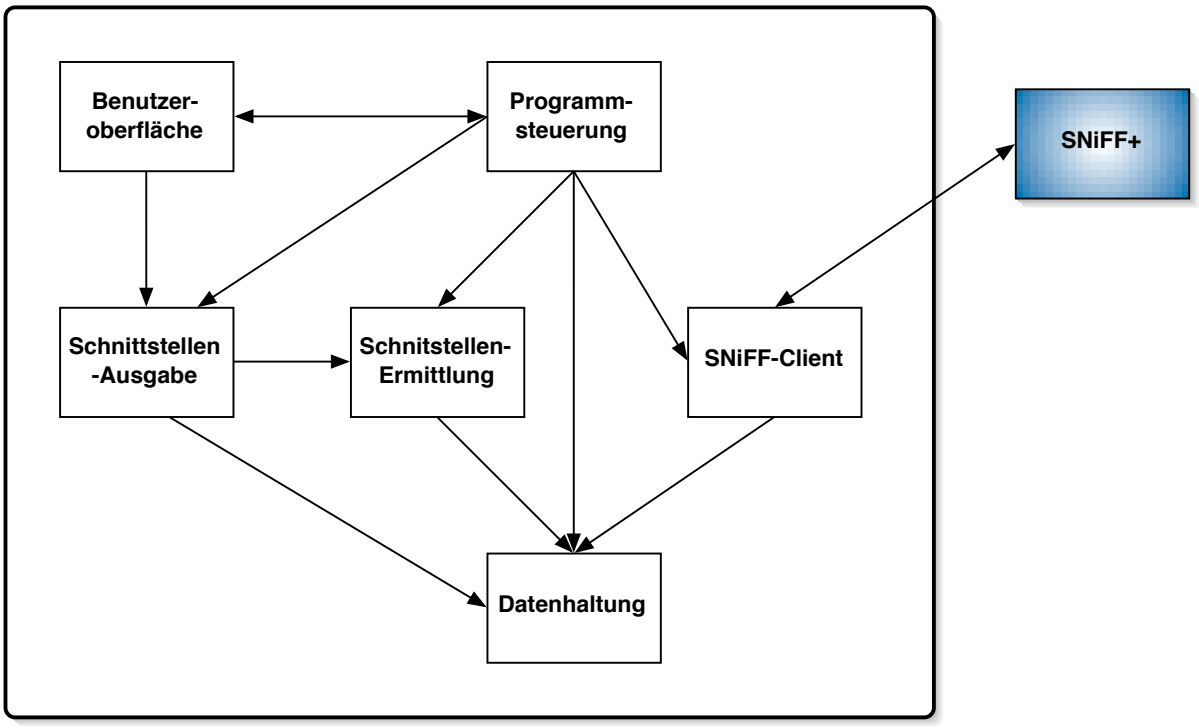


Ausgabe der Ergebnisse

- *tabellarisch*
- *Export in Textdatei*

Type	Name	File
define	DegToRad	DATAVISA.H
define	ArToRLx	DATAVISA.H
define	ArToRLy	DATAVISA.H
class	TCurve	DATAVISA.H
enum	TOutputType	DATAVISA.H
class	TPlotData	DATAVISA.H
class	TBitmapSource	DATAVISA.H
typedef	LPCurve	DATAVISA.H
class	TDataBase	DATAVISA.H
typedef	LPDataBase	DATAVISA.H
struct	TDisplay	DATAVISA.H
typedef	TKSystem	DATAVISA.H
enum	TSaveFormat	DATAVISA.H
typedef	TData	M_DATA.CPP
struct	TCoorSystem	DATAVISA.H
struct	TBMContens	DATAVISA.H
struct	CPoint	DATAVISA.H
class	TCurveShowParam	DATAVISA.H
variable	lpDataBase	M_CURVE.CPP
enum	TOrder	DATAVISA.H

Architektur



Schwierigkeiten

- *Makros*
- *externe Variable*
- *Probleme mit SNiFF+:*
  - *SNiFF-API schlecht dokumentiert*
  - *Parser/Cross-Referencer offensichtlich nicht fehlerfrei:*
  - *Operatorüberladung*
  - *undefined references*

Test

- *getrennter Test von Funktionalität und Oberfläche*
- *Funktionalität:*
  - *Testsystem:*
    einfaches C++ Programm, zwei Subsysteme
    -> Schnittstelle ablesbar
  - *XCTL:*
    Vergleich der Ergebnisse der Studienarbeit (Headerdateien) mit berechneten Schnittstellen
  - *beide Tests erfolgreich*
- *Oberfläche*
  - *Testfälle: Aktionen und erwartete Reaktionen,*
    jedes GUI-Element wird mindestens einmal aktiviert
  - *auch erfolgreich*

Erweiterungsmöglichkeiten

- *Abspeichern der Subsystemzuordnung*
- *erweiterte Ausgabemöglichkeiten*
- *Sprachumfang: C++ - Namespaces, Java*