Spring Framework
tvorba obchodních aplikací
v Javě bez J2EE

Lukáš Zapletal

Enterprise Application

- co je to obchodní aplikace?
  jedná se o softwarovou aplikaci, kterou firma vyvinula, zakoupila nebo převzala, poskytující strategické služby dané firmě
- informační systémy (účetnictví, sklad, person.)
- manažerské systémy (datawarehouses)
- ERP – plánování výrobního procesu
- CRM – podpora zákazníků
- a jiné

Opakování – komponenta

- softwarová komponenta:
  - samostatná jednotka dodávaná v binární podobě
  - lze snadno vyměnit za jinou komponentu
  - musí být možné ji nasadit nezávisle jako produkt třetí strany
  - je vhodné mít komponenty jednoho typu (s jasně definovaným rozhraním a pravidly pro nasazení), aby byla usnadněna výměna

Enterprise JavaBeans

- je to specifikace vyvinutá firmou Sun
- součástí J2EE, kde je také
  - Servlet/JSP/JSF
  - JNDI
  - JTA
  - JMS
  - Web Services
  - jádro ale tvoří komponentová technologie EJB

EJB = Enterprise JavaBean

- softwarové komponenty napsané v jazyce Java a vyhovující standardizovaným rozhraním, která jsou součástí platformy J2EE
- komponenty, které se nasazují na aplikačním serveru (EJB kontejner)
- dostupné a instalovatelné přes síť a jejich úlohou je poskytovat nástroj pro tvorbu enterprise aplikací
- příklad: EJB pro odeslání e-mailu

Proč EJB vznikly?

- existuje mnoho jiných řešení, například CORBA
- Sun chtěl čistou Javu a možnost využít plný potenciál
- Sun chtěl mít také vývoj více pod kontrolou
- J2SE již obsahuje plnou podporu pro CORBA API
- ***Write once, run everywhere***

Vícevrstvé aplikace

- obchodní aplikace se tvoří jako vícevrstvé
- vrstva aplikační logiky (business logic) je tvořena EJB komponentami
- softwarové komponenty se dělí podle účelu do několika kategorií, a proto se zavedlo několik EJB komponent:
  - business objekt (doménový objekt)
  - aplikační kontroler (vlastní akce s dom. objekty)
  - ostatní

Tři typy EJB komponent

- entity bean
  - business objekt („databázový“ objekt)
  - obvykle představuje řádek v relační databázi
  - 2 kategorie: CMP a BMP
- session bean
  - vlastní akce, kontejnery jsou schopny transakčního zpracování, 2 kategorie: stateful a stateless
- message-driven bean
  - pro zasílání zpráv

Spring Framework

- alternativa k J2EE a EJB
- Ron Johnson, autor knihy
  Expert One-on-One J2EE Design and Development
- zabývá se problémy J2EE
- své poznámky a komentáře zpracoval do podoby open-source knihovny Spring

Spring Framework - Motivace

- v J2EE snadné věci nelze dělat jednoduše
- J2EE má mnoho závislostí, bez kterých nefunguje (má monolitický návrh)
- aplikace jsou přímo závislé na J2EE
- nedokonalé mapování u Entity EJB
- J2EE neklade důraz na best practices
- J2EE se těžko rozšiřuje
- J2EE není open-source software (licence, dokumentace a podobně)

Dependency Injection

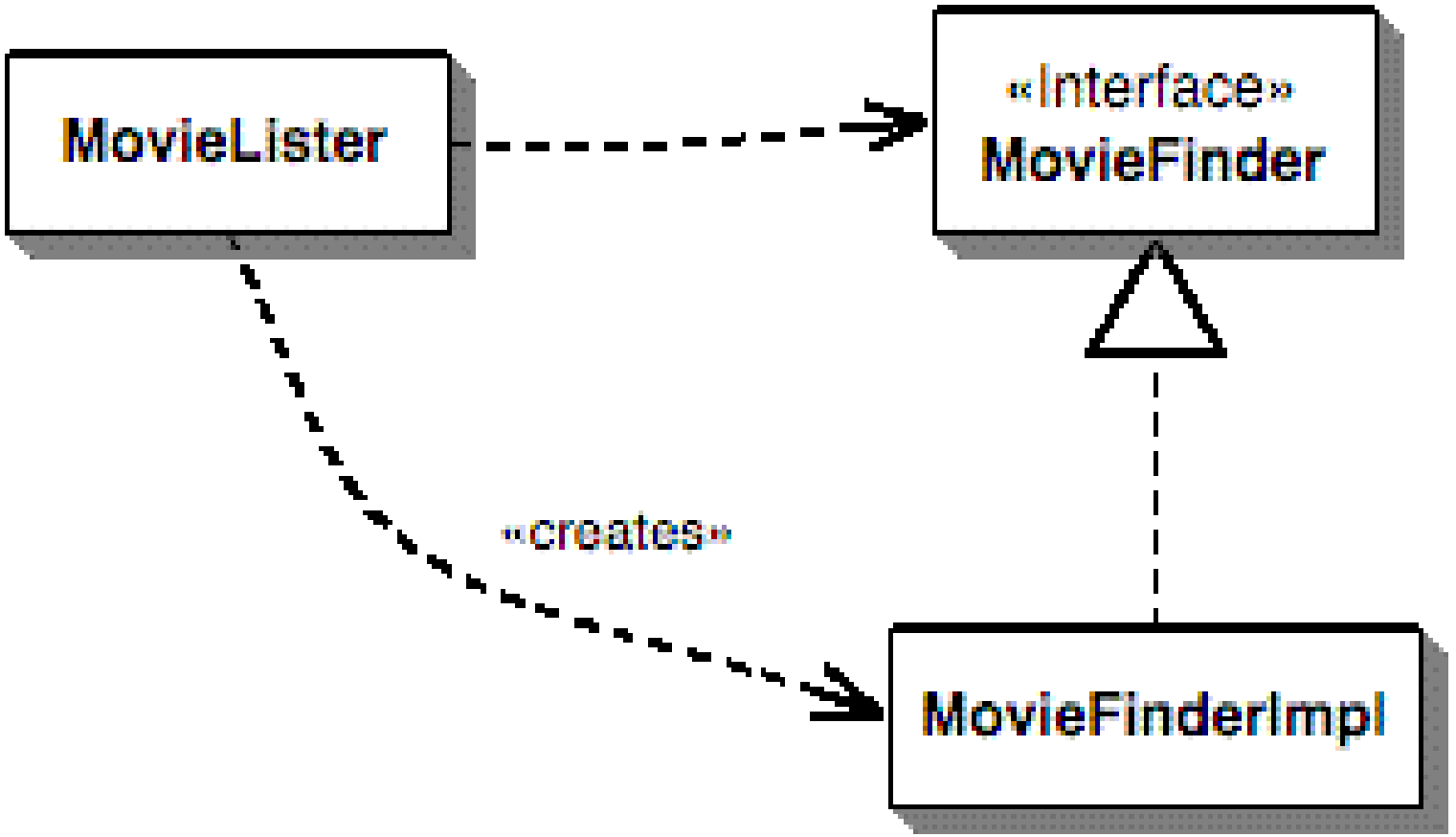
- Spring jako komponentová technologie je postavena na myšlence Martina Flowera:

  Inversion Of Control

  což později přejmenoval na

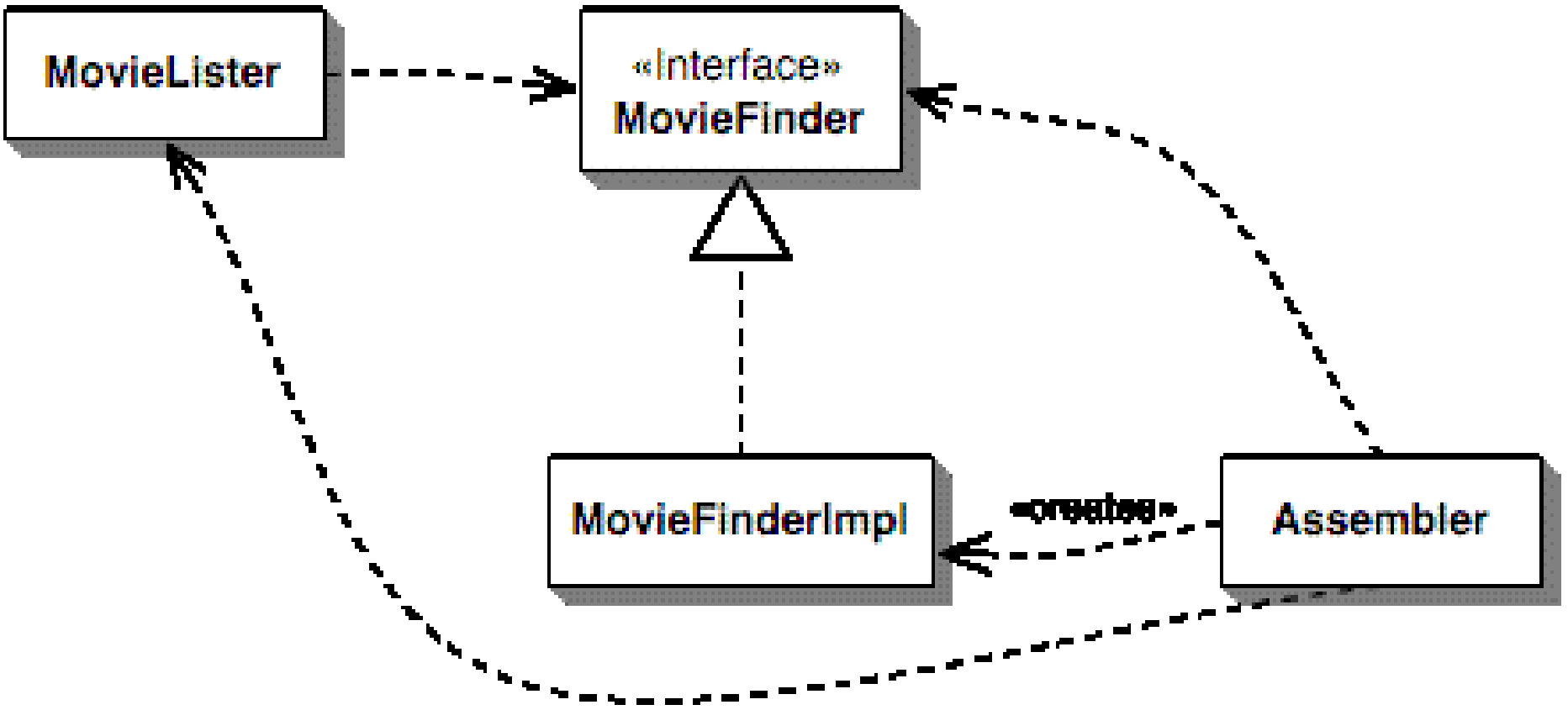
  Dependency Injection
  (vpíchnutí závislostí)

J2EE = Service Locator



- MovieLister **vyhledá** implementaci MovieFinder

Vpíchnutí závislostí



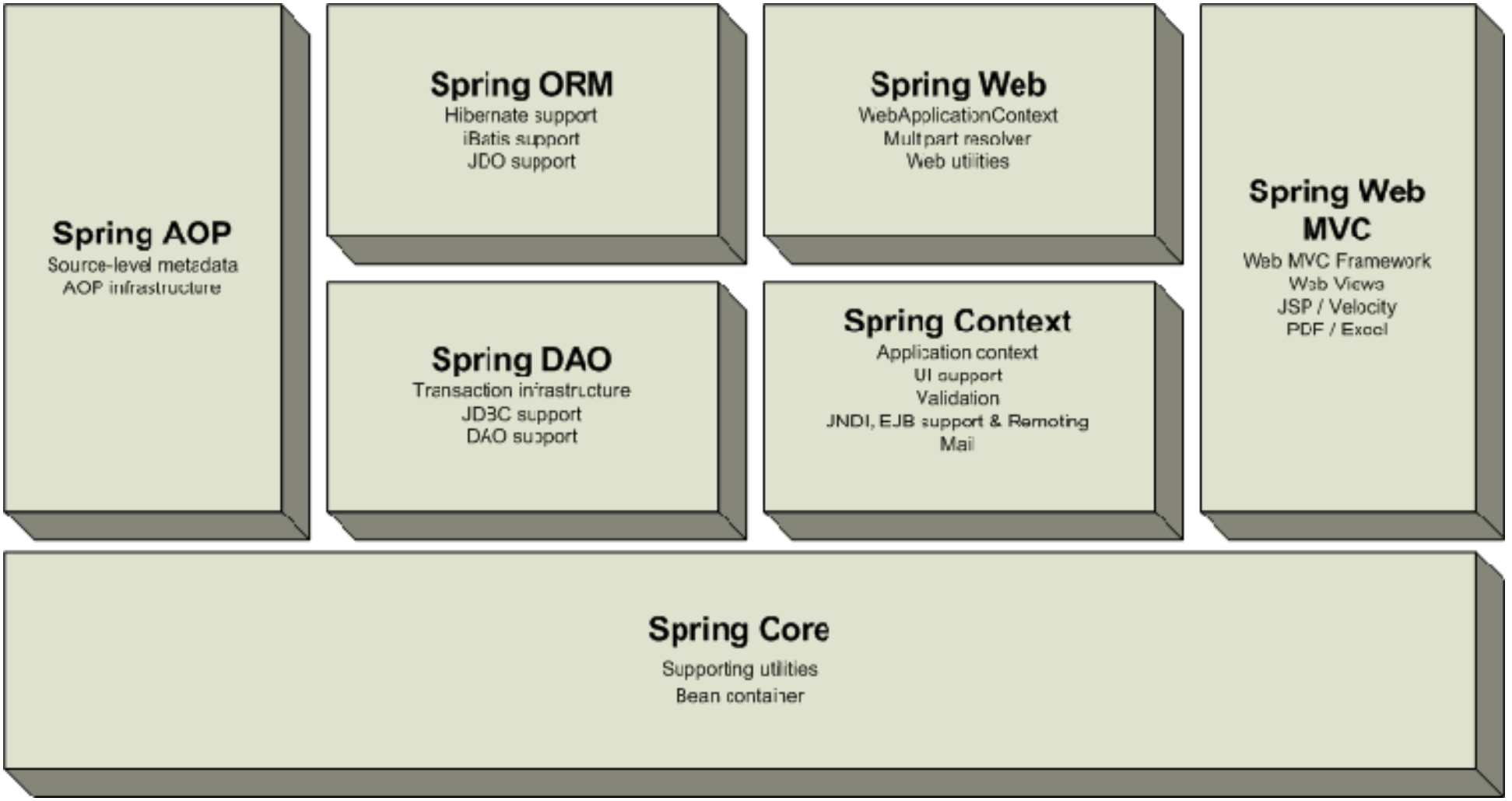
- o sestavení se stará assembler
- komponenty o sobě nevědí – nejsou závislé

Typy vpíchnutí závislostí

- constructor injection (pomocí konstruktoru)
- setter injection (pomocí vlastnosti objektu)
- interface injection (pomocí rozhraní)

- tento přístup používají i jiné komponentové technologie, jako například Avalon (3+SL) nebo PicoContainer (1+2)

Spring Framework



Spring - komponenty

- pro assembler se používá XML konfigurace
- XML má promyšlenou strukturu
- máte možnost předávat veškeré možné hodnoty (čísla, řetězce), odkazy na jiné objekty či null hodnoty
- podpora kolekcí (List, Set, Map, Properties), singletonů, abstraktních tříd a abstraktních továren
- pokročilé techniky jako jsou runtime delegace a výměny metod (informace v dokumentaci)

Spring - komponenty

```
class examleBean {
  ...
  public void setDatabase(DatabaseBean db)
  ...
  public void setNumConnections(int n)
  ...
}

<bean id="exampleBean" class="examples.ExampleBean">
  <property name="database"><ref bean="db"/></property>
  <property name="numConnections">
    <value>32</value>
  </property>
</bean>

<bean id="db" class="examples.DatabaseBean"/>
```

Spring - komponenty

- komponenty v Beanu nemusejí povinně implementovat žádné metody (init, destroy a podobně)
- teprve až když je nutné explicitně kontrolovat životní cyklus komponenty, je možno implementovat definovaná rozhraní
- Spring umí tzv. Autowiring mode

Spring Context

- napojení na EJB
- validace:

```
public class PersonValidator implements Validator {

	public boolean supports(Class clzz) {
		return Person.class.equals(clzz);
	}

	public void validate(Object obj, Errors e) {
		ValidationUtils.rejectIfEmpty(e, "name", "name.empty");
		Person p = (Person)obj;
		if (p.getAge() < 0) {
			e.rejectValue("age", "negativevalue");
		} else if (p.getAge() > 110) {
			e.rejectValue("age", "tooold");
		}
	}
}
```

Transakční zpracování

- dostatečně zobecněno
- možno použít JDBC, JTA, Hibernate API
- jako vše je TX bean taky komponenta:

```
<bean id="txManager"
class="org.springframework.orm.hibernate.HibernateTransactionManager">
  <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

Source-level metadata

- až od Javy verze 5.0
- to je ale pouze statické
- Spring toto umožňuje díky XDoclet knihovně
- podporuje abstraktní API (oba přístupy)

```
/**
 * @@org.springframework.aop.framework.autoproxy.target.PoolingAttribute(10)
 * @author Rod Johnson
 */
public class MyClass {
```

DAO a O/R mapping

- DAO ve Springu slučuje výjimky
- uložení POJO objektů je možné buď přímo JDBC, nebo pomocí jednoho z mapperů:
  - Hibernate
  - Apache OJB
  - JDO
  - iBatis
  - Oracle TopLink

DAO a O/R mapping - Hibernate

```
<beans>
  <bean id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
    <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
    <property name="url" value="jdbc:hsqldb:hsql://localhost:9001"/>
    <property name="username" value="sa"/>
    <property name="password" value=""/>
  </bean>

  <bean id="mySessionFactory" class="org.springframework.orm.hibernate.LocalSessionFactoryBean">
    <property name="dataSource" ref="myDataSource/>
    <property name="mappingResources">
      <list>
        <value>product.hbm.xml</value>
      </list>
    </property>
    <property name="hibernateProperties">
      <props>
        <prop key="hibernate.dialect">net.sf.hibernate.dialect.OracleSQLDialect</prop>
      </props>
    </property>
  </bean>
</beans>
```

DAO a O/R mapping - Hibernate

```
public class ProductDaoImpl implements ProductDao {

    private SessionFactory sessionFactory;

    public void setSessionFactory(SessionFactory sf) {
        this.sessionFactory = sf;
    }

    public Collection loadProductsByCategory(String category) {
        return this.sessionFactory.getCurrentSession()
                .createQuery("from test.Product product“ +
                  "where product.category=?")
                .setParameter(0, category)
                .list();
    }
}
```

DAO a O/R mapping - Hibernate

```
public class ProductServiceImpl implements ProductService {

    private PlatformTransactionManager tManager;
    private ProductDao productDao;

    public void setTransactionManager(PlatformTransactionManager transactionManager) {
        this.tManager = transactionManager;
    }

    public void setProductDao(ProductDao productDao) {
        this.productDao = productDao;
    }

    public void increasePriceOfAllProductsInCategory(final String category) {
        TransactionTemplate transactionTemplate = new TransactionTemplate(this.tManager);
        transactionTemplate.execute(
            new TransactionCallbackWithoutResult() {
                public void doInTransactionWithoutResult(TransactionStatus status) {
                    List productsToChange = productDAO.loadProductsByCategory(category);
                    ...
                }
            }
        );
    }
}
```

Spring Web

- implementuje známé best practices
- můžeme se rozhodnout:
  - Spring MVC + libovolná view technologie (JSP, Velocity/Freemaker, XSLT, PDF, Jasper)
  - nezávislý web framework (JSF, Struts, Tapestry, WebWork)

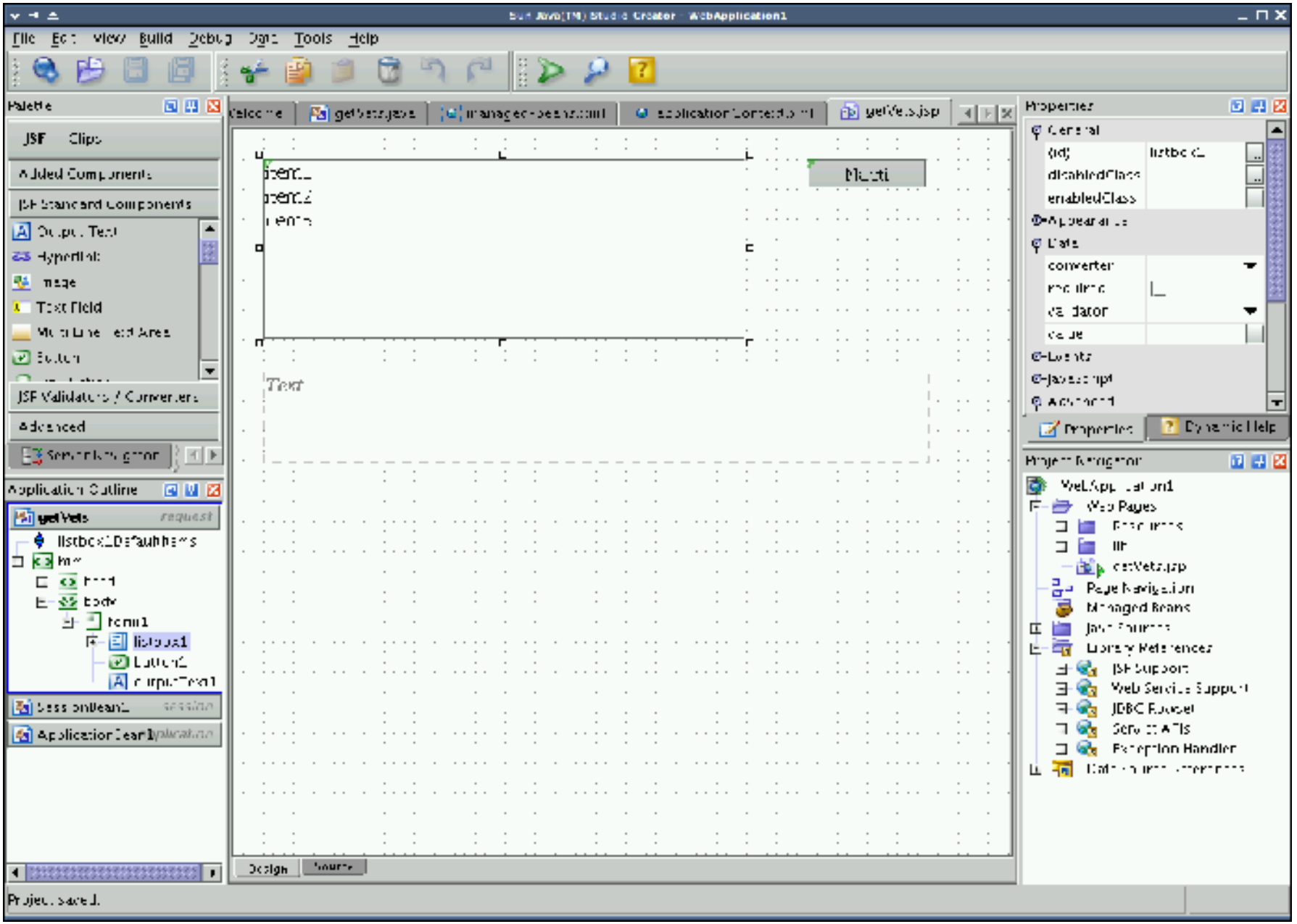
Spring MVC

- MODEL – uchovává informace (bean)
- VIEW – zobrazuje výsledek
- CONTOLLER – provádí vlastní akci
- Spring MVC je úzce svázán se Springem (žádné form-beany) a je rozšiřitelný
- Spring přidává další doplňky:
  - nezávislé mapování
  - locales, themes
  - exceptions

Java Server Faces

- mladá technologie
- podobnost s ASP.NET 2.0
- vizuální programování
- vyvíjela se několik let, finální verze teprve nedávno
- Spring s ní spolupracuje

Java Server Faces



Další technologie

- webové služby (RMI, JAX-RPC)
- možnost vytvářet i používat EJB komponenty
- JMS
- Spring email infrastructure
- JCA CCI (Java Connector Architecture – Common Client Interface)

Odkazy, reference

- Rod Jonhson: Expert One-on-One J2EE Design and Development, Wiley Publ. 2003
- Martin Flower: Inversion of Control Containers and the Dependency Injection pattern, http://www.martinfowler.com/articles/injection.html
- Sun Microsystems, Core J2EE Patterns, Sun Press, http://java.sun.com/blueprints/corej2eepatterns/Patterns/ServiceLocator.html
- Das Spring Framework als Teil eines Paradigmenwechsels? Vergleich der leichtgewichtigen Alternative zur traditionellen J2EE Entwicklung, diplomová práce. http://www.martinmaier.name/archives/5
- Sum Microsystems, J2EE Portal, oficiální dokumentace technologie J2EE, http://java.sun.com/j2ee/