

Wielowymiarowe rozdzielanie zagadnień a podejście komponentowe

Andrzej Krzywda^{1,2}

Tomasz Nazar^{1,2}

Ewa Gurbiel¹

¹Uniwersytet Wrocławski, Instytut Informatyki

²Siemens AG, CT, SE2, Architecture, Monachium

08.10.2004

Wielowymiarowe rozdzielanie zagadnień

- Motywacja - Siemens AG
- Problemy w zarządzaniu liniami produkcyjnymi
- Projekt TopoTree
- Cechy proponowanego rozwiązania
- Podsumowanie

Motywacja

- Siemens AG – linie produkcyjne
 - Wiele aplikacji o zbliżonych funkcjonalnościach
 - Różne platformy
 - Urządzenia mobilne
 - Komputery osobiste

Motywacja

- Siemens AG – linie produkcyjne
 - Podział na zespoły
 - Specyficzne platformy
 - Modele telefonów komórkowych
 - Systemy operacyjne
 - Specjaliści dziedzin
 - Programy biurowe
 - Gry

Motywacja

- Siemens AG – linie produkcyjne
 - Wiele wersji jednej aplikacji
 - Różne interfejsy użytkownika
 - Ograniczone zestawy funkcjonalności
 - Mechanizmy pamiętania danych

Każde wsparcie dla lepszego rozdzielania zagadnień
to wymierny zysk dla firmy!

Programowanie aspektowe

- Zagadnienia przecinające
- Zespół d/s programowania aspektowego :
 - Badanie zastosowań programowania aspektowego
 - Szkolenia pracowników
 - Porównania języków aspektowych

Wielowymiarowe rozdzielanie zagadnień
należy do
programowania aspektowego

Projekt TopoTree

- Motywacja:
 - Linie produkcyjne w Siemens AG
- Wymagania
 - Reprezentacja sieci urządzeń
 - Sieć urządzeń jest hierarchiczna
 - Opcjonalny interfejs użytkownika
- Wczesne wydanie pierwszej wersji

Klasa TopoTree

- Zapamiętanie korzenia drzewa urządzeń
- `setRoot(Node)`
- `fillUpData()`
- Wypełnienie drzewa przykładowymi węzłami

Klasa Node

- Nazwa węzła
- Zapamiętanie ojca
- `setName (String)`
- `setFather (Node)`

```
public class TopoTree {  
    Node root;  
  
    public static void main(String[] args) {  
        TopoTree topoTree = new TopoTree();  
        topoTree.fillUpData();  
    }  
    private void fillUpData() {  
        Node siemens = new Node("Siemens");  
        Node poland = new Node("Poland", siemens);  
        Node germany = new Node("Germany", siemens);  
        setRoot(siemens);  
        System.out.println("filled up!");  
    }  
}
```

```
public class Node {  
    String name;  
    Node father;  
    List children;  
  
    public Node(String string) {...}  
    public Node(String name, Node father) {...}  
  
    public void setFather(Node father) {...}  
    public void addChild(Node son) {...}  
}
```

Pierwsze wydanie

- Zaimplementowano podstawową funkcjonalność wymaganą do pierwszej wersji systemu
- Efektem działania jest tylko wypełnienie danymi drzewa urzędzeń

Graficzny interfejs użytkownika

- Opcjonalność wymagania
 - wersja bez GUI
 - wersja z GUI
- Wyświetlenie okienka z drzewem
- Użycie biblioteki Swing

Interfejs użytkownika – implementacja

- Klasa TopoTree
 - void **initGUI**()
 - Wywołanie wewnątrz metody **fillUpData**()
- Klasa Node
 - DefaultMutableTreeNode **asJTreeNode**()

TopoTree – zrzut ekranu



```
public class TopoTree {  
    Node root;  
  
    setRoot() {...}  
    getRoot() {...}  
    fillUpData() {...}
```

```
    void initGUI(){  
        ...  
    }
```

```
}
```

```
public class Node {  
    String name;  
    Node father;  
    Node children;  
  
    getName() {...}  
    getFather() {...}
```

```
    JTreeNode asJTreeNode(){  
        ..  
    }
```

```
}
```

GUI CORE

Jak uzyskać opcjonalność wymagań?

- Podział kodu źródłowego na dwie części odpowiedzialne za:
 - zagadnienia modelu
 - graficzny interfejs użytkownika

MODEL.CORE

```
public class TopoTree {
    setRoot() {...}
    getRoot() {...}
    fillUpData() {...}
}

public class Node {
    getName() {...}
    getFather() {...}
    List getChildren() {...}
    //..
}
```

GUI.CORE

```
public class TopoTree {
    initGUI(){
        //..
        new JTree(getRoot());
        //..
    }
}

public class Node {
    asJTreeNode() {...}
}
```

Problem

- Jak wywołać metodę znajdującą się w innym module?

Deklaratywna kompletność

- Wystarczy zadeklarować jednostki kodu, z których chcemy korzystać
 - Klasy
 - Metody
 - Pola

Język Hyper/J

- Wspiera deklaratywną kompletność
- Język aspektowy, IBM, New York
- Wielowymiarowe rozdzielanie zagadnień
- Bez modyfikacji języka Java
 - AspectJ rozszerza język Java
 - Nie wymaga specjalnych IDE
- Operuje na plikach .class

Deklaratywna kompletność w Hyper/J

- Deklarowanie kompletności:
 - `{throw new UnimplementedError(); }`
 - Klasy abstrakcyjne
 - Metody abstrakcyjne

```
public class TopoTree {  
    void initGUI() {  
        ...  
        new JTree(getRoot());  
        ...  
    }  
    Node getRoot() {  
        throw new UnimplementedError();  
    }  
}
```

Modularyzacja z Hyper/J

- Moduły wprowadzające kod nazywamy hiperwarstwami
- Każda hiperwarstwa jest deklaratywnie kompletna
- Skąd wiadomo jak połączyć oba moduły?
 - Opis kompozycji i konfiguracji aplikacji (hipermodułu)

Aplikacja = hipermoduł

- Lista hiperwarstw składających się na aplikację
- Strategia kompozycji
 - Kompozycja wg nazwy - mergeByName
- Szczegóły kompozycji
 - Equate class
 - Equate operation

TopoTree

bez interfejsu użytkownika

```
hypermodule TopoTree
```

```
hyperslices:
```

```
    Model.Core;
```

```
relationships:
```

```
    mergeByName;
```

TopoTree

wersja z interfejsem użytkownika

```
hypermodule TopoTreeWithGui
```

```
hyperslices:
```

```
    Model.Core,
```

```
    Gui.Core;
```

```
relationships:
```

```
    mergeByName;
```

```
equate operation
```

```
    Model.Core.fillUpData,
```

```
    Gui.Core.initGui;
```

Hiperwarstwy jako komponenty

- Opcjonalne
- Mogą być rozwijane niezależnie
- Użycie w innych aplikacjach
- Wsparcie dla Composite Applications

Ponowne wykorzystanie komponentu Gui.Core

- Hiperwarstwa Gui.Core używa terminologii
TopoTree

Hiperwarstwa Gui.Core

- Zmiana nazw klas:
 - TopoTree → TreeApplication
 - Node → TreeElement
- Wymusza zmiany w opisach hipermodułów

```
hypermodule TopoTreeWithGui
hyperslices:
    Model.Core,
    Gui.Core;
relationships:
    mergeByName;
equate class
    TopoTree,
    TreeApplication;
equate class
    Node,
    TreeElement;
```

Hipermoduł TopoTree

- Zalety
 - Możliwość powtórnego użycia hiperwarstwy GUI
- Wady
 - Bardziej skomplikowane definicje hipermodułów

Aplikacja FamilyTree

- Reprezentuje drzewo genealogiczne
- Klasy
 - FamilyTree
 - FamilyMember

Ponowne użycie Gui.Core

- Komponujemy FamilyTree.Core z Gui.Core
 - FamilyTree = TreeApplication
 - FamilyMember = TreeElement

```
hypermodule FamilyTree
hyperslices:
    FamilyTree.Core,
    Gui.Core;
relationships:
    mergeByName;
equate class
    FamilyTree,
    TreeApplication;
equate class
    FamilyMember,
    TreeElement;
```

FamilyTree – zrzut ekranu



Korzyści 1/2

- Wersje aplikacji dla różnych klientów
- Ponowne użycie daje oszczędność
- Szybkie prototypowanie z wykorzystaniem gotowych hiperwarstw interfejsu użytkownika
 - Pasek statusu
 - Pasek postępu
 - Panele informacyjne, edycyjne
- Z repozytorium hiperwarstw UI praca ogranicza się do implementacji modelu

Korzyści 2/2

- Wsparcie dla implementacji modułów przez różne zespoły
- Możliwość powtórnego wykorzystania komponentów
- Zarządzanie liniami produkcyjnymi
- Naprawianie błędów możliwe dla wszystkich istniejących wersji (konfiguracji)

Problemy

- Praca nad definicją hipermodułu
- Brak wsparcia narzędzi (IDE)
- Autorzy pracują nad wtyczką do Eclipse wspomagającą pracę z hiperwarstwami

Zbliżone techniki

- Programowanie aspektowe
 - (IBM) Concern Manipulation Environment
 - (IBM) AspectJ
 - (JBoss) JBossAOP
 - (Siemens) Caesar
 - (BEA WebLogic) AspectWerkz

Dziękujemy!