Hiearchical MVC (Model-view-controller)

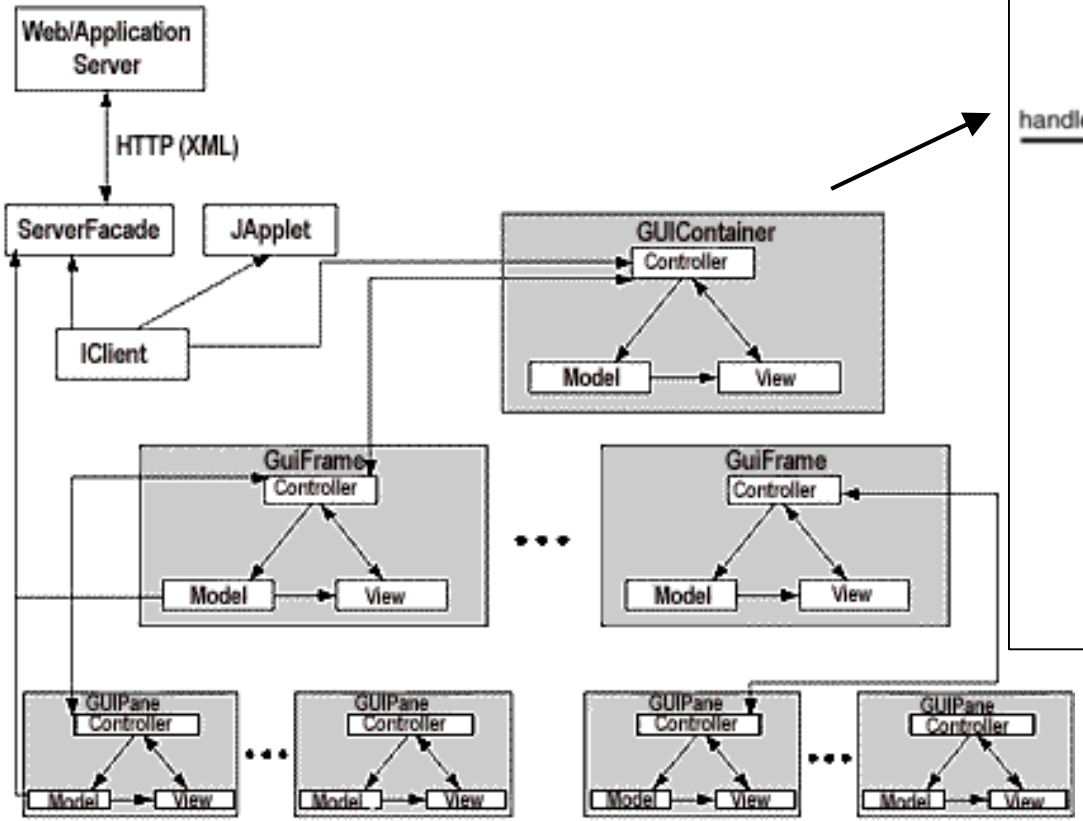
vs.

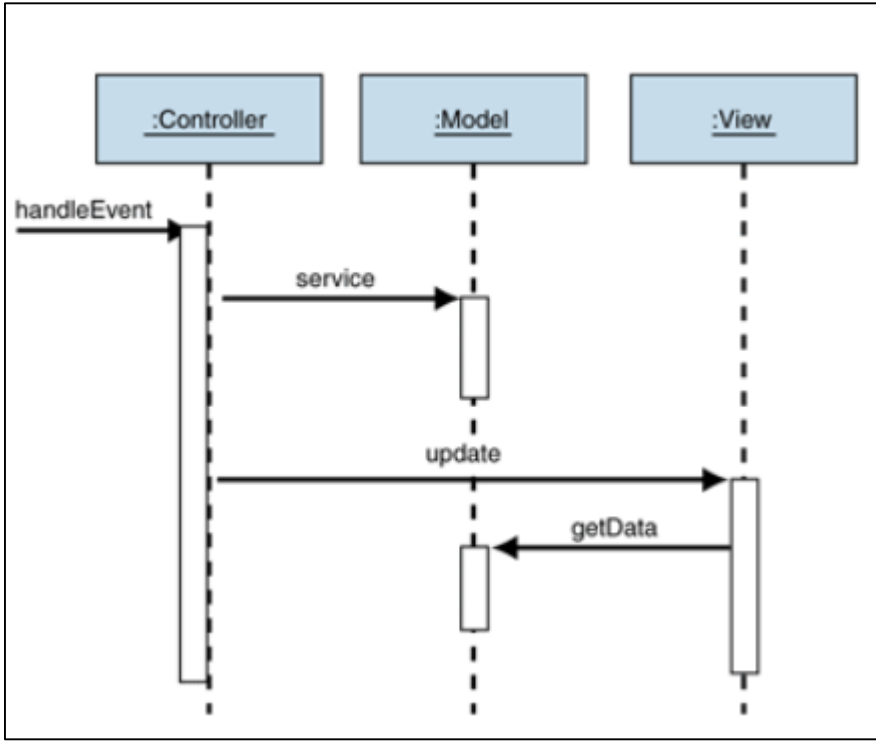
PAC (Presentation-abstraction-control)

HMVC – úvod

- **Problém**
  - MVC v určitých aplikacích nedostačující
- **Příklad: webová stránka s widgety**
  - Např. kalendář, hodnocení, okno s tweety na určitý #...
- **Jak na to?**
  - Více modulů/agentů (odpovídající MVC vzoru) rozdělených do 3 a více vrstev
    - Top-Level-Agent, Intermediate-Level-Agent, Bottom-Level-Agent (u PAC)

HMVC – příklad





HMVC – účastníci

- **Client**
  - Interaguje s View
- **Controller**
  - Zpracová události od View - input od uživatele.
  - Využivá návrhový vzor Chain of Responsibility, pokud Controller nedokáže zpracovat událost, tak přepošle tuto událost rodičovskému Controlleru atd.
- **Model**
  - Uchovává všechna data, stavy, logiku. Poskytuje rozhraní k manipulaci a získání dat, které uchovává.
  - Posílá notifikace do View (o změně stavu apod.)
- **View**
  - UI aplikace. View zpracovává události od Modelu a posílá notifikace Controlleru o změnu stavu.
  - Data potřebná k zobrazení si bere **přímo z Modelu.**

HMVC - implementace

- **Spodní vrstva**

```
public class ContentGUIPane extends javax.swing.JMenuBar implements ActionListener
{
    ...
    public void actionPerformed(ActionEvent actionEvent)
    {
        if(actionEvent.getActionCommand().equals("CLICK"))
        {
            ...
            AppEvent ae = new AppEvent(AppEvent.DATA_EVENT);
            ae.setMessage("HELLO_WORLD");
            ctlr_.handleAppEvent(ae);
        }...
    }
}
```

Spodní vrstva - pokračování

```
public class ContentGUIPaneController
{...
    public void handleAppEvent(AppEvent evt)
    {
        if(evt.isDataEvent())
            contentModel_.handleAppEvent(evt);
        else
         if(evt.isStatusEvent() || evt.isNavEvent())
            parentCtlr_.handleAppEvent(evt);
    }
    ...
}
```

```
public class ContentGUIPaneModel
{...
    public void handleAppEvent(Sample.controller.AppEvent ae)
    {
        String[] stra = new String[1];
        // stra = ServerFacade.getData(xyz...);
        stra[0] = "Hello World!";
        contentPane_.refreshView(stra);
    } ...
}
```

HMVC - implementace

- **Spodní vrstva - pokračování**

```
public class ContentGUIPane
{    ...
     public void refreshView(String[] data)
     {
          txtMsg_.setText(data[0]);
          validate();
     }...
}
```

Střední vrstva

```
public class GUIFrameController ...
{
    ...
    public void handleAppEvent(AppEvent ae)
    {
        if(ae.getMessage().equals("SHOW_HELLO_WORLD"))
        {
            createChildTriad();
            childCtlr_.init();
        }
    }
    public void createChildTriad()
    {
        ContentGUIPane contentPane = new ContentGUIPane(new Dimension(20,20));
        ContentGUIPaneModel contentModel = new ContentGUIPaneModel();
        ContentGUIPaneController contentController = new ContentGUIPaneController();
        contentController.setView(contentPane);
        contentController.setModel(contentModel);
        contentController.setParentController(this);
        setChildController(contentController);
        view_.setPane(contentPane);
    }...
}
```

HMVC – použití

- **Aplikace**
  - Webove stranky s widgety
- **Knihovny**
  - http://www.thecentric.com/resources/java-gnome-hmvc.tar.gz

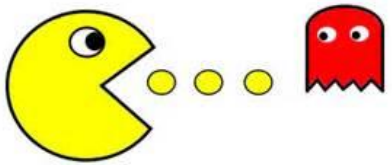
HMVC – shrnutí

- **Vlastnosti**
  - Vyšší coupling
  - Přístup k datům skze Controller z jakýkoli vrstvy
  - Specificky vzor pro UI s widgety
  - Není ideální řešení pro tenké klienty

- **Související vzory**
  - MVC, PAC, MVVM, MVP

PAC – Úvod

- **Problém**
  - Máme-li množství podproblémů / agentů:
    - s hierarchickým uspořádáním
    - se vzájemnou komunikací
    - ale jinak na sobě takřka nezávislí



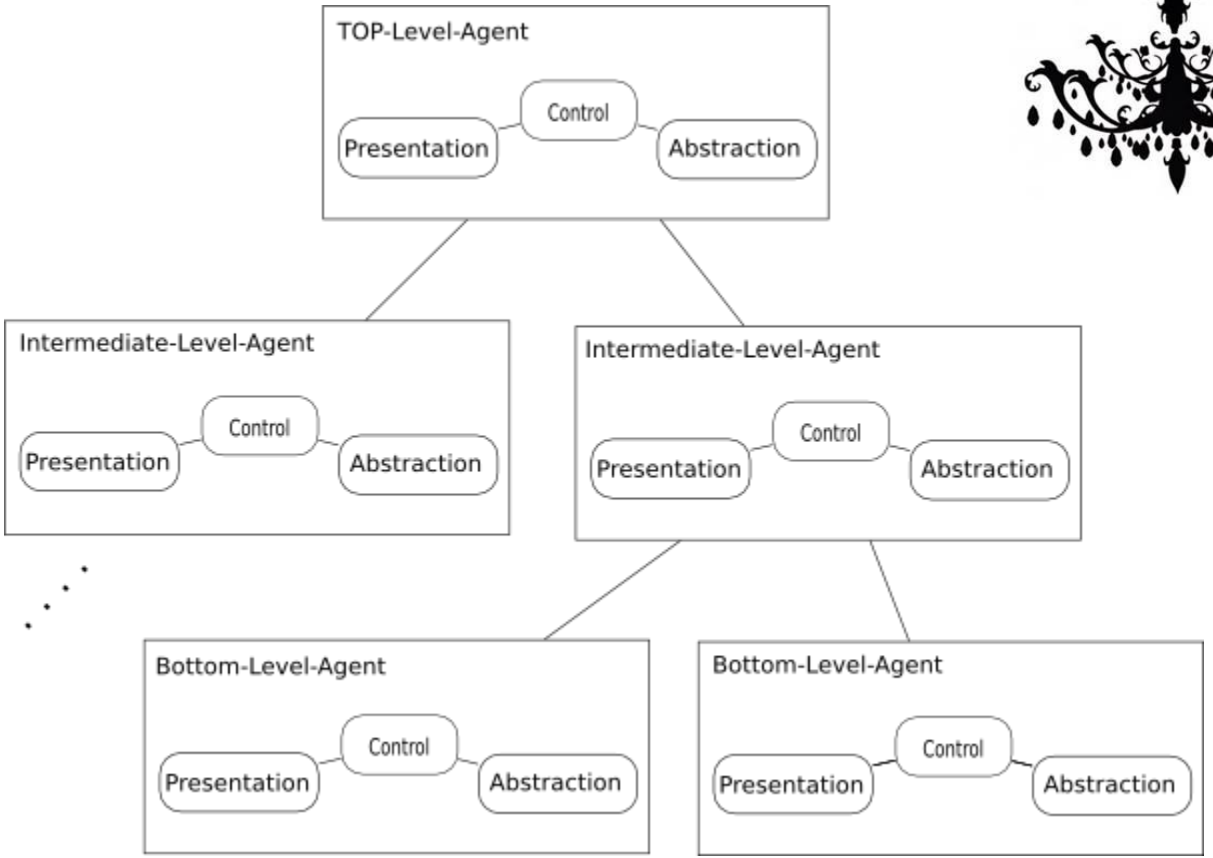
- **Příklad:**
  - Rozwidgetování
  - Air traffic control
  - Dynamické spreadsheety

- **Základ architektury**
  - Rozdělení agentů do 3+ vrstvého stromu
  - Každý agent = triáda P + A + C

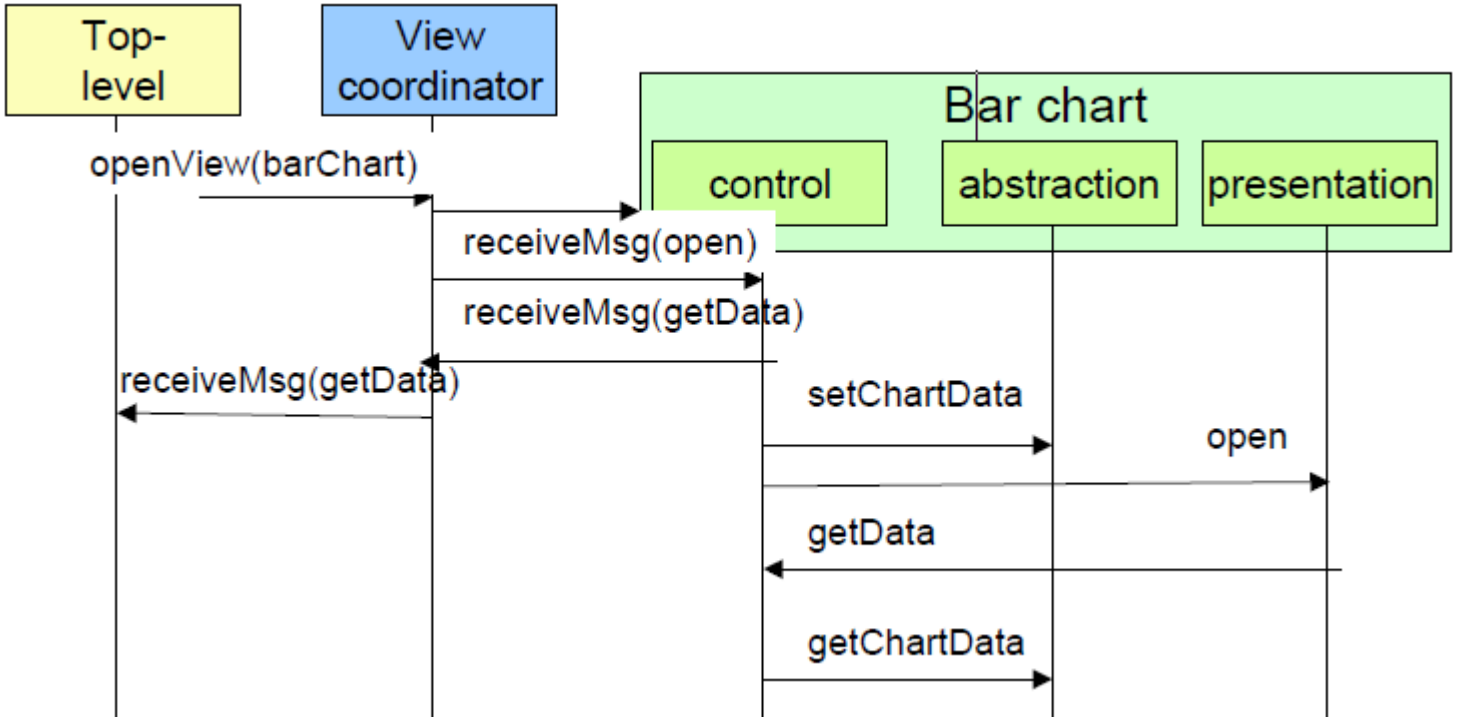
PAC – Triáda

- **Presentation**
  - Stará se o prezentační vrstvu
  - Např. šablona pro převod dat do HTML
- **Abstraction**
  - Poskytuje přístup k datům a umožňuje jejich transformaci
- **Control**
  - Obsahuje veškerou řídící logiku
  - Zprostředkovává komunikaci mezi:
    - Prezentací
    - Abstrakcí
    - a maminkou

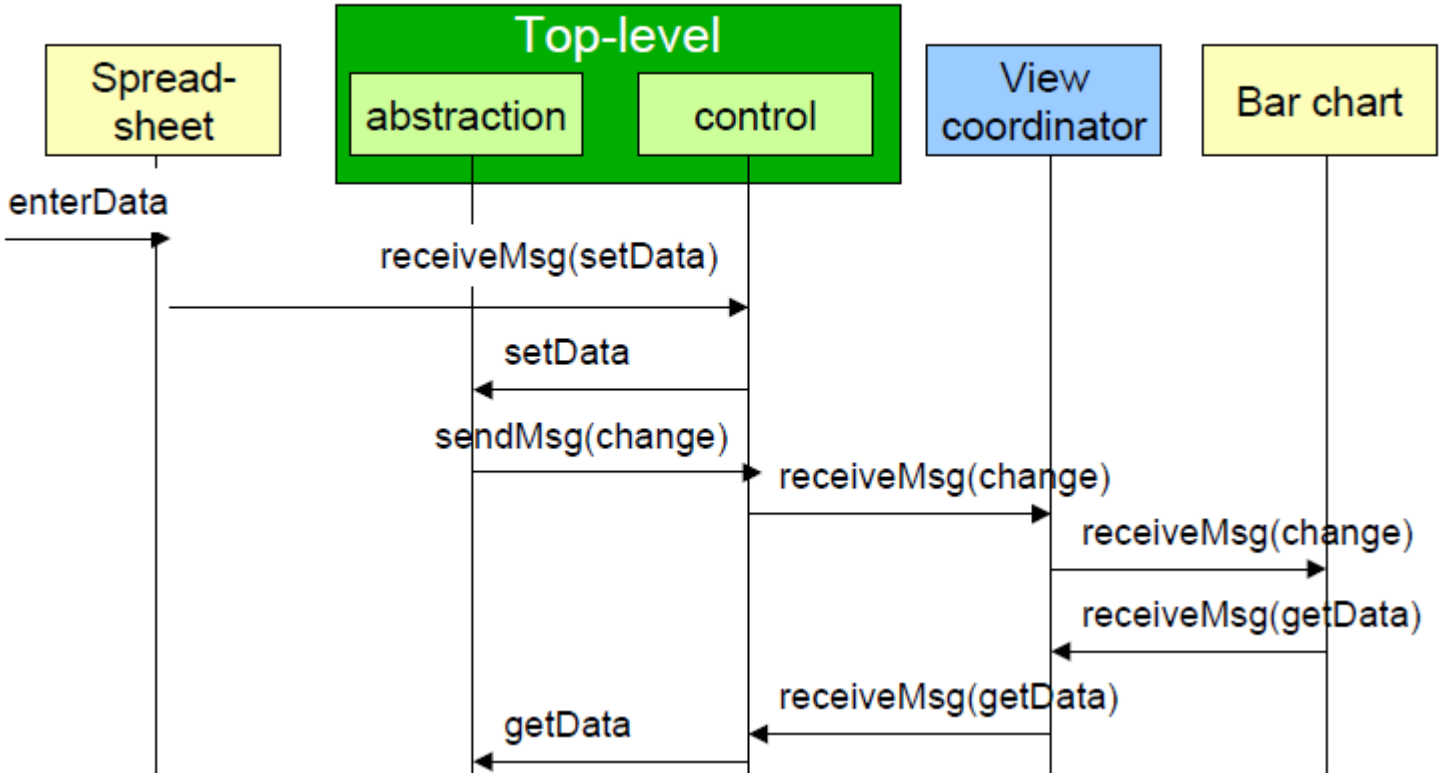
PAC – ilustrace



PAC – Příklad 1 - Tvorba widgetu



PAC – Příklad 2 - Změna dat



PAC – Použití

- **Aplikace**
  - Drupal
    https://www.drupal.org/

  - GIMP
    http://www.gimp.org/

- **Knihovny**
  - Java PAC Framework (?)
    http://jpac.sourceforge.net/
  - SporeJS Framework
    https://github.com/springload/SporeJS

PAC – Shrnutí

- **+++**
  - Podpora pro multi-tasking a multi-viewing
  - Reusability a extensibility
  - Oddělení zájmových oblastí

- **!!!**
  - Obtížnější implementace - těžké určit správný počet agentů
  - Složité kontrolery
  - Vyšší výpočetní náročnost
  - Relativně málo zdrojů

- **Související vzory**
  - HMVC, MVC, MVP...
  - Chain of Responsibility, Mediator, Observer

HMVC vs. PAC

- Publikováno 2000 (JavaWorld)
- Model, View, Controller
- Vyšší coupling
- Přístup k datům skze Controller z jakýkoli vrstvy
- Více specifický vzor - poddruh PAC vzoru můžeme říct, určený hlavně na UI

- Publikováno 1987
- Presentation, Abstraction, Control
- Menší coupling - vše jde přes mediatora Control
- Přístup k datům pouze přes Control z Top-Level-Agenta
- Více obecný vzor použitelný na širokou škálu interaktivního softwaru