Domain-Driven Web-Development with Tapestry, HiveMind and Hibernate

tapestry

HiveMind

Marcus Schulte

marcus.schulte@s-i.ch

Overview

- Background: What were we trying to achieve?
- Why domain-driven?
- Architecture of the foundation frameworks
- Putting it all together – lifecycle of domain-entities
- Bottom-line: advantages and desiderata

Non-functional Requirements

- Extra-/Intranet Applications
- 10 to 1000 users, max 100 concurrent
- Relatively complex domain – compared to typical Web-2.0-app, anyway.
- Users cherish snappyness. Response-times above 200 ms makes them call for their 3270s

Historical Background

- Started J2EE 2002
- Back then:
  - Struts-based home-brewn web-framework (action-centric)
  - EJB 1.1 architecture based on „Core J2EE patterns“ by Deepak, Alur, et al.
- 2005: re-evaluation, run-time behaviour was good but: tired of technical anomalies.
- Main objective: „pure“ business logic

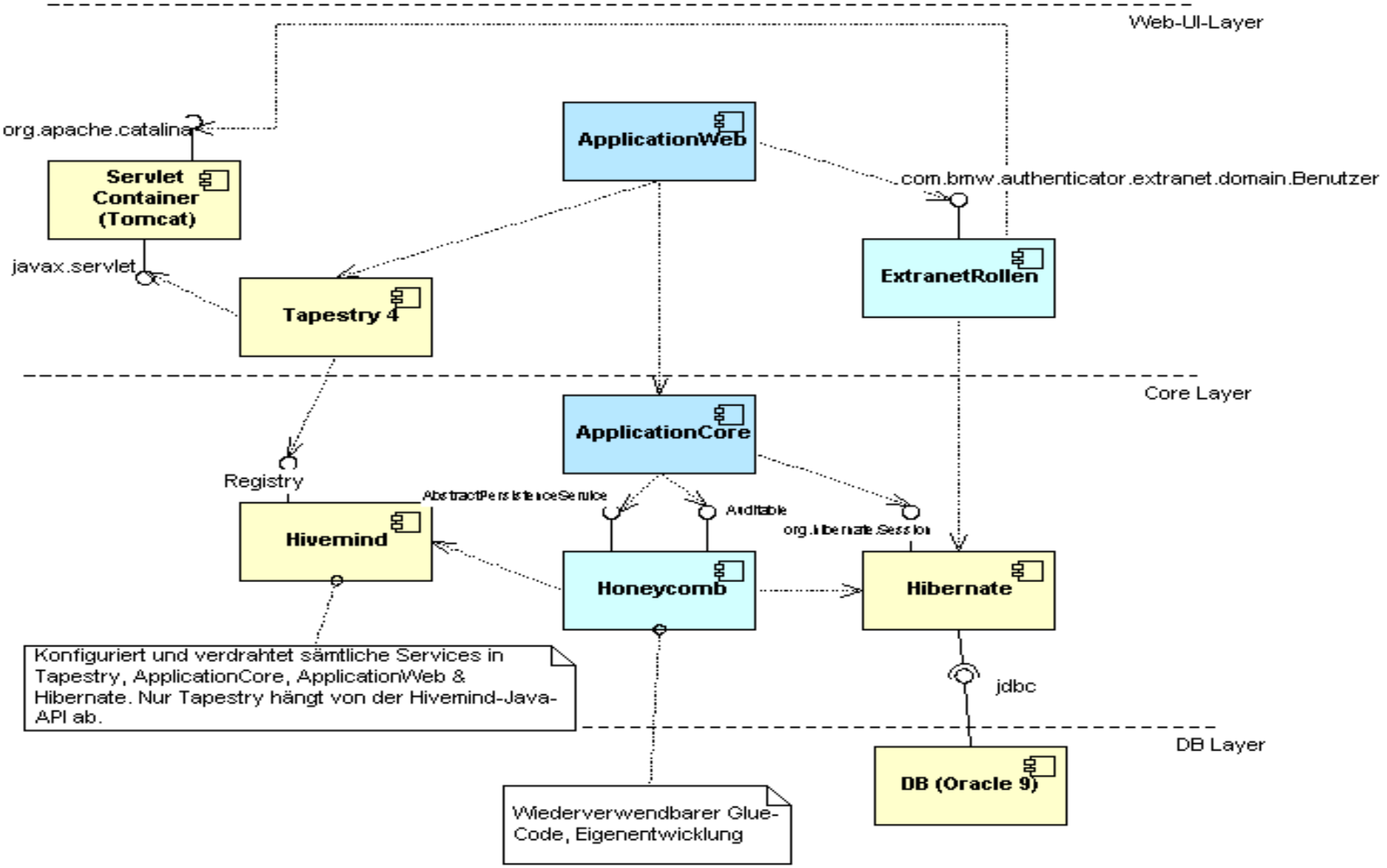
Becoming X-driven

- X= model or X=domain?
- The aim is the same – essentially.
- Continuous abstraction („the Eiffel way“)
- UML-Models tend to be either incomplete or not very abstract.
- Bottom-line: pure Java-domain, no technical slicing of business concerns (PersonEJB, PersonDTO, PersonDAO, PersonDRS, ...)

A Band of Frameworks

- Hibernate – *the* persistence, EJB-3
- Tapestry – „an action is a method“
  - Reusable components
  - Java-types make it through the complete request cycle – you deal with objects everywhere
  - Very clean & customizable architecture
- HiveMind – the glue to assimilate them all
  - IoC Container
  - The jar is the component, Beans need interfaces

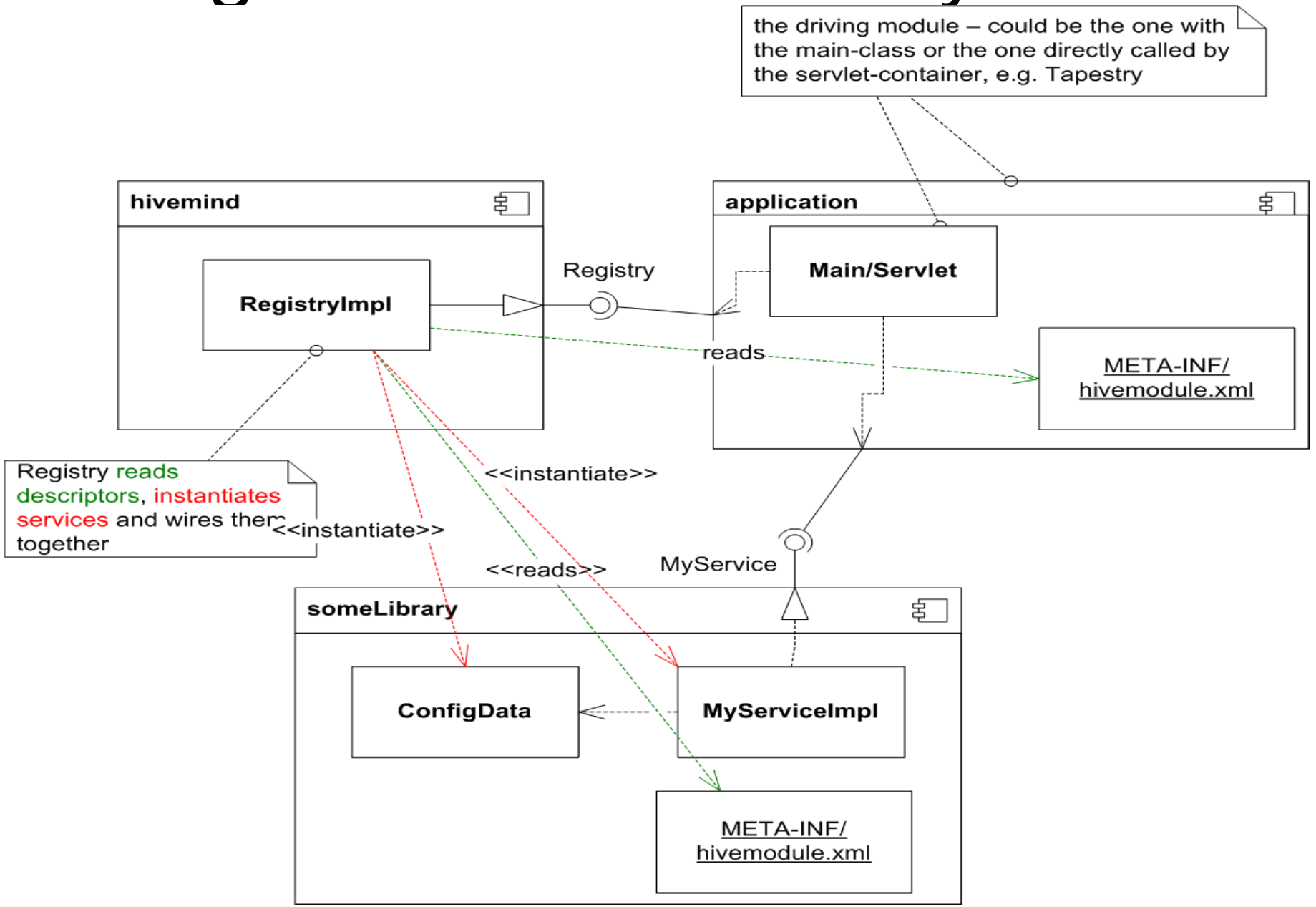
Blueprint


Web-UI-Layer
org.apache.catalina
Servlet Container (Tomcat)
javax.servlet
ApplicationWeb
com.bmw.authenticator.extranet.domain.Benutzer
ExtranetRollen
Tapestry 4
Core Layer
ApplicationCore
Registry
AbstractPersistenceService
Auditable
org.hibernate.Session
Hivemind
Honeycomb
Hibernate
Konfiguriert und verdrahtet sämtliche Services in Tapestry, ApplicationCore, ApplicationWeb & Hibernate. Nur Tapestry hängt von der Hivemind-Java-API ab.
jdbc
DB Layer
DB (Oracle 9)
Wiederverwendbarer Glue-Code, Eigenentwicklung

HiveMind Use-Cases

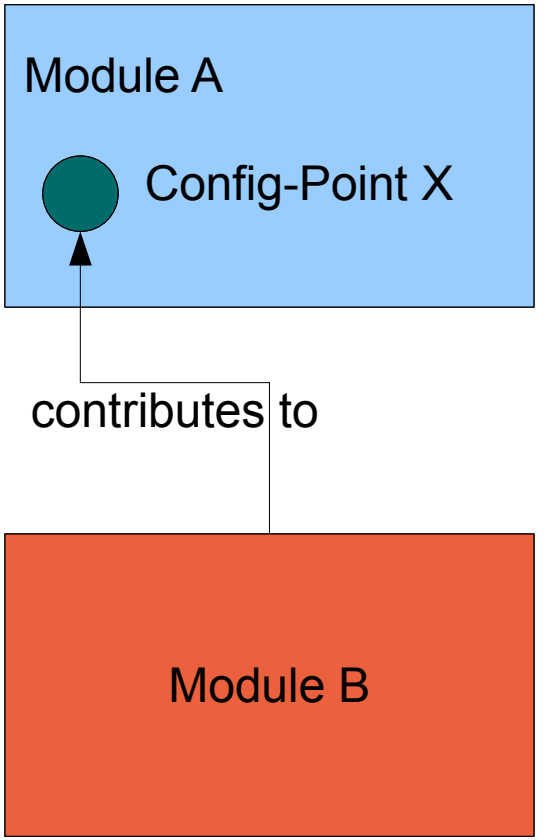
- Using a library-module, wiring up an application
- Customising an application or framework by
  - contributing to configuration points
  - overriding services
- Managing Service-Instances with service-models
- Aside: Spring got Service-models with 2.0, called the scope of a bean there.

Using a HiveMind Library Module

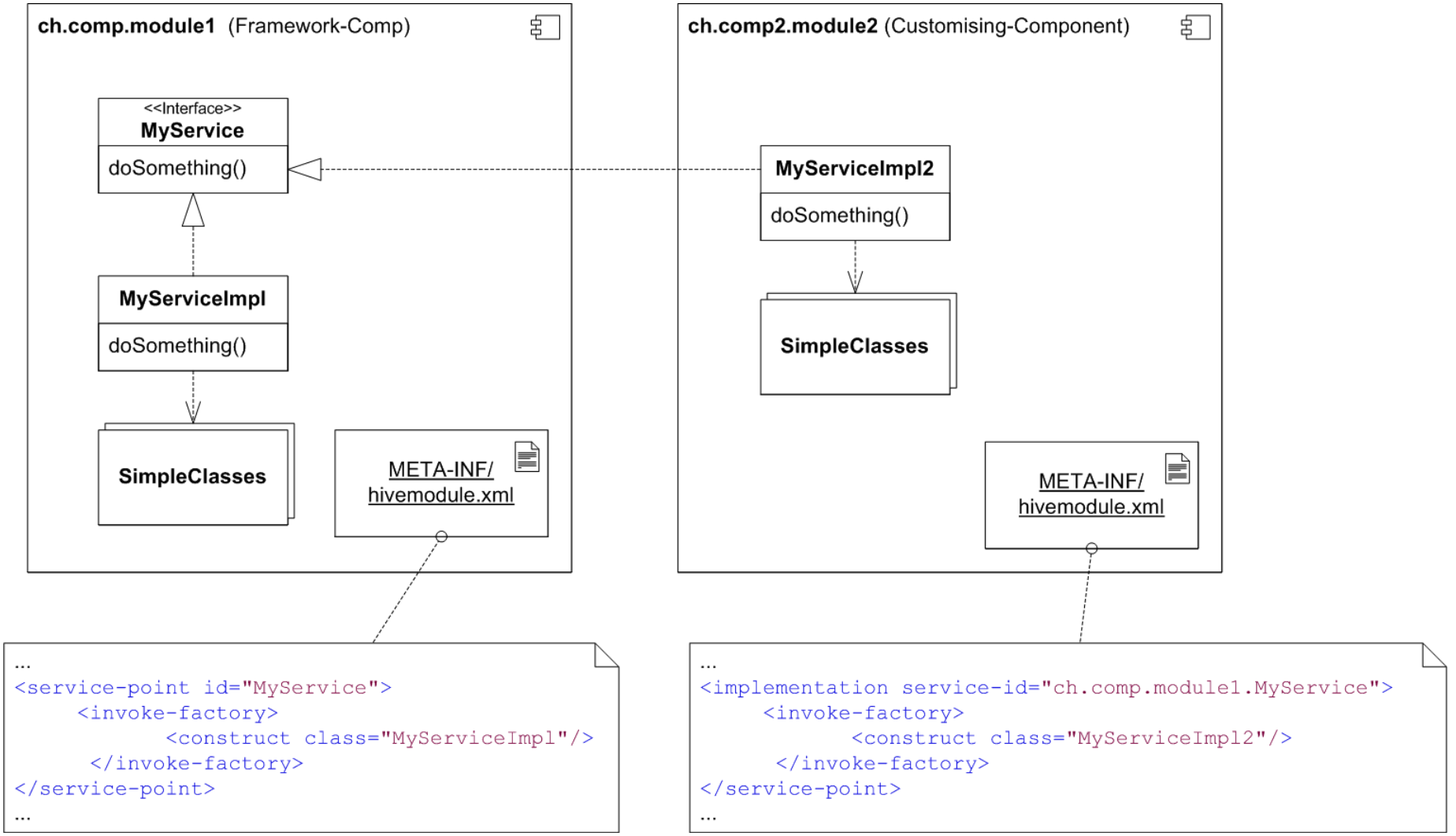


Configuration Points

- Modules define configuration points
- Configuration point adhere to schemas
- Any module can contribute to any configuration point



Overriding a Service



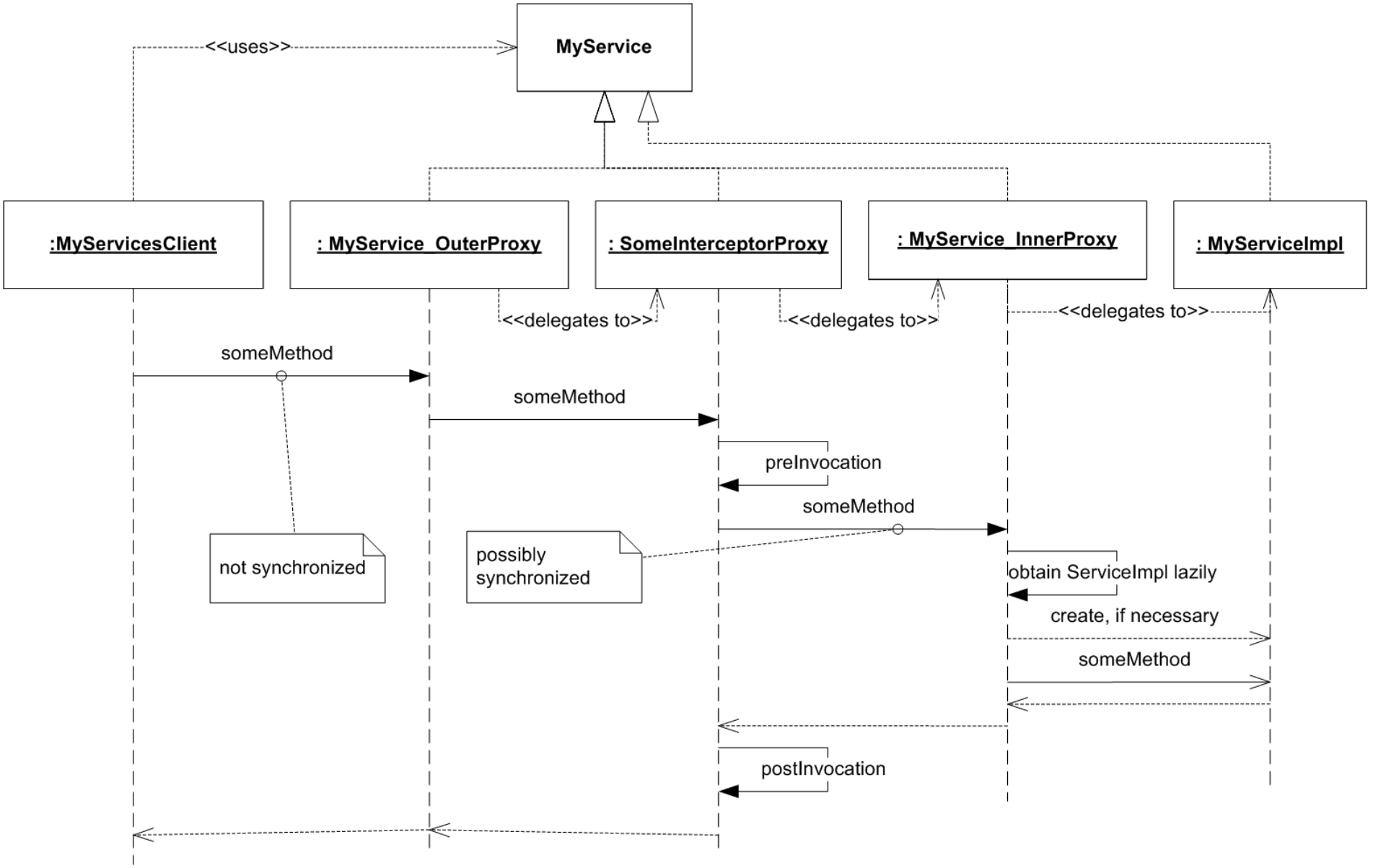
marcus.schulte@s-i.ch

Service-Models

- Primitive (simple class-instantiation)
- Singleton
- Threaded
- Pooled
- Whatever you want, e.g. „stateful“

```
<service-point id="Xyz">
  <invoke-factory model="threaded">
    ...
  </invoke-factory>
</service-point>
```

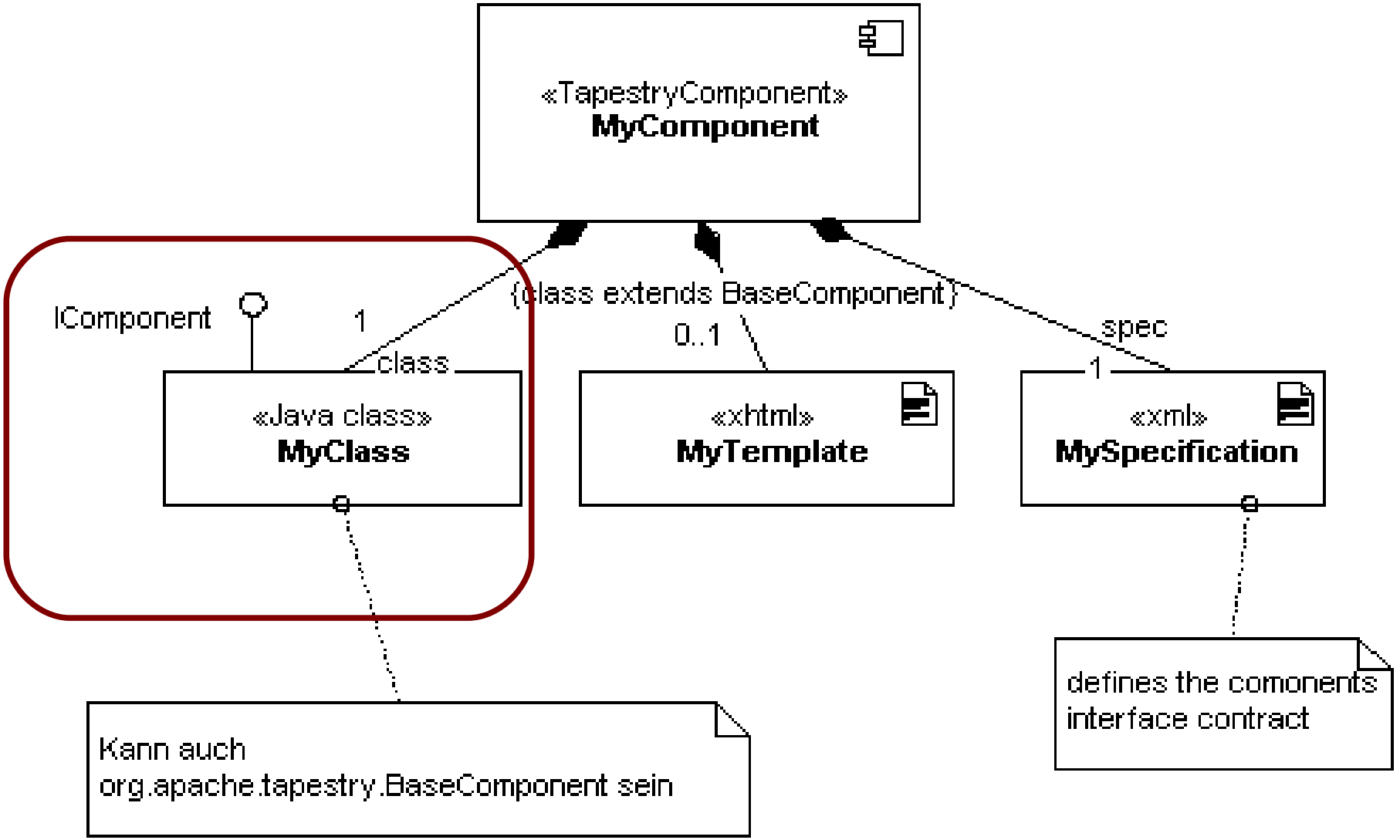
HiveMind Service-Proxies



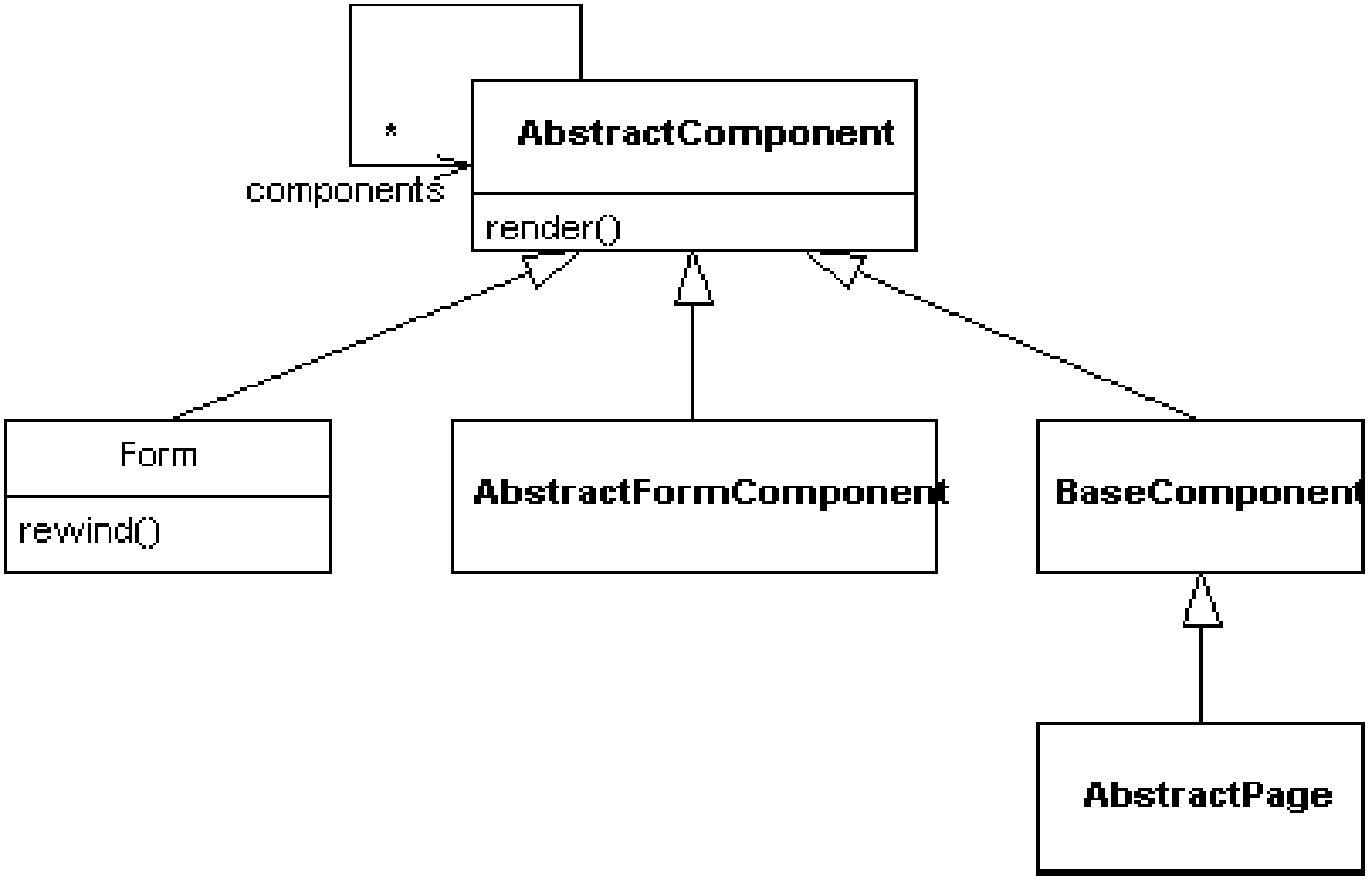
Tapestry starts and a form is submitted

- The App-Servlet instantiates HiveMind Registry
- request comes in, Servlet calls DirectService
- target page pulled from pool
- Service-parameters are decoded, page properties are set up.
- Form rewind is triggered
- Form/button listeners are called
- Response-page renders

Tapestry Components



Tapestry-Components – Composite Pattern



Tapestry Component Interfaces

```
<component-specification class="com.javaforge.honeycomb.tapestry.components.Watch"
    allow-body="no"
    allow-informal-parameters="no">

    <description>Watches a property</description>

    <parameter name="listener" required="yes" />
    <parameter name="value" required="yes" />

    <property name="oldValue"  />

    <component id="hidden" type="Hidden" >
        <binding name="value" value="ognl:oldValue"/>
    </component>

    <inject property="listenerInvoker" object="infrastructure:listenerInvoker"/>

</component-specification>
```

Component Classes

- Simple Java Classes, extending AbstractComponent. Possibly annotated
- Contain (usually):
  - Abstract property-accessors implemented by the framework (javassist) at runtime.
  - Listeners
  - Lifecycle-related callbacks
- Can render their contribution to a page in code or via the associated template (BaseComponent)

A Very Simple Component

```
package com.bmw.fzch.components;

import org.apache.tapestry.BaseComponent;

public abstract class GwbLink extends BaseComponent {

    @Parameter( required=true )
    public abstract Long getGwNr();

    @InjectObject("app-property:gwb.BaseUrl")
    public abstract String getGwbBaseUrl();


    @Component( id="genLink", type="GenericLink",
                inheritInformalParameters=true,
                bindings={"href=gwbUrl"} )
    public abstract IComponent getGenLink();

    public String getGwbUrl() {
        return getGwbBaseUrl()+getGwNr();
    }

}
```

The even Simpler Template

```
<a jwcid="genLink">
    <span jwcid="@Insert" value="ognl:gwNr">1234445</span>
</a>
```

```
public abstract class Watch extends AbstractFormComponent
                            implements PageBeginRenderListener {

    public abstract IActionListener getListener();

    public abstract Object getValue();

    public abstract Object getOldValue();
    public abstract void setOldValue( Object t );

    public abstract ListenerInvoker getListenerInvoker();

    protected void renderFormComponent(IMarkupWriter writer, IRequestCycle cycle) {
        setOldValue( getValue() );
        getComponent("hidden").render( writer, cycle );
    }

    protected void rewindFormComponent(IMarkupWriter writer, final IRequestCycle cycle) {
        getComponent("hidden").render( writer, cycle );
         {
            getForm().addDeferredRunnable( new Runnable() {
                        public void run() {
                            if ( valueChanged() ) {
                                getListenerInvoker().invokeListener( getListener(),
                                                        Watch.this, cycle );
                            }
                        };});
        }

    }
```

Component (Page) Templates

```
<form jwcid="@Form">
    <ul>
        <li jwcid="@For" source="ognl: foos
                          value="ognl: foo"   element="li">

            Name: <input jwcid="@TextField" value="ognl: foo.name" />

            <button jwcid="@Submit"
                    tag="ognl: foo" selected="ognl: fooToBeDeleted"
                    action="ognl:listeners.onDelete" value="delete"/>
            </li>
        </ul>
</form>
```

Page-Class (for previous Template)

```
public abstract class ListOfFoos extends TestAppBasePage {

    public abstract Foo getFoo();
    public abstract Foo getFooToBeDeleted();

    public List<Foo> getFoos() {
        return getPersistenceService().retrieveAllFoos();
    }

    public void onDelete() {
        getPersistenceService().delete( getFooToBeDeleted() );
    }
}
```

Libraries of Components

- Everything inside one jar (templates, images, css, javascript, classes, messages)
- Separate namespace
- Can include HiveMind services/contributions

```
<library-specification>

  <meta key="org.apache.tapestry.component-class-packages"
          value="com.javaforge.honeycomb.tapestry"/>
 <component-type type="ExcelTableLink" specification-path="excel/ExcelTableLink.jwc"/>
 <component-type type="ExcelIcon" specification-path="excel/ExcelIcon.jwc"/>
 <component-type type="Watch" specification-path="components/Watch.jwc"/>

</library-specification>
```

Examples for Powerful Components

- @PropertySelection, @contrib:Palette
- @tacos:Tree
- @contrib:Table and @honey:ExcelLink
- @bmw:ResourceLink @bmw:ReportView
- @contrib:BeanForm (not yet tried myself)

Come with HiveMind Engine-Service

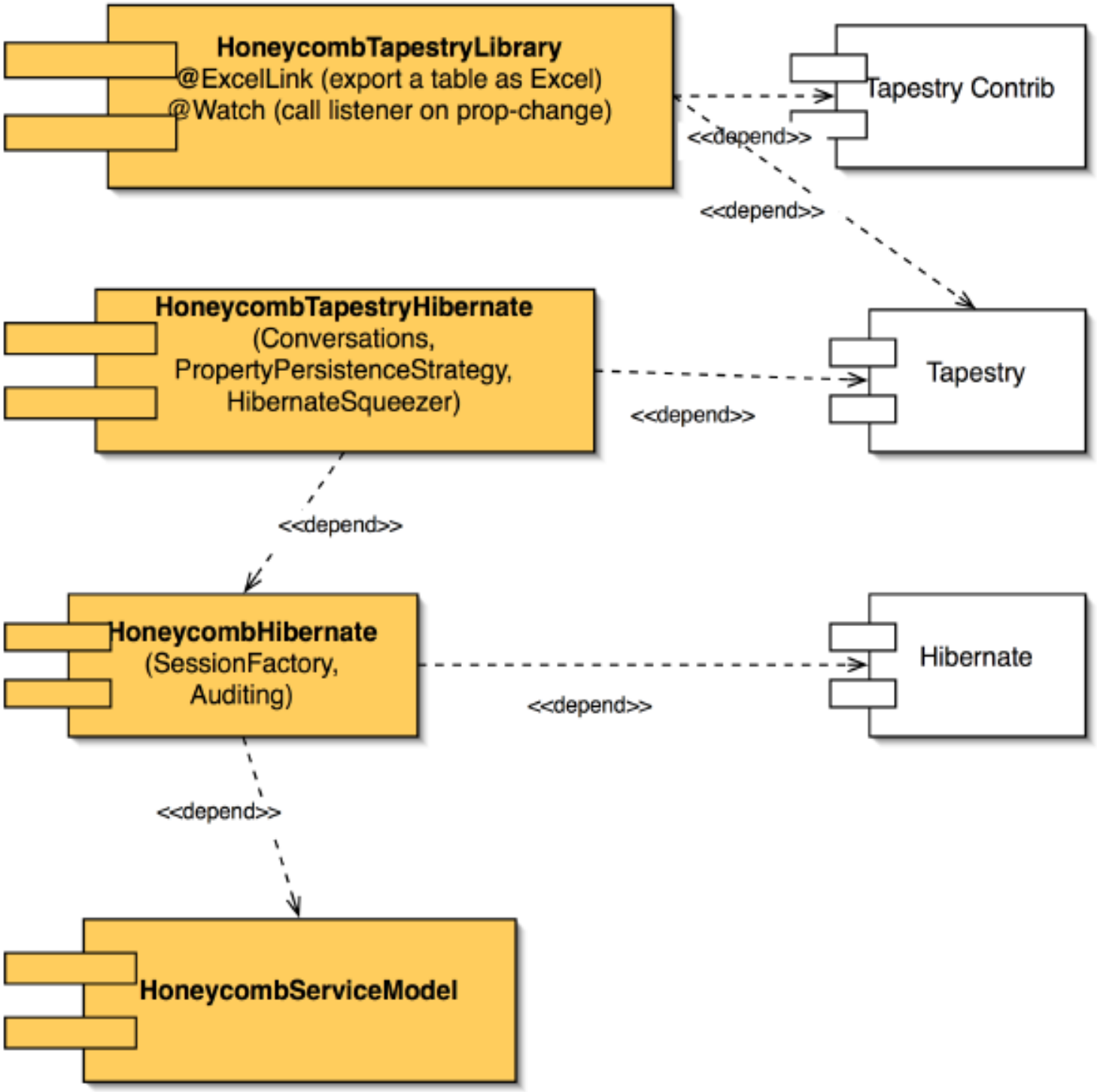
marcus.schulte@s-i.ch

Hibernate – the not-so-plain POJOs

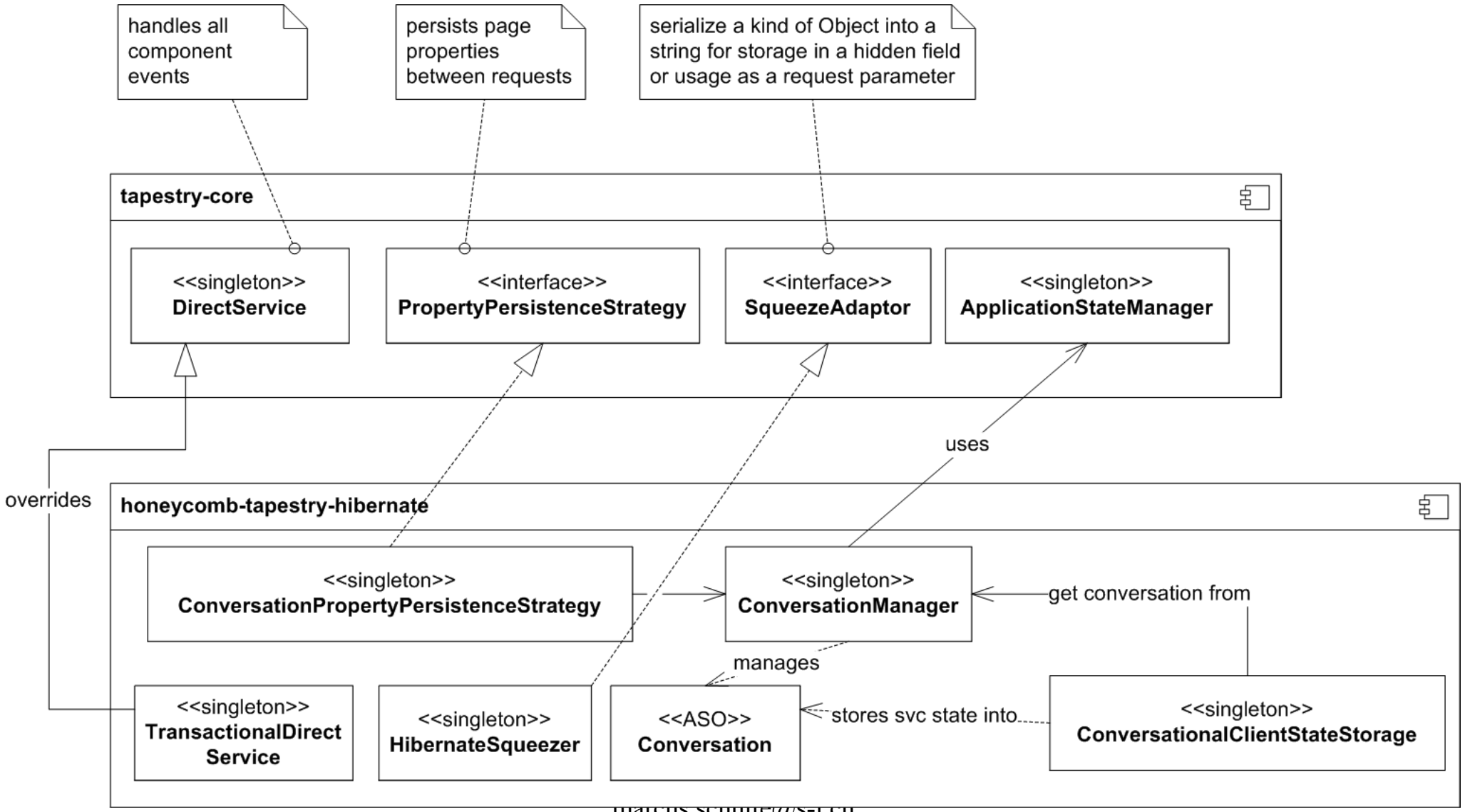
- Domain classes can be POJCs
- But: That doesn't make their instances POJOs.
- Hibernated Entities grow
  - proxies and lazy collections,
  - associations to the session in which they were loaded (dangling reference, when closed)
- Think carefully about your session, units-of-work, working-copy
- LazyInitializationException and NonUniqueObjectException crop up otherwise

Session per Conversation

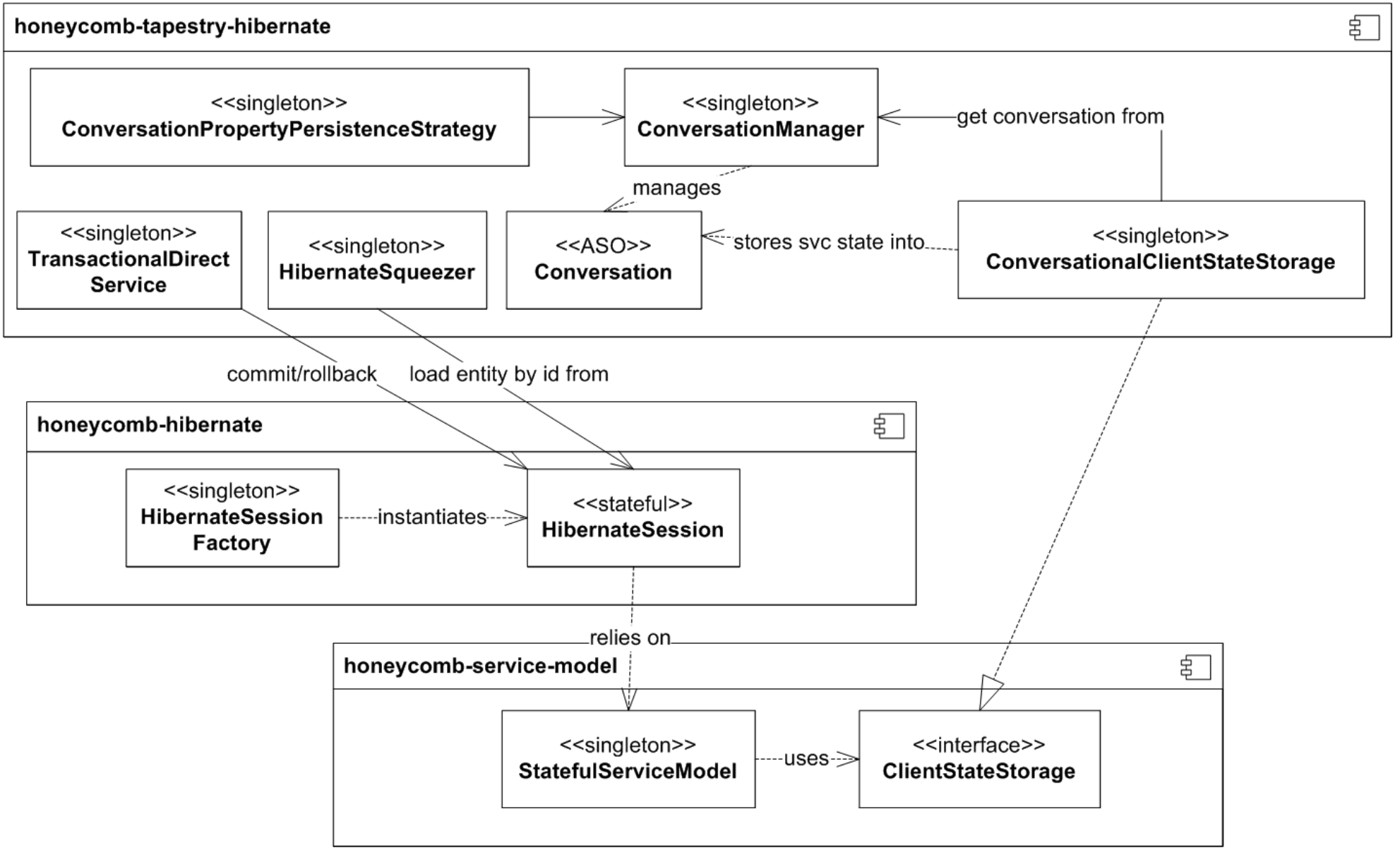
- The life of the first level cache may exceed the life of a request (stateful persistence service)
- An entities in-memory representation (working copy) must not outlive the session in which it was loaded
- No need for detach/re-attach and correct „merge“-mappings
- See also: Seam and http://hibernate.org/42.html

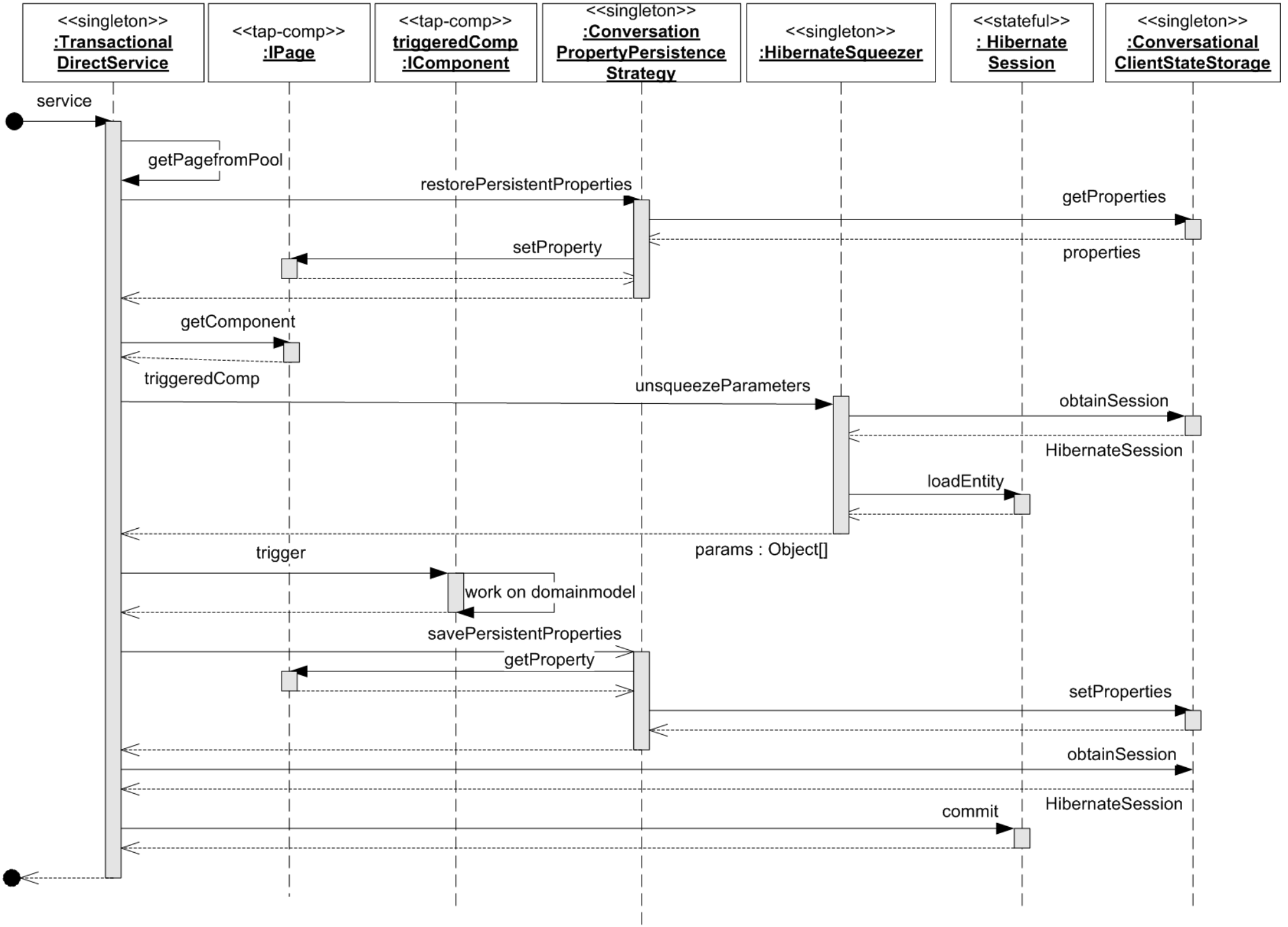

HoneycombTapestryLibrary
@ExcelLink (export a table as Excel)
@Watch (call listener on prop-change)
Tapestry Contrib
<<depend>>
<<depend>>
HoneycombTapestryHibernate
(Conversations,
PropertyPersistenceStrategy,
HibernateSqueezer)
Tapestry
<<depend>>
<<depend>>
HoneycombHibernate
(SessionFactory,
Auditing)
Hibernate
<<depend>>
<<depend>>
HoneycombServiceModel

Plugging into Tapestry



Honeycomb static




<<singleton>>
:Transactional DirectService
<<tap-comp>>
:IPage
<<tap-comp>>
triggeredComp :IComponent
<<singleton>>
:Conversation PropertyPersistence Strategy
<<singleton>>
:HibernateSqueezer
<<stateful>>
: Hibernate Session
<<singleton>>
:Conversational ClientStateStorage
service
getPagefromPool
restorePersistentProperties
getProperties
properties
setProperty
getComponent
triggeredComp
unsqueezeParameters
obtainSession
HibernateSession
loadEntity
params : Object[]
trigger
work on domainmodel
savePersistentProperties
getProperty
setProperties
obtainSession
HibernateSession
commit

Experience with HiveMind/Tapestry

- Excellent reusability of components
- Pervasive, rich domain model.
- Great fun for developers
- Good match for Hibernate (s-p-c)
- Actively developed, helpful community

- Docs sometimes incomplete
- Furiously developed (Tap 5)

marcus.schulte@s-i.ch

Demo & a Duke for Howard

marcus.schulte@s-i.ch