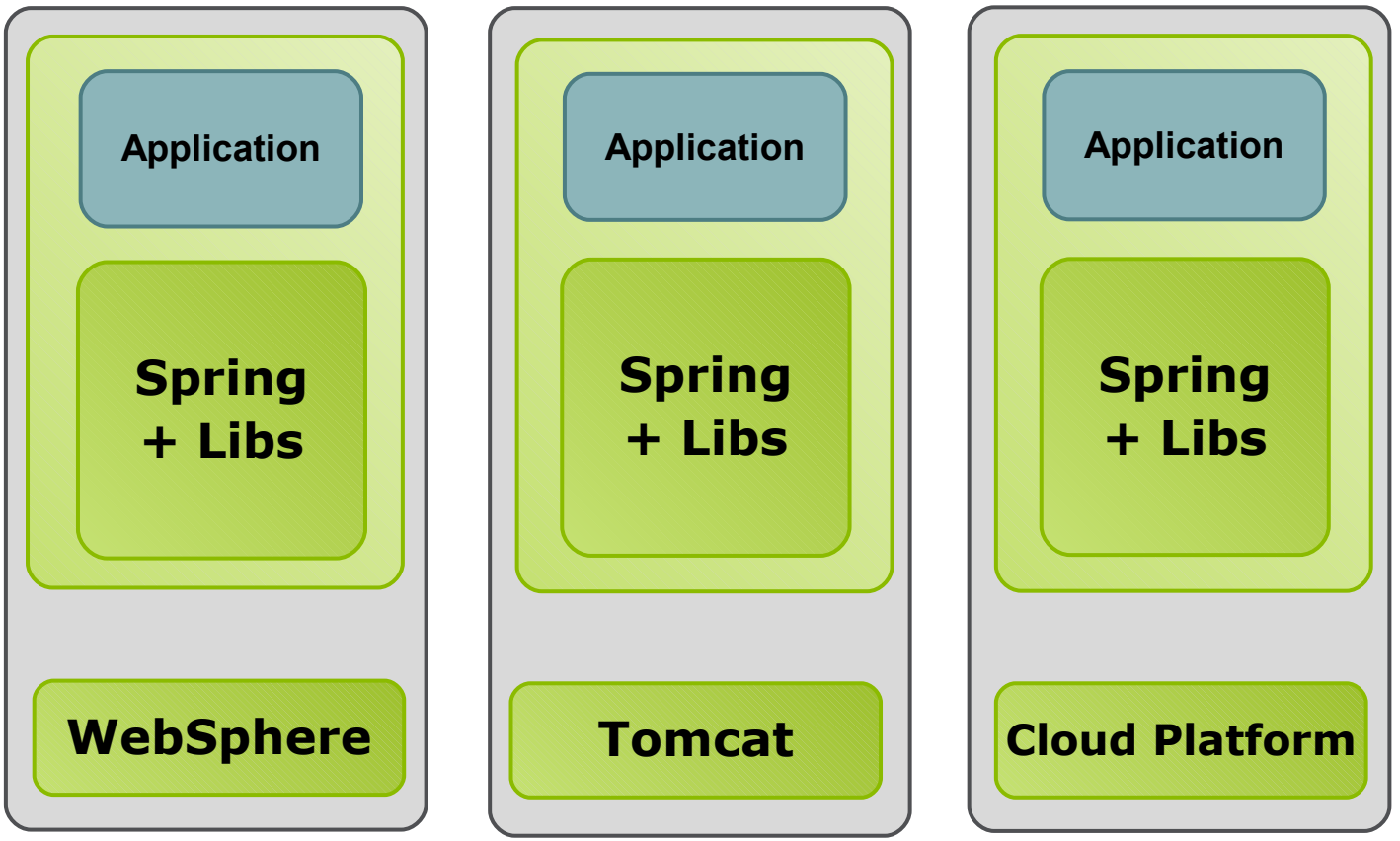
Spring 3.1 and Beyond – Themes and Trends

Jürgen Höller, Principal Engineer, SpringSource

springsource
A division of vmware

CONFIDENTIAL

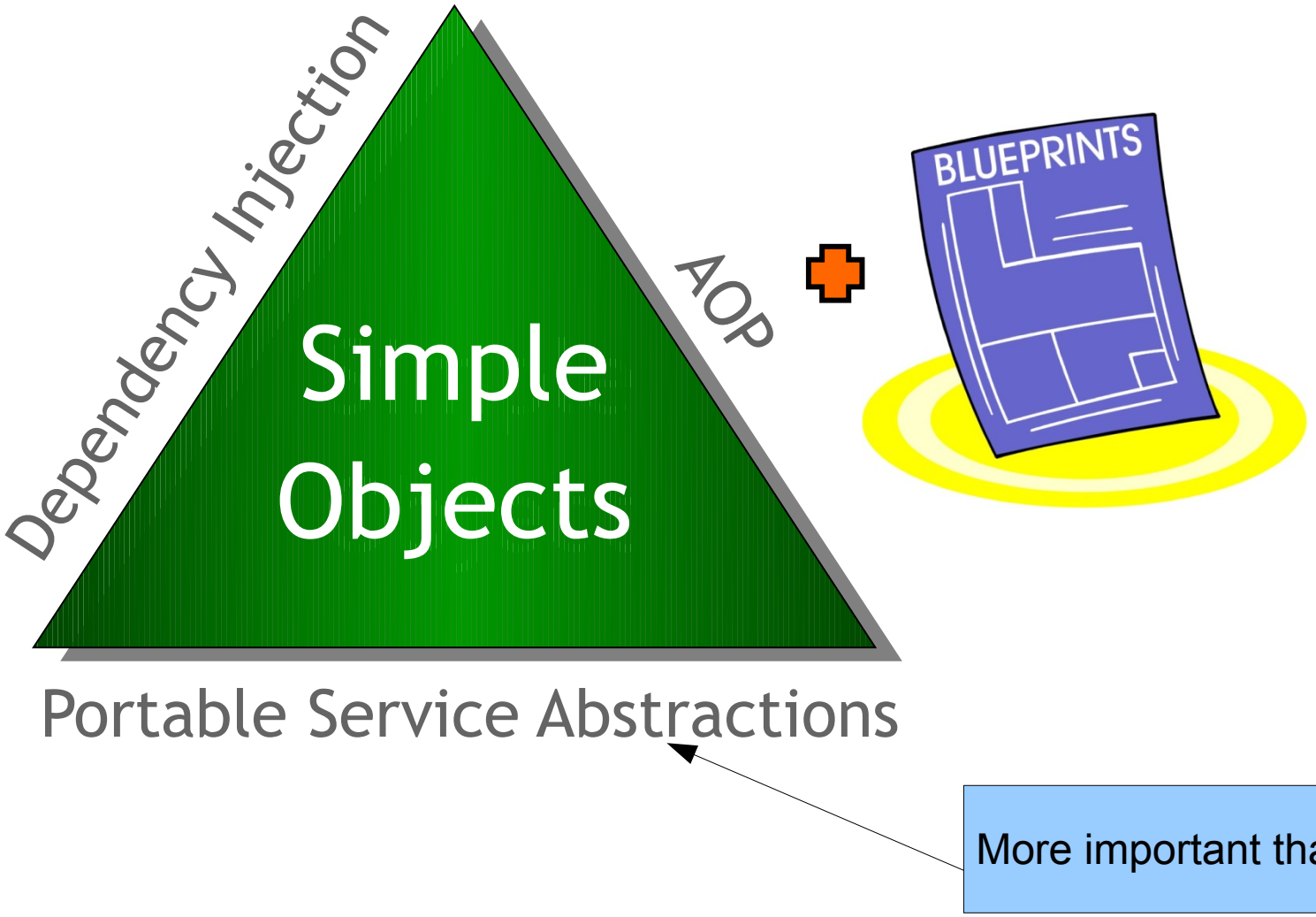
Deployment Platforms: Becoming More Diverse



Deployment Platforms in 2011: Latest Releases

- **Java EE moving on to Java EE 6**
  - GlassFish 3
  - JBoss 7
  - WebSphere 8
- **Tomcat moving on to Tomcat 7**
  - Servlet 3.0 based (Java EE 6 level)
- **Cloud platforms becoming a serious option for regular Java web application deployment**
  - Google App Engine: Jetty++
  - Amazon Elastic Beanstalk: Tomcat++
  - VMware's CloudFoundry: Tomcat++

Key Elements of Spring: Ready for 2011 & Beyond



A Quick Review: Spring Framework 3.0

- **Powerful annotated component model**
  - stereotypes, factory methods, JSR-330 support
- **Spring Expression Language**
  - Unified EL++
- **Comprehensive REST support**
  - and other Spring @MVC additions
- **Support for Portlet 2.0**
  - action/event/resource request mappings
- **Declarative model validation**
  - integration with JSR-303 Bean Validation
- **Support for Java EE 6**
  - in particular for JPA 2.0

Spring Framework 3.1: Key Themes

- Environment profiles for bean definitions
- Java-based application configuration
- Overhaul of the test context framework
- „c:“ namespace
- Cache abstraction & declarative caching
- Explicit support for Servlet 3.0
- @MVC processing & flash attributes
- Refined JPA support
- Hibernate 4.0 & Quartz 2.0
- Support for Java SE 7

Environment Abstraction

- **Grouping bean definitions for activation in specific environments**
  - e.g. development, testing, production
  - possibly different deployment environments
- **Custom resolution of placeholders**
  - dependent on the actual environment
  - hierarchy of property sources

- Injectable environment abstraction API
  - org.springframework.core.env.Environment
- Property resolution SPI
  - org.springframework.core.env.PropertyResolver

Environment Profiles

```
<beans profile="production">
  <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
    <property name="driverClass" value="${database.driver}"/>
    <property name="jdbcUrl" value="${database.url}"/>
    <property name="username" value="${database.username}"/>
    <property name="password" value="${database.password}"/>
  </bean>
</beans>

<beans profile="embedded">
  <jdbc:embedded-database id="dataSource" type="H2">
    <jdbc:script location="/WEB-INF/database/schema-member.sql"/>
    <jdbc:script location="/WEB-INF/database/schema-activity.sql"/>
    <jdbc:script location="/WEB-INF/database/schema-event.sql"/>
    <jdbc:script location="/WEB-INF/database/data.sql"/>
  </jdbc:embedded-database>
</beans>
```

Environment Configuration

- **Environment association of specific bean definitions**
  - **XML 'profile' attribute** on <beans> element
  - **@Profile annotation** on configuration classes
  - **@Profile annotation** on individual component classes
- **Activating specific profiles by name**
  - e.g. through a system property
    - -Dspring.profiles.active=development
  - or other means outside of the deployment unit
    - according to environment conventions
- **Ideally: no need to touch deployment unit across different stages/environments**

Java-Based Application Configuration

- **Application-specific container configuration**
  - aligned with the @Configuration style
  - focus on customizing the annotation-based processing parts of Spring
- **Equivalent to XML namespace functionality**
  - but not a one-on-one mapping
  - 'natural' container configuration from an annotation-oriented perspective
- **Typical infrastructure setup**
  - transactions
  - scheduling
  - MVC customization

Application Configuration Example

```
@Configuration
@EnableTransactionManagement(proxyTargetClass=true)
public class DataConfig {

        @Bean
        public PlatformTransactionManager txManager() {
                return new HibernateTransactionManager(sessionFactory());
        }

        @Bean
        public SessionFactory sessionFactory() throws Exception {
                return new AnnotationSessionFactoryBuilder(dataSource())
                        .addAnnotatedClasses(Order.class, Account.class)
                        .buildSessionFactory();
        }

        @Bean
        public DataSource dataSource() {
                // ... configure and return JDBC DataSource ...
        }
}
```

```
<tx:annotation-driven transaction-manager="txManager"
                      proxy-target-class="true"/>
```

Test Context Framework

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(
    loader=AnnotationConfigContextLoader.class,
    classes={TransferServiceConfig.class, DataConfig.class})
@ActiveProfiles("dev")
public class TransferServiceTest {

    @Autowired
    private TransferService transferService;

    @Test
    public void testTransferService() {
            ...
    }
}
```

"c:" Namespace for XML Configuration

- **New XML namespace for use with bean configuration**
  - shortcut for <constructor-arg>
    - inline argument values
    - analogous to existing "p:" namespace
  - use of constructor argument names
    - recommended for readability
    - debug symbols have to be available in the application's class files

```
<bean class="…" c:age="10" c:name="myName"/>

<bean class="…" c:name-ref="nameBean"
      c:spouse-ref="spouseBean"/>
```

Cache Abstraction

- **CacheManager and Cache abstraction**
  - in org.springframework.cache
    - which up until 3.0 just contained EhCache support
  - particularly important with the rise of distributed caching
    - not least of it all: in cloud environments

- **Backend adapters for EhCache, GemFire, Coherence, etc**
  - EhCache adapter to be shipped with Spring core
  - plugging in custom adapters if necessary

- **Specific cache setup per environment profile?**
  - potentially even adapting to a runtime-provided service

Declarative Caching

```
@Cacheable
public Owner loadOwner(int id);


@Cacheable(condition="name.length < 10")
public Owner loadOwner(String name);


@CacheEvict
public void deleteOwner(int id);
```

Cache Configuration

- **Cache namespace**
  - <cache:annotation-driven>
  - convenient setup for annotation-driven caching
  - pointing to a "cacheManager" bean by default

- **CacheManager SPI**
  - EhCacheCacheManager
    - backed by an EhCacheFactoryBean
  - GemFireCacheManager
    - part of the Spring GemFire project

Support for Servlet 3.0

- **Explicit support for Servlet 3.0 containers**
  - such as Tomcat 7 and GlassFish 3
  - while at the same time preserving compatibility with Servlet 2.4+
- **Support for XML-free web application setup (no web.xml)**
  - Servlet 3.0's ServletContainerInitializer in combination with Spring 3.1's AnnotationConfigWebApplicationContext plus the environment abstraction
- **Exposure of native Servlet 3.0 functionality in Spring MVC**
  - support for asynchronous request processing
  - standard Servlet 3.0 file upload support behind Spring's MultipartResolver abstraction

WebApplicationInitializer Example

```
/**
 * Automatically detected and invoked on startup by Spring's ServletContainerInitializer.
 * May register listeners, filters, servlets etc against the given Servlet 3.0 ServletContext.
 */
public class MyWebAppInitializer implements WebApplicationInitializer {

        public void onStartup(ServletContext sc) throws ServletException {
                // Create the 'root' Spring application context
                AnnotationConfigWebApplicationContext root = new AnnotationConfigWebApplicationContext();
                root.scan("com.mycompany.myapp");
                root.register(FurtherConfig.class);

                // Manages the lifecycle of the root application context
                sc.addListener(new ContextLoaderListener(root));

                // Enables support for DELETE and PUT request methods with web browser clients
                sc.addFilter("hiddenHttpMethodFilter", HiddenHttpMethodFilter.class)
                        .addMappingForUrlPatterns(null, false, "/*");

                ...
        }
}
```

@MVC Processing & Flash Attributes

- **RequestMethodHandlerAdapter**
  - arbitrary mappings to handler methods across multiple controllers
  - better customization of handler method arguments
    - HandlerMethodArgumentResolver
    - HandlerMethodReturnValueHandler
    - etc

- **FlashMap support and FlashMapManager abstraction**
  - with RedirectAttributes as a new @MVC handler method argument type
  - explicitly calling addFlashAttribute to add values to the output FlashMap
  - an outgoing FlashMap will temporarily get added to the user's session
  - an incoming FlashMap for a request will automatically get exposed to the model

Refined JPA Support

- **Package scanning without persistence.xml**
  - „packagesToScan“ feature on LocalContainerEntityManagerFactoryBean
  - similar to AnnotationSessionFactoryBean for Hibernate 3
  - ideal for applications with a single default persistence unit

- **Consistent JPA setup by persistence unit name**
  - JPA specification quite strongly based on the notion of persistence unit names
  - @PersistenceContext / @PersistenceUnit referring to persistence unit names
  - now Spring's JpaTransactionManager and co support setup by unit name as well
  - as an alternative to referring to EntityManagerFactory beans directly

Third-Party Support Updates

- **Hibernate 4.0**
  - natively and through JPA
  - dealing with package rearrangements in the Hibernate API
  - in a dedicated **org.springframework.orm.hibernate4 package**
  - preserving compatibility with Hibernate 3.2+ in orm.hibernate3

- **Quartz 2.0**
  - dealing with JobDetail and Trigger being interfaces in the Quartz 2.0 API
  - SchedulerFactoryBean now adapts to Quartz 2.0 if present
  - no support for JobDetailBean and Cron/SimpleTriggerBean anymore
  - preserving compatibility with Quartz 1.5+

Java SE 7

- **Spring 3.1 introduces Java SE 7 support**
  - making best use of JRE 7 at runtime
  - support for JDBC 4.1
  - support for fork-join framework
- **Oracle's OpenJDK 7 released in summer 2011**
  - IBM JDK 7 following quite soon after
  - chance of quite quick adoption
- **Spring Framework not being built against Java 7 yet**
  - build upgrade coming in Spring 3.2

Java SE 7: Concurrent Programming

- **A challenge: concurrent programming in a multi-core world**
  - user-level APIs and recommended programming styles?
- **Servers with more cores than concurrent requests**
  - how to actually use your processor power in such a scenario?
- **Java SE 7: java.util.concurrent.ForkJoinPool**
  - specialized ForkJoinPools to be locally embedded within the application
  - different kind of pool, separate from regular Runnable-oriented Executors
- **Ideal for Spring bean setup: ForkJoinPoolFactoryBean**

Spring Framework 3.1 Summary

- Selected improvements to the Spring 3.0 programming model

- 3.1 RC1 released a few weeks ago
  - **Environment profiles for bean definitions**
  - **Java-based application configuration**
  - **Overhaul of the test context framework**
  - **„c:“ namespace**
  - **Cache abstraction & declarative caching**
  - **Explicit support for Servlet 3.0**
  - **@MVC processing & flash attributes**
  - **Refined JPA support**
  - **Hibernate 4.0 & Quartz 2.0**
  - **Support for Java SE 7**

A Quick Preview: Spring Framework 3.2

- **Java EE 7 umbrella theme**
  - JCache support
  - JMS 2.0
  - JPA 2.1
  - Bean Validation 1.1
  - Servlet 3.0/3.1 asynchronous request processing
  - JSF 2.2

- **New framework build based on Gradle**
- **Revised container error reporting**
- **Javassist to replace CGLIB?**

Spring 3.2 Strategy

- **Early support for the latest Java specifications**
  - Java EE 7 as the central theme
  - as usual, support for selected specifications in individual form
  - with Java 8's language and API enhancements in mind already
- **Preserving compatibility with Java 5+**
  - Java SE 5+ as well as Java EE 5+
  - for the entire Spring 3.x branch
  - however, stronger focus on a Java SE 7 and Servlet 3.0+ world
- **Best possible experience on modern deployment environments**
  - from Tomcat 7 and WebSphere 8 to Google App Engine and Cloud Foundry