Software Wartung und Evolution

Teil 3: Restructuring, Reengineering

Harald Gall
Institut für Informatik
Universität Zürich
http://seal.ifi.unizh.ch

Universität Zürich

s.e.a.l.
software evolution & architecture lab

Lecture 3

- Restructuring
- Reengineering
- Re-Architecting
  - Reengineering vom prozeduralen ins objekt-orientierte Paradigma am Beispiel von CORET

Fallbeispiel: „Adaption auf der falschen Abstraktionsstufe“

- Datenbankmodell
  - Datenbankfeld: „SA_EINZEL“
    - Vermutete Semantik
      - SA ... Sportart
      - EINZEL … Einzelsportart
    - Typ: char(1), keine Constraints
    - Wertbelegung in der Datenbank
      - „J“ ... Folgerung: Ja, Sportart ist eine Einzelsportart
      - „N“ ... Folgerung: Nein, Sportart ist keine Einzelsportart
      - **„T“ ... Überraschung: Sportart ist eine „Tennissportart“** ☹

Restructuring

Restructuring: Definition

- Restructuring is the transformation of
  - one representation form to another
  - at the same relative abstraction level,
  - while preserving the subject system's external behavior (functionality and semantics).

[Chikofsky/Cross]

Restructuring

- Abstraktionsniveau bleibt erhalten
- Code-to-code transformation
  - Beispiele
    - Ersetzung von goto statements
    - IF zu CASE Transformation
    - Verbesserung der Modularisierung
- Data-to-data transformation
  - Datennormalisierung (z.B. Transformation in 3. Normalform)
- Ziele
  - Verbesserte Struktureigenschaften, Modularisierung
  - Dadurch erhöhte Lesbarkeit, Testbarkeit, Effizienz

Restructuring: Historie

- 1966 Boehm/Jacobini
  - Jeder synchrone Ablauf eines Algorithmus kann mit 3 Konstrukten a) Sequenz b) Selektion c) Iteration logisch äquivalent zu hergebrachten mit goto programmierten Formen dargestellt werden
- 1968 Dijkstra (goto statement considered harmful)
  - Programme ohne goto sind übersichtlicher und besser lesbar
- 1972 Ashcroft und Manna
  - Alle Kontrollstrukturen in einem Programm können durch einen Algorithmus in while-Strukturen ersetzt werden

Restructuring: Historie

- 1975 Erste Restructuring Tools kommen auf den Markt
  - „Structured Engine“ für Fortran
  - Neater/2 für PL/I
- 1980 Belady et al
  - „A graphic representation of structured programs“
  - Bedeutsam für die grafische Unterstützung des Restrukturierungsprozesses
- Modern
  - Refactoring (entspricht Restructuring im OO Paradigma)

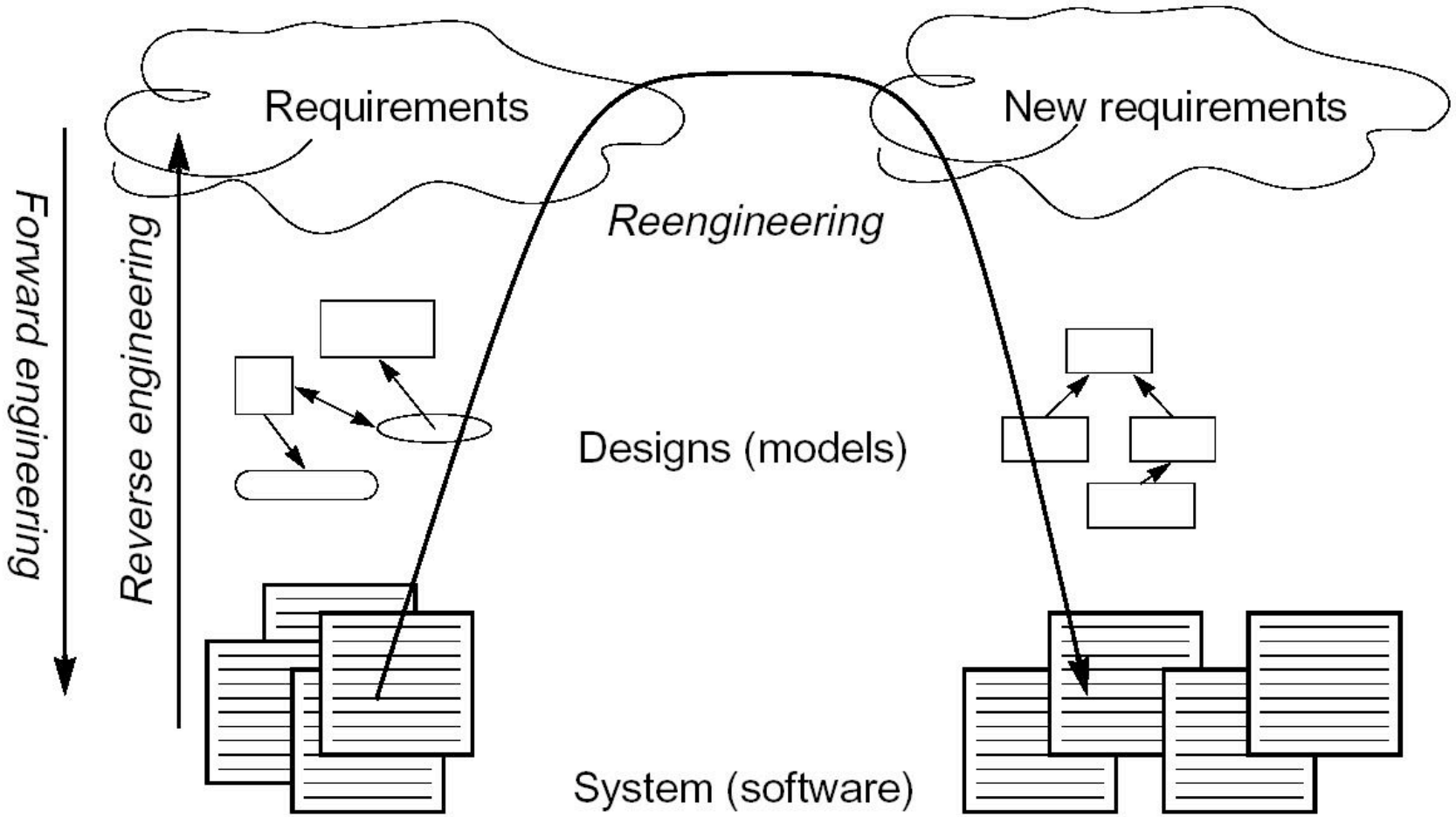
Re-Engineering

Re-Engineering: Definition

- Re-Engineering ist die Untersuchung und Änderung eines bestehenden Systems mit dem Ziel, das System in einer neuen, geänderten Form zu implementieren
  - Strukturiert in Reverse Engineering und Forward Engineering Phase

[Chikofsky/Cross]

Reverse and Reengineering



Re-Engineering: Probleme

- Komplexe Systeme können meist nur schrittweise migriert werden
  - Anbindung über *forward*- und *reverse*-gateways
- Reverse Engineering Ergebnisse sind unter Umständen dürftig
  - Fehlendes Know How, Werkzeuge
  - Hohe Komplexität der untersuchten Systeme
  - Zeitdruck
  - ➢ Im durch Forward Engineering erstellten System fehlt daher oft Funktionalität

Reengineering: Examples

- Data migration
  - Flat Files Data Repository to Database
- Programming language migration
  - e.g. 3GL to 4GL (Cobol to C++ or Java)
- User interface migration
  - Command line to GUI
- Procedural to object-oriented paradigm
  - „Re-architecting“

Universität Zürich

Objekt-Orientiertes Re-Architecting

s.e.a.l.
software evolution & architecture lab

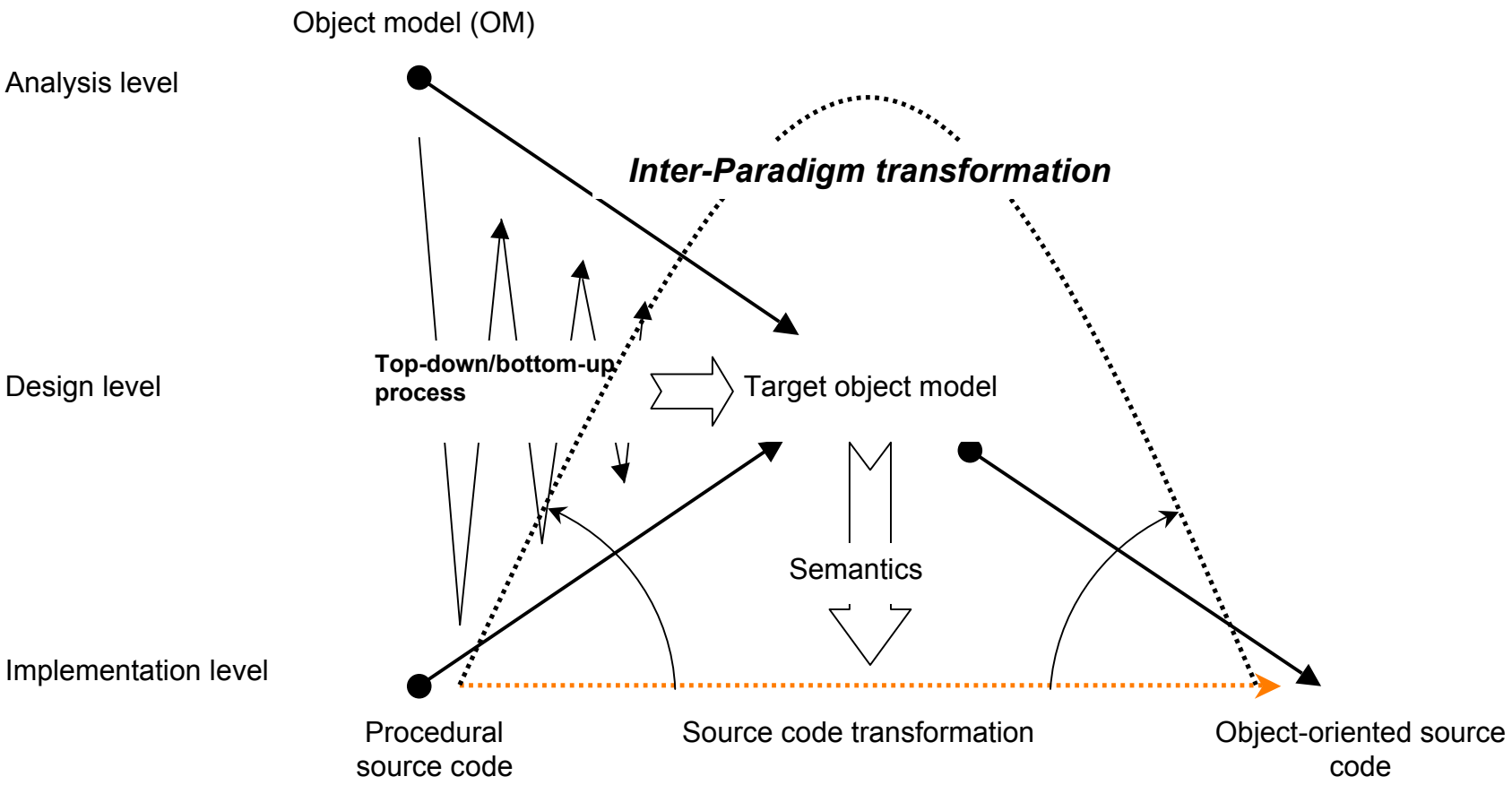
Objekt-Orientiertes Re-Architecting

- Definition
  - Unter objekt-orientiertem Re-Architecting versteht man die Transformation eines nicht objekt-orientieren Ausgangssystems in ein objekt-orientiertes Zielsystem
- Die inter-paradigmatische (i.e. Ausgangsparadigma ungleich Zielparadigma) Transformation wird als „Re-Architecting“ bezeichnet
- Automatisierungsstufen einer Transformation
  - Manuell
  - Semi-automatisch
  - Vollautomatisch

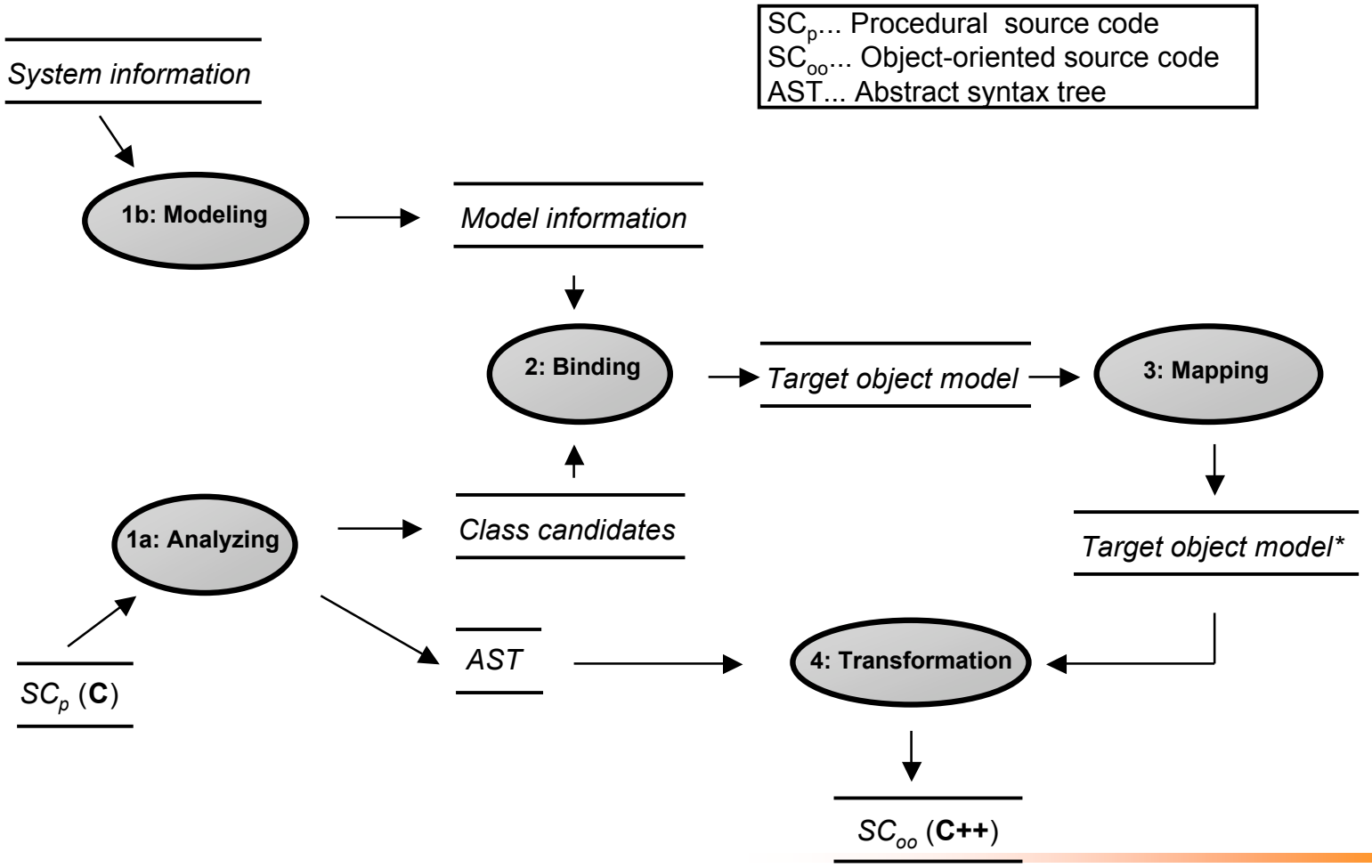
Methoden des OO Re-Architecting

- COREM
  - Capsule Oriented Reverse Engineering Method
    - TU Wien (Gall, Klösch)
    - Pascal nach OO-Pascal
- CORET
  - Capsule Oriented Reverse Engineering Technique
    - TU Wien (Gall, Klösch, Weidl-Rektenwald)
    - C nach C++
    - Prototyp Tool zur Semi-Automation

CORET: Ansatz



CORET: Prozess



Analyse: Identifikation von Klassen-kandidaten

- Parsen des C source codes und Erstellen eines Abstract Syntax Trees (AST)
- Objekterkennung
  - Function driven objectification
    - Objects are groups of procedures/functions
  - Data driven objectification
    - Objects are data with procedures/functions working on the data
    - e.g. [Liu et al.]
  - ➢ Automatische Identifikation von „Klassenkandidaten“

CORET: Modeling

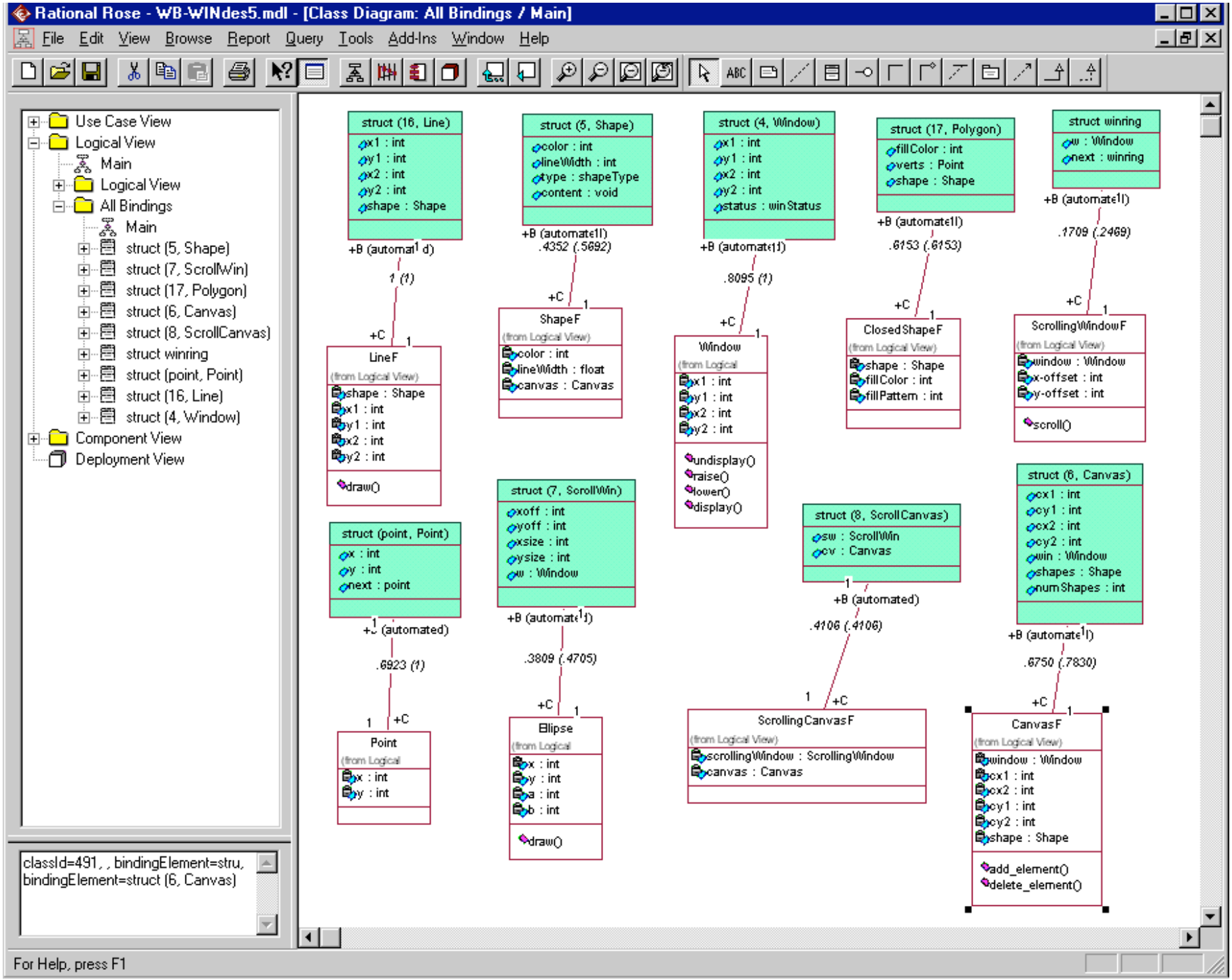
- Aus den Requirements wird ein Objektmodell OM_r (Klassendiagramm) „forward“ erstellt
- Unter Berücksichtigung des Source Code, der Dokumentation und Experten wird OM_r in ein Objektmodell OM_d (Klassendiagramm) verfeinert
  - Die Modellierungsschritte werden manuell durchgeführt
  - Die Ergebnisse werden in Rational Rose in Form von UML Klassendiagrammen gespeichert

CORET: Binding – Schritt I

- Mittels eines Ähnlichkeitsmaßes wird die „Ähnlichkeit“ zwischen den Klassenkandidaten und Klassen aus OM_d ermittelt
  - Automatisch
  - Kriterien des Ähnlichkeitsmaßes
    - Namensinformation (Vergleich auf Trigramm Basis)
    - Typinformation (Typen der Instanzvariablen, Methoden-Parameter)
    - Strukturinformation (Anzahl der Instanzvariablen und Methoden)

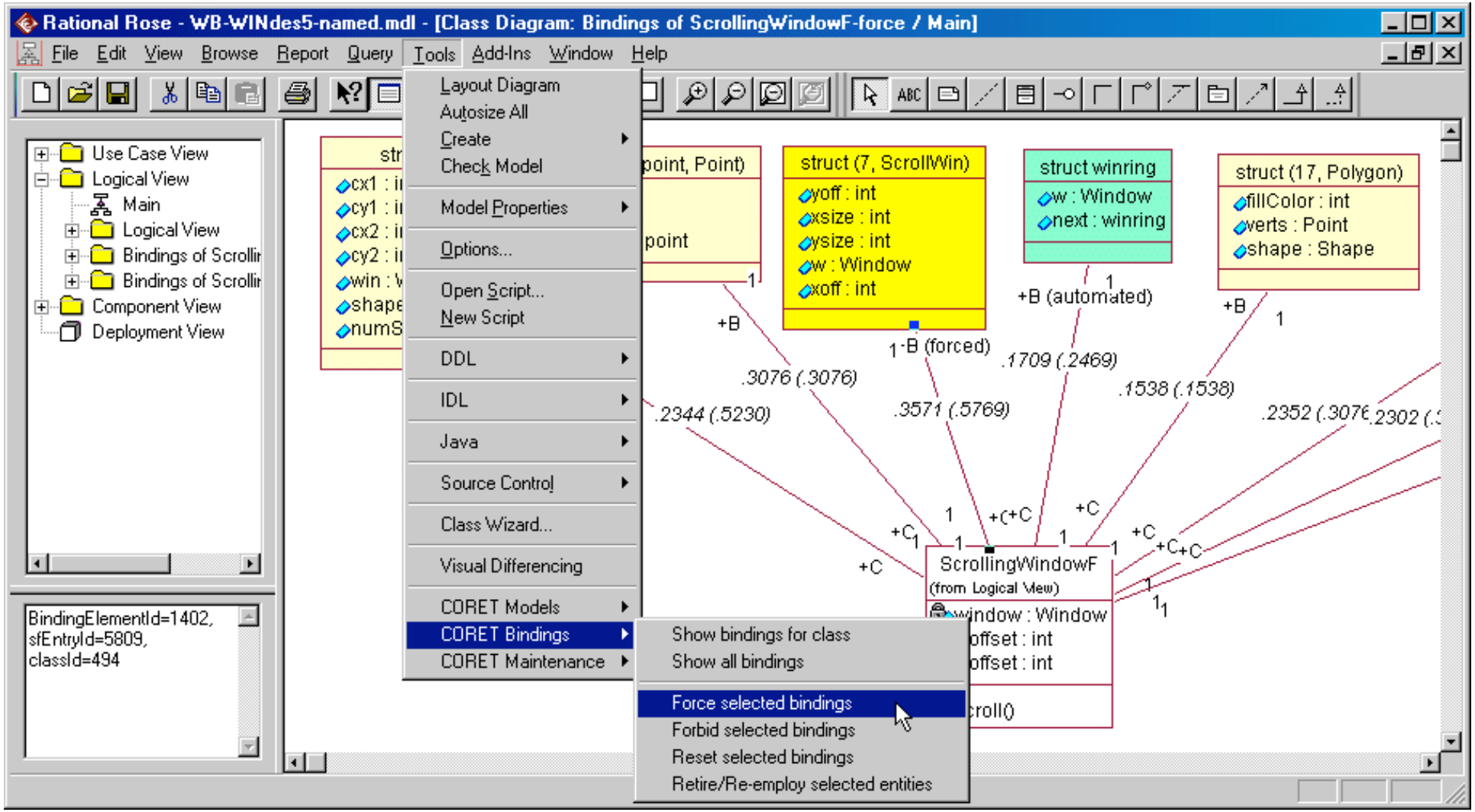
CORET: Binding – Schritt IIa

- Aufgrund der automatisch berechneten Ähnlichkeitswerte wird automatisch eine Reihe von Zuordnungen Klassenkandidat – Klasse getroffen
  - Ergebnis: Menge temporärer Bindungen Klassenkandidat - Klasse


Rational Rose - WB-WINdes5.mdl - [Class Diagram: All Bindings / Main]
File Edit View Browse Report Query Tools Add-Ins Window Help
Use Case View
Logical View
Main
Logical View
All Bindings
Main
struct (5, Shape)
struct (7, ScrollWin)
struct (17, Polygon)
struct (6, Canvas)
struct (8, ScrollCanvas)
struct winring
struct (point, Point)
struct (16, Line)
struct (4, Window)
Component View
Deployment View
classId=491, , bindingElement=stru,
bindingElement=struct (6, Canvas)
For Help, press F1
struct (16, Line)
x1 : int
y1 : int
x2 : int
y2 : int
shape : Shape
+B (automated)
1 (1)
+C
LineF
(from Logical View)
shape : Shape
x1 : int
y1 : int
x2 : int
y2 : int
draw()
struct (5, Shape)
color : int
lineWidth : int
type : shapeType
content : void
+B (automated)
.4352 (.5692)
+C
ShapeF
(from Logical View)
color : int
lineWidth : float
canvas : Canvas
struct (4, Window)
x1 : int
y1 : int
x2 : int
y2 : int
status : winStatus
+B (automated)
.8095 (1)
+C
Window
(from Logical
x1 : int
y1 : int
x2 : int
y2 : int
undisplay()
raise()
lower()
display()
struct (17, Polygon)
fillColor : int
verts : Point
shape : Shape
+B (automated)
.6153 (.6153)
+C
ClosedShapeF
(from Logical View)
shape : Shape
fillColor : int
fillPattern : int
struct winring
w : Window
next : winring
+B (automated)
.1709 (.2469)
+C
ScrollingWindowF
(from Logical View)
window : Window
x-offset : int
y-offset : int
scroll()
struct (point, Point)
x : int
y : int
next : point
+C (automated)
.6923 (1)
+C
Point
(from Logical
x : int
y : int
struct (7, ScrollWin)
xoff : int
yoff : int
xsize : int
ysize : int
w : Window
+B (automated)
.3809 (.4705)
+C
Ellipse
(from Logical
x : int
y : int
a : int
b : int
draw()
struct (8, ScrollCanvas)
sw : ScrollWin
cv : Canvas
+B (automated)
.4106 (.4106)
+C
ScrollingCanvasF
(from Logical View)
scrollingWindow : ScrollingWindow
canvas : Canvas
struct (6, Canvas)
cx1 : int
cy1 : int
cx2 : int
cy2 : int
win : Window
shapes : Shape
numShapes : int
+B (automated)
.6750 (.7830)
+C
CanvasF
(from Logical View)
window : Window
cx1 : int
cx2 : int
cy1 : int
cy2 : int
shape : Shape
add_element()
delete_element()

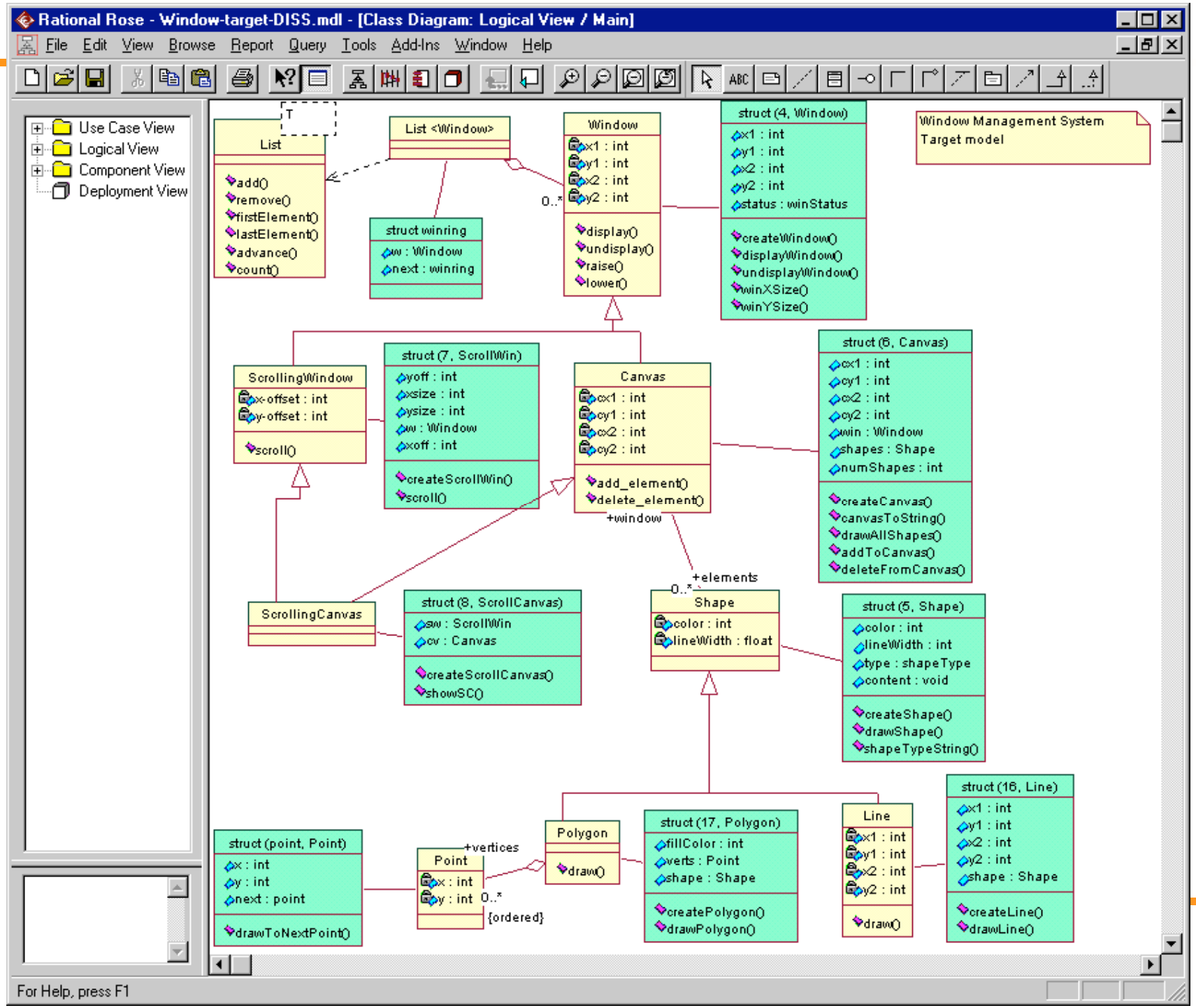
CORET: Binding – Schritt IIb

- Die automatisch berechneten Bindungen Klassenkandidat – Klasse werden in Rational Rose grafisch als Vorschläge angezeigt
  - Der „Reuse Engineer“ geht durch alle temporären Bindungen und
    - Fixiert die Bindung (forced binding) oder
    - Verbietet die Bindung (forbidden binding)
  - Zwischen den noch nicht endgültig durch den Reuse Engineer zugeordneten Klassenkandidaten und Klassen startet die automatische Bindung aufgrund der Ähnlichkeitswerte erneut
- Man spricht von iterativ-inkrementellem, semi-automatischem Binding

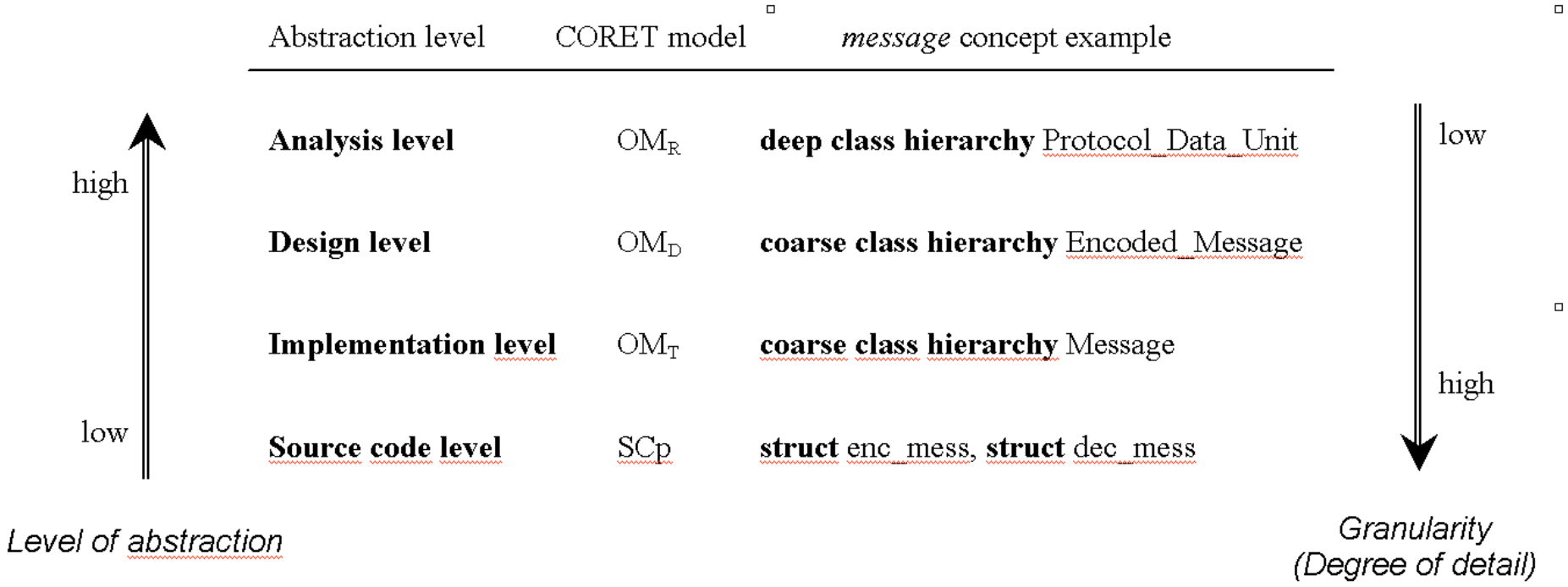

Rational Rose - WB-WINdes5-named.mdl - [Class Diagram: Bindings of ScrollingWindowF-force / Main]
File Edit View Browse Report Query Tools Add-Ins Window Help
Layout Diagram
Autosize All
Create
Check Model
Model Properties
Options...
Open Script...
New Script
DDL
IDL
Java
Source Control
Class Wizard...
Visual Differencing
CORET Models
CORET Bindings
CORET Maintenance
Show bindings for class
Show all bindings
Force selected bindings
Forbid selected bindings
Reset selected bindings
Retire/Re-employ selected entities
Use Case View
Logical View
Main
Logical View
Component View
Deployment View
BindingElementId=1402,
sfEntryId=5809,
classId=494
struct (7, ScrollWin)
yoff : int
xsize : int
ysize : int
w : Window
xoff : int
struct winring
w : Window
next : winring
struct (17, Polygon)
fillColor : int
verts : Point
shape : Shape
ScrollingWindowF
(from Logical View)
+B (forced)
+B (automated)
.3076 (.3076)
.2344 (.5230)
.3571 (.5769)
.1709 (.2469)
.1538 (.1538)

CORET: Binding – Ergebnis

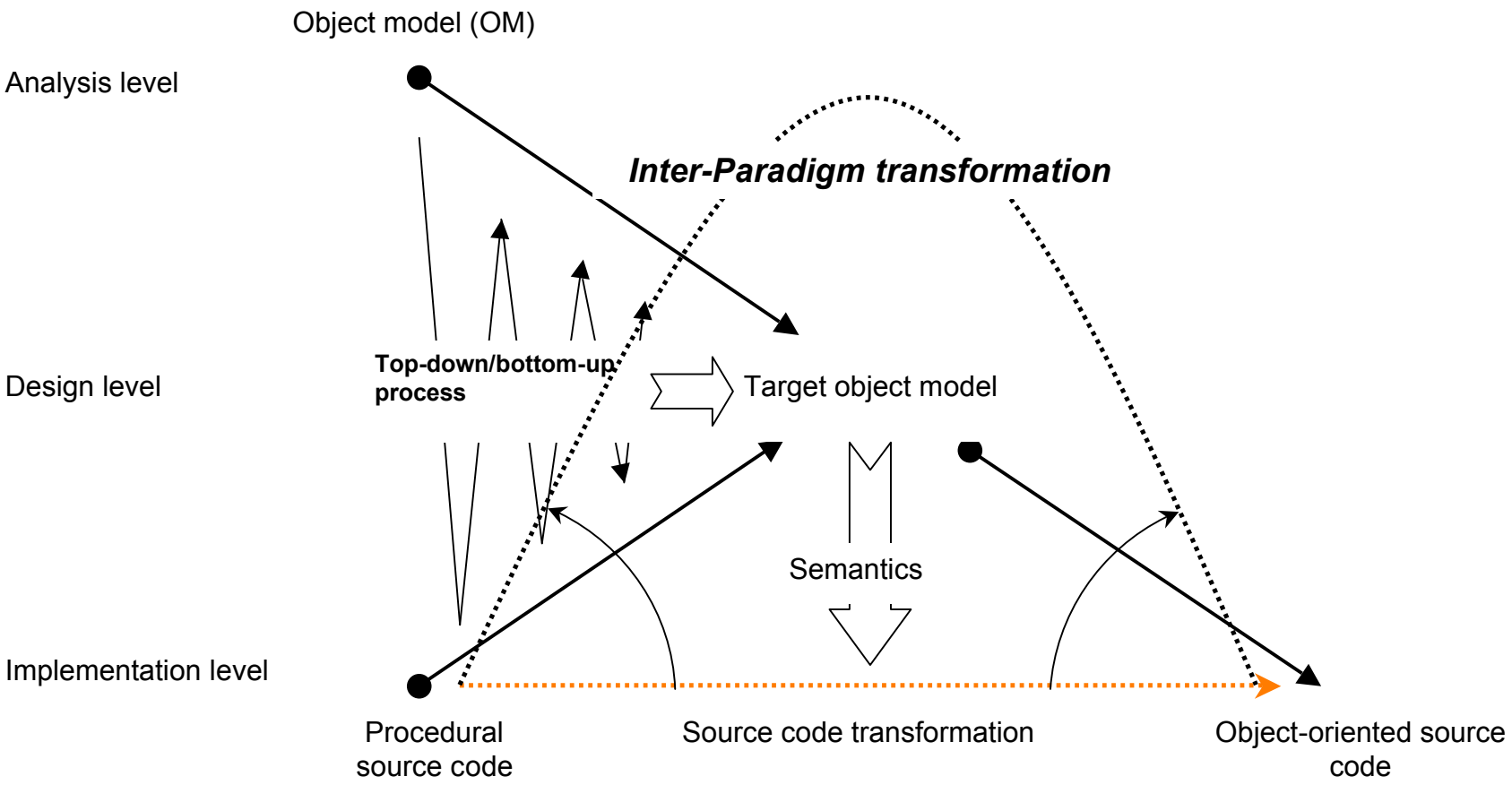
- Ergebnis der Binding Phase ist der *Binding Table*, der eine Zuordnung von Konzepten zu Source Code repräsentiert
  - z.B. wird dem Konzept „Rechnung“ Source Code in Form von Modulen mit Variablen und Prozeduren zugewiesen, die das Konzept *Rechnung* im System implementieren
  - Damit ist das Binding ein Ansatz, um das *Concept Assignment Problem* nach [Biggerstaff] zu lösen
    - i.e. das Problem, wie man Source Code real-world Konzepte aus der Application Domain zuordnen kann


Rational Rose - Window-target-DISS.mdl - [Class Diagram: Logical View / Main]
File Edit View Browse Report Query Tools Add-Ins Window Help
Use Case View
Logical View
Component View
Deployment View
T
List
add()
remove()
firstElement()
lastElement()
advance()
count()
List <Window>
struct winring
w : Window
next : winring
Window
x1 : int
y1 : int
x2 : int
y2 : int
display()
undisplay()
raise()
lower()
0..*
struct (4, Window)
x1 : int
y1 : int
x2 : int
y2 : int
status : winStatus
createWindow()
displayWindow()
undisplayWindow()
winXSize()
winYSize()
Window Management System
Target model
ScrollingWindow
x-offset : int
y-offset : int
scroll()
struct (7, ScrollWin)
yoff : int
xsize : int
ysize : int
w : Window
xoff : int
createScrollWin()
scroll()
Canvas
cx1 : int
cy1 : int
cx2 : int
cy2 : int
add_element()
delete_element()
+window
struct (6, Canvas)
cx1 : int
cy1 : int
cx2 : int
cy2 : int
win : Window
shapes : Shape
numShapes : int
createCanvas()
canvasToString()
drawAllShapes()
addToCanvas()
deleteFromCanvas()
ScrollingCanvas
struct (8, ScrollCanvas)
sw : ScrollWin
cv : Canvas
createScrollCanvas()
showSC()
+elements
0..*
Shape
color : int
lineWidth : float
struct (5, Shape)
color : int
lineWidth : int
type : shapeType
content : void
createShape()
drawShape()
shapeTypeString()
struct (point, Point)
x : int
y : int
next : point
drawToNextPoint()
+vertices
Point
x : int
y : int
0..*
{ordered}
Polygon
draw()
struct (17, Polygon)
fillColor : int
verts : Point
shape : Shape
createPolygon()
drawPolygon()
Line
x1 : int
y1 : int
x2 : int
y2 : int
draw()
struct (16, Line)
x1 : int
y1 : int
x2 : int
y2 : int
shape : Shape
createLine()
drawLine()
For Help, press F1

Concept Assignment Problem



CORET: Ansatz



CORET: Mapping

- Durch das Binding wurde Source Code an das Designmodell OM_d gebunden
- Das Designmodell OM_d wird dem Requirementsmodell OM_r angenähert
  - Einbringung von Vererbung
  - Transformation von Klassen in Design Patterns
  - Einbringung von Namensinformation
- Probleme
  - Viel Know-How nötig
  - Hohe Anforderung an die Tool-Unterstützung

CORET: Transformation

- Der Abstract Syntax Tree (AST) wird mittels der Information aus Binding und Mapping in den objekt-orientierten Zielcode transformiert.
  - Vollautomatisch
- Ergebnis
  - Nach OM_r strukturierter objekt-orientierter Code
- Refactoring kann zur weiteren Verbesserung des Zielsystems angewandt werden

CORET: Fazit

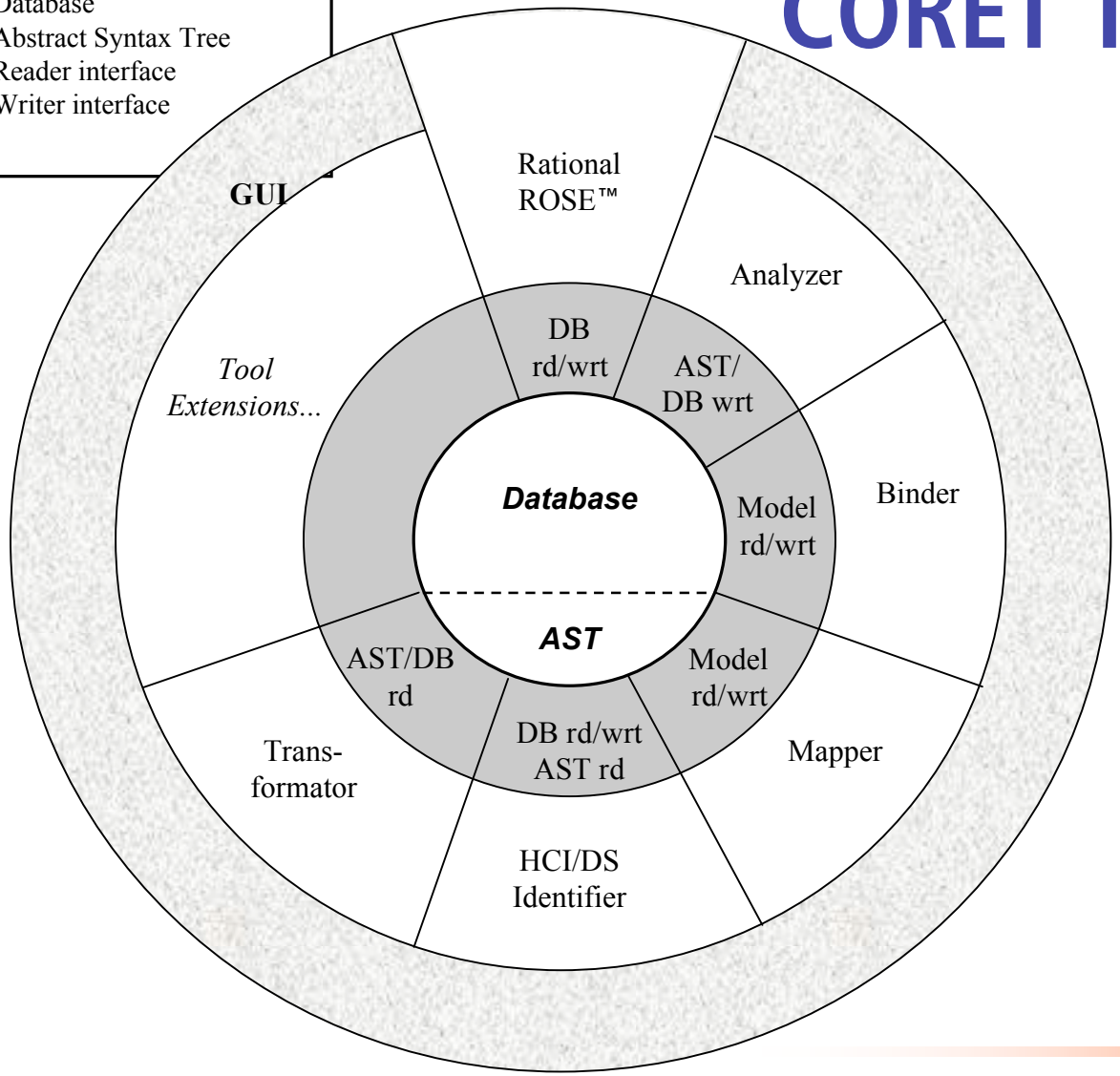
- Objektbildung erfolgt in CORET durch „Object driven objectification“
  - Existing knowledge of the application domain is used to define objects
- Daher
  - Qualitativ bessere Objekte und damit eine bessere Struktur/Architektur als durch Ansätze, die nicht auf Domain- und Human Knowledge zurückgreifen

CORET: Voraussetzungen

- CORET funktioniert gut, wenn
  - Der Source Code bereits eine *gute* Struktur aufweist
  - Genügend Dokumentation vorhanden ist
  - Zugriff auf Applikations- und Domainexperten möglich ist
  - Namensinformation ausreichend vorhanden ist
  - Die Klassen sich nicht zu ähnlich sind

CORET Toolset

HCI	Human Computer Interface
DS	Data Structure
DB	Database
AST	Abstract Syntax Tree
rd	Reader interface
wrt	Writer interface



Dimensionen der Neuentwicklung

- Neuentwicklung
  - From scratch / Auf der grünen Wiese
  - Durch Anforderungsanalyse, typisches Forward Engineering
- Re-Engineering
  - Reverse Engineering von
    - Design
    - Architektur
    - Anforderungen
  - Anschließendes Forward Engineering
- Transformation
  - Intra-paradigmatisch vs. inter-paradigmatisch

Zusammenfassung

- Restructuring
- Refactoring (oo)
- Reengineering
- Re-Architecting

- Funktions- und Semantik-Äquivalenz nach Transformationen
- Concept assignment problem
- Architektur <--> Source Code