Projet Informatique

2019

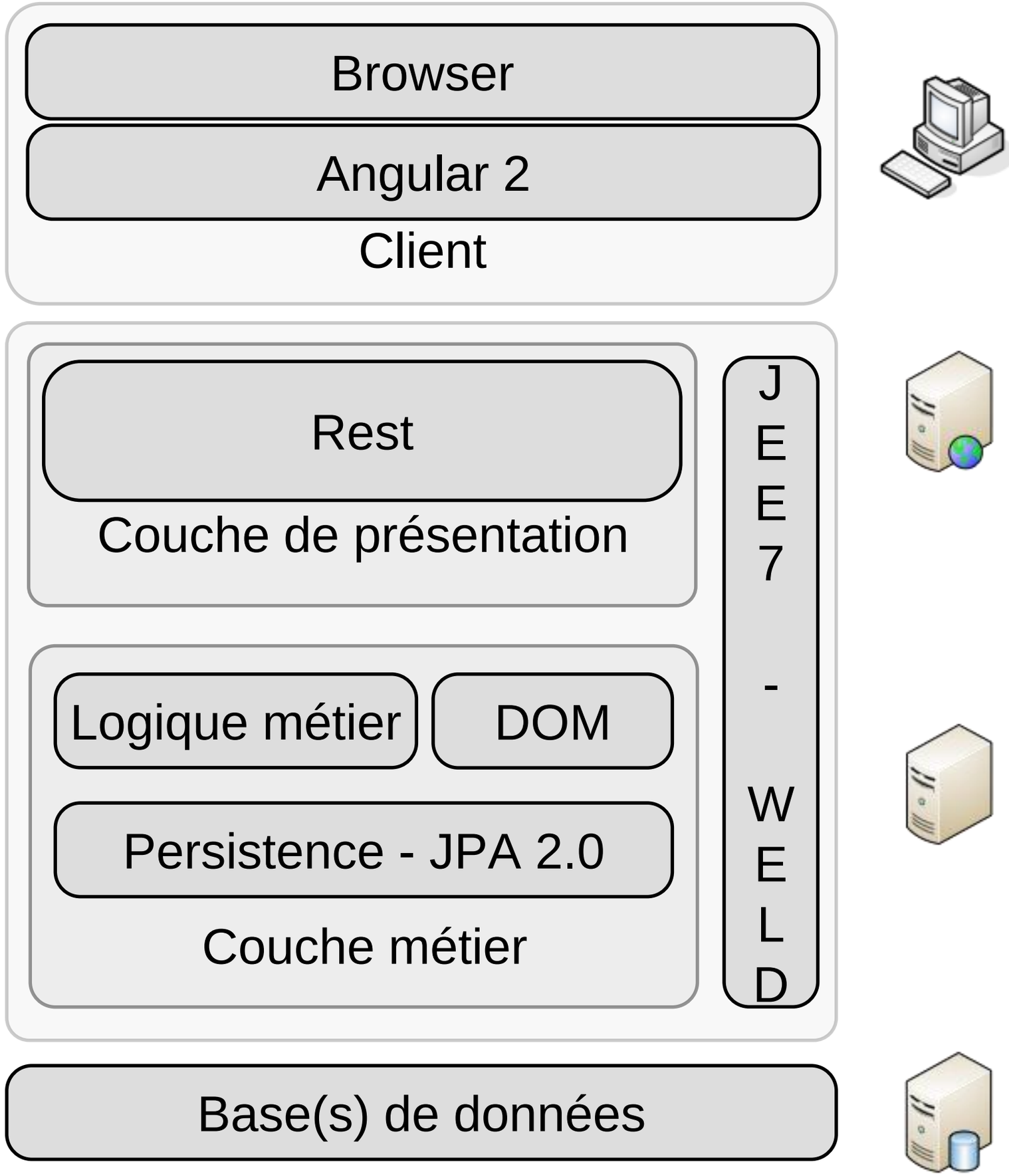
Steve Hostettler

Software Modeling and Verification Group

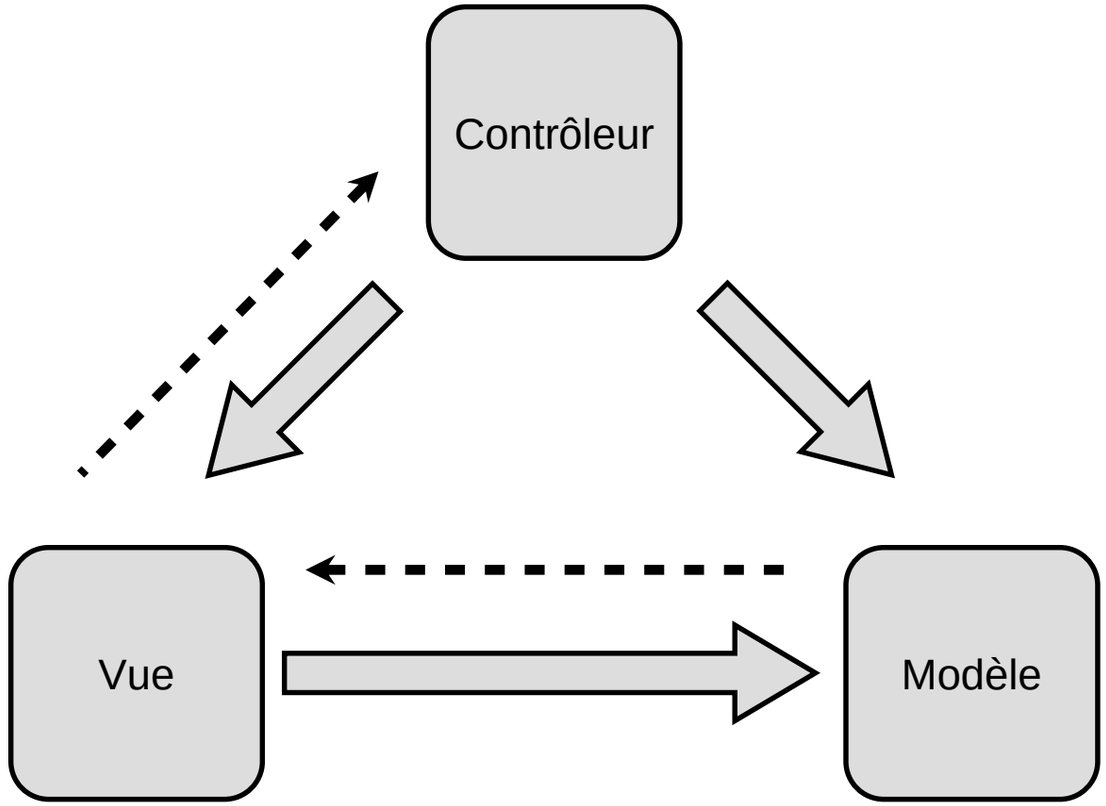
University of Geneva

Résumé des épisodes précédents

- Base de la programmation Java Web : JSP + Servlet
  - Architecture très simple
  - Portable (pas de vendor-lock)
- Principaux problèmes
  - Risque de mélanger les objectifs (separation of concerns)
  - Requête à la base de données dans les Java Server Pages
  - Tout est manuel (enchainement des pages, interprétations des formulaires, pas de composants réutilisables, sécurité)
    - Pas “user-friendly”, ni “programmer-friendly”
    - Beaucoup de temps perdu sur la partie opérationnelle vs la partie business.


Browser
Angular 2
Client
Rest
Couche de présentation
Logique métier
DOM
Persistence - JPA 2.0
Couche métier
J
E
E
7
-
W
E
L
D
Base(s) de données

Modèle-Vue-Contrôleur



Bean Properties

```
import java.io.Serializable;
import java.util.Date;

public class Student implements Serializable {
   /** The serial-id. */
   private static final long serialVersionUID = -6146935825517747043L;

   /** The student last name. */
   private String mLastName;

   public Student() {  }

   public final String getLastName() {
      return mLastName;
   }

   public final void setLastName(final String lastName) {
      mLastName = lastName;
   }
}
```

Constructeur par défault obligatoire

Bean Properties

```
import java.io.Serializable;
import java.util.Date;

public class Student implements Serializable {
   /** The serial-id. */
   private static final long serialVersionUID = -6146935825517747043L;

   /** The student last name. */
   private String mLastName;

   public Student() {  }

   public final String getLastName() {
      return mLastName;
   }

   public final void setLastName(final String lastName) {
      mLastName = lastName;
   }
}
```

La serialization est recommandée

Bean Properties

```
import java.io.Serializable;
import java.util.Date;

public class Student implements Serializable {
   /** The serial-id. */
   private static final long serialVersionUID = -6146935825517747043L;

   /** The student last name. */
   private String mLastName;

   public Student() {  }

   public final String getLastName() {
      return mLastName;
   }

   public final void setLastName(final String lastName) {
      mLastName = lastName;
   }
}
```

Getter = get + Nom de la propriété avec la première lettre en majuscule

Bean Properties

```
import java.io.Serializable;
import java.util.Date;

public class Student implements Serializable {
   /** The serial-id. */
   private static final long serialVersionUID = -6146935825517747043L;

   /** The student last name. */
   private String mLastName;

   public Student() {  }

   public final String getLastName() {
      return mLastName;
   }

   public final void setLastName(final String lastName) {
      mLastName = lastName;
   }
}
```

Setter = set + Nom de la propriété avec la première lettre en majuscule

Bean Properties

Les conventions de nommage des “beans” permet à une technologie appelée Reflection de retrouver une méthode à partir de son nom.

Reflection

C’est la possibilité d’interroger et de manipuler dynamiquement les elements d’une application écrite en JAVA.

Reflection

Débogueurs

GUI

Cross-language calls

Dynamic programming

Diminuer le couplage

Reflection

```
/**
 * Prints out the name and the type of the public fields.
 * @param o the object to scan
 */
void printFieldNames(final Object o) {
    Class<?> c = o.getClass();
    Field[] publicFields = c.getFields();
    for (int i = 0; i < publicFields.length; i++) {
        String fieldName = publicFields[i].getName();
        Class<?> typeClass = publicFields[i].getType();
        String fieldType = typeClass.getName();
        System.out.println("Name: " + fieldName + ", Type: " + fieldType);
    }
}
```

Reflection

```
/**
 * Prints out the name, the parameters and the return type of the public methods.
 * @param o  the object to scan
 */
void showMethods(final Object o) {
    Class<?> c = o.getClass();
    Method[] meths = c.getMethods();
    for (int i = 0; i < meths.length; i++) {
        String methName = meths[i].getName();
        System.out.println("Name: " + methName);
        String retName = meths[i].getReturnType().getName();
        System.out.println(" Return Type: " + retName);
        Class<?>[] paramTypes = meths[i].getParameterTypes();
        System.out.print(" Parameter Types:");
        for (int k = 0; k < paramTypes.length; k++) {
            String paramName = paramTypes[k].getName();
            System.out.print(" " + paramName);
        }
        System.out.println();
    }
}
```

Reflection

```
/**
 * Prints out the name, the parameters and the return type of the public methods.
 * @param o  the object to scan
 */
void showMethods(final Object o) {
    Class<?> c = o.getClass();
    Method[] meths = c.getMethods();
    for (int i = 0; i < meths.length; i++) {
        String methName = meths[i].getName();
        System.out.println("Name: " + methName);
        String retName = meths[i].getReturnType().getName();
        System.out.println(" Return Type: " + retName);
        Class<?>[] paramTypes = meths[i].getParameterTypes();
        System.out.print(" Parameter Types:");
        for (int k = 0; k < paramTypes.length; k++) {
            String paramName = paramTypes[k].getName();
            System.out.print(" " + paramName);
        }
    }
}
```

Reflection

```
//Concats Hello to World using the reflection API
Class<?> c = String.class;
Class<?>[] paramTypes = new Class[] { String.class };
String result = null;
try {
    Method method = c.getMethod("concat", paramTypes);
    Object[] args = new Object[] { "World!" };
    result = (String) method.invoke("Hello", args);
} catch (NoSuchMethodException e) {
    e.printStackTrace();
} catch (IllegalAccessException e) {
    e.printStackTrace();
} catch (InvocationTargetException e) {
    e.printStackTrace();
}
System.out.println(result);
```