# Smalltalk in Enterprise Applications

ESUG Conference 2010

Barcelona

## About

NovaTec GmbH

- German software consulting company
- Offering full IT  services for complex business applications
- Software engineering, enterprise architectures, integrated system life cycle management, development processes and enterprise application integration

Offices in Stuttgart (Leinfelden-Echterdingen), Munich and Frankfurt

About 120 employees, quickly growing

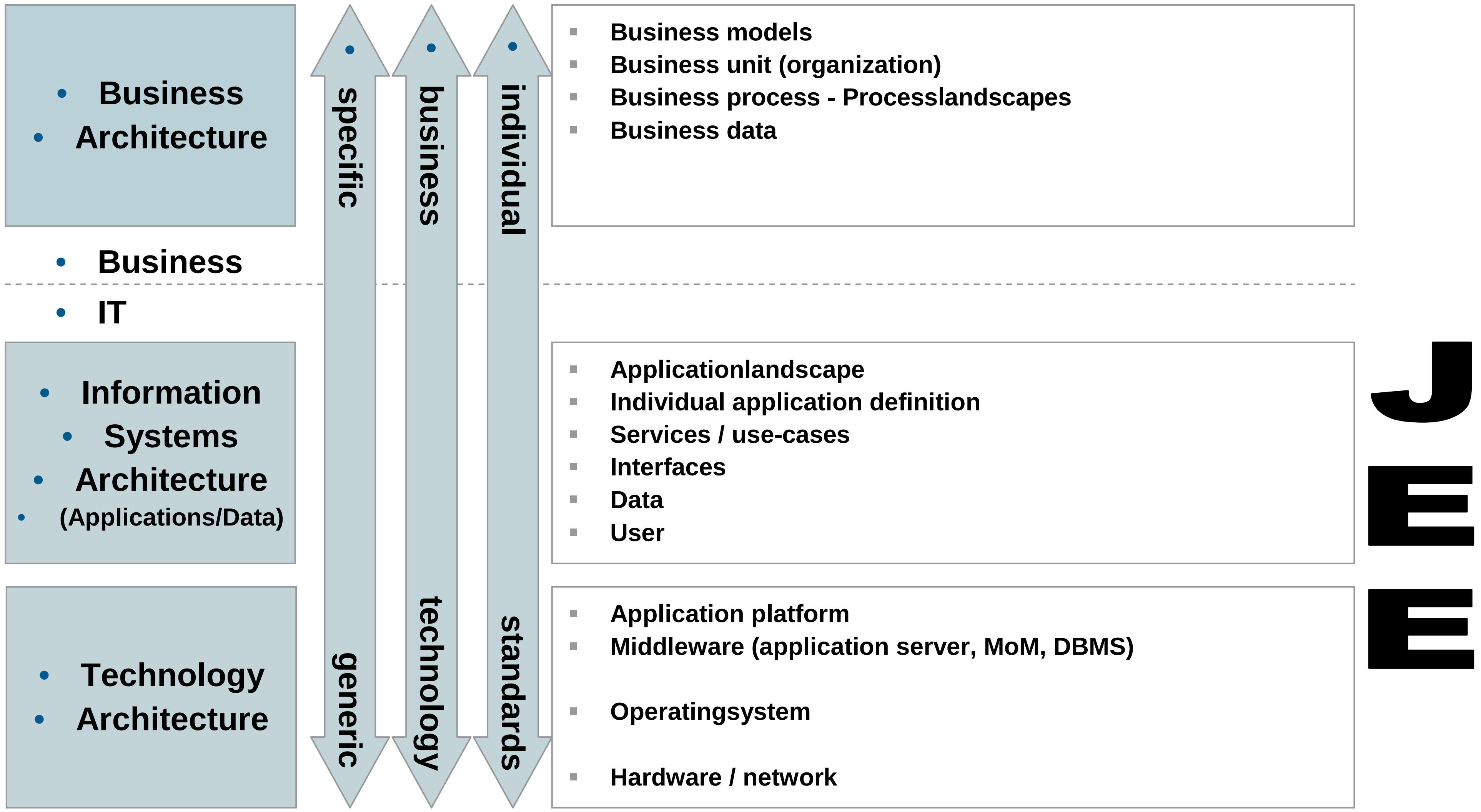
Andreas Tönne

- Managing consultant at NovaTec GmbH, based in Frankfurt
- Responsible for developing the Smalltalk business area
- Working mainly with Smalltalk for 23 years
- Previously: Lead Consultant Europe at Cincom, vice-president at Georg Heeg

NOVATEC
*Make IT happen!*

## Finding a home for Smalltalk in Enterprise Architectures

# Where to Place Smalltalk in the Enterprise (TOGAF)



- Business
- Architecture

- Business models
- Business unit (organization)
- Business process - Processlandscapes
- Business data

- Business
- IT

- Information
- Systems
- Architecture
- (Applications/Data)

- Applicationlandscape
- Individual application definition
- Services / use-cases
- Interfaces
- Data
- User

- Technology
- Architecture

- Application platform
- Middleware (application server, MoM, DBMS)
- Operatingsystem
- Hardware / network

## Let us turn back the clock by 15 years

## Classes in 1995

Smalltalk was mature already

Java fought for reasons to exist

## Classes in 2010

Java is the biggest player in enterprise application development

And Smalltalk?

… is not known in the enterprise

## What happened?

## Wake Up!

This is a wake-up call!

## What to Do?

NOVATEC
*Make IT happen!*

JEE/EJB is here to stay for a long time

Put Smalltalk into action in a JEE-centric application architecture

- Make a Smalltalk system a first-class citizen of JEE
  - ? What needs to be changed in Smalltalk
  - ? What needs to be added to integrate with JEE

## You Are Good, We Are Good!

Smalltalk is good

At the language level

- Less syntax
- Less boilerplate code for a language-level task

Java EE is good

At the architecture level

- Less frameworks to write yourself
- Less boilerplate code for server infrastructure tasks

## JEE is not as complex as it used to be

NOVATEC
*Make IT happen!*

Do you believe that Java EE / EJB applications are

- Awful to write?
- Complex with lots of technical details?

## Hello World – JEE 6 / EJB 3 Style

- Write server component

1. 
```
package org.acme;
@Stateless
@Remote
public class HelloBean {
        public String sayHello() {
                return "Hello World!";
        }
}
```
2. Compile and package
3. Deploy

## Hello World – JEE 6 / EJB 3 Style cont.

- Write remote client application

4. package org.acme;
```
public class HelloClient {
     @EJB HelloBean remoteBean;
    public static void main(String[] args) throws Exception {
            System.out.println(remoteBean.sayHello());
    }
}
```
5. Compile
6. Run

## You Are Good, We Are Good! cont.

- EJB server gave us instant and free
  1. Scalability and robustness for thousands of users
  2. Distributed server architecture
  3. Standard persistence and distributed transactional control
  4. Separation of client and remote business logic in components
  5. Complete component life-cycle management

- Standard set of frameworks
  - Supported by implementations from various vendors
  - Very portable across vendor implementations

**I like this!**

Why did this not become the instant role model for Smalltalk on the server?

Java took its time

## Java Timeline

NOVATEC
*Make IT happen!*

- 1995 – Java introduced to the public
- 1996 – JDK 1.0 released, NovaTec founded
- 1997 – JavaOne conference attracted 8.000 attendees, JDK 1.1, EJB 1.0
- 1999 – Java 2, J2EE goes beta, JavaOne has 20.000 attendees
- 2001 – JEE SDK has 1 mil. downloads
- 2002 – 2 mil. downloads
- 2003 – Java runs on 550 mil. desktops, Eclipse released
- 2004 – Java EE5 released
- 2005 – 10 years old, 4.5 mil. developers worldwide
- 2006 – current Java SE6 released with 21 updates, Java 7 coming this year
- 2009 – current Java EE6 released

**Java showed the most impressive and successful run in computing history ever**

## What Did the Java Community Learn?

- **POJO**: Separate the business logic from the infrastructure
  - Remove dependency on technical superclasses and framework required code
- **Bean**: A lightweight component model that works
  - Supports abstract encapsulation, implementation independence, portability
  - Allows tools and frameworks of independent vendors
- **Dependency Injection**: Declarative composition of objects removes need for "plumbing"
  - Separation of concerns – business developer vs. framework supplier
- **Annotation**: Meta-programming reduces complexity and framework skills needs
  - Avoid cross-cutting concerns
  - Support POJOs
- **Container**: Externally supplied bean life-cycle and infrastructure is good

## Container

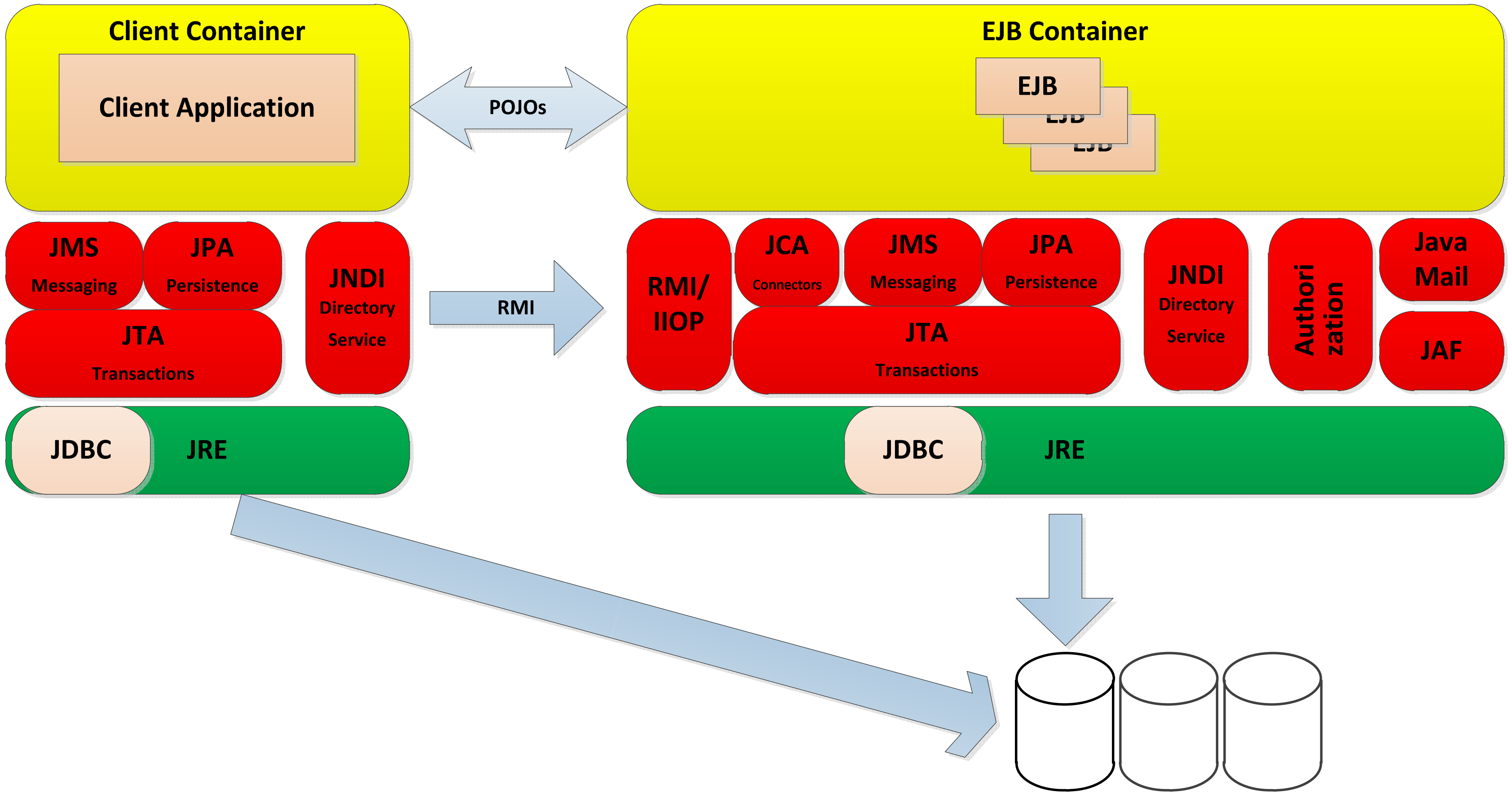
Container: Operating system for running (enterprise) beans

Different types of containers for different kinds of beans

- Key benefits:
  - Separate the business logic from the server infrastructure
  - Make the business logic independent of server vendor implementation
  - Framework and tool-supported component life-cycle, logging, transactions, …

## EJB Container-Architecture of JEE

NOVATEC *Make IT happen!*



## Dependency Injection and Annotations

Pluggable business components, resources and infrastructure objects

- Declaratively by annotations or XML files
- Container builds the beans

```
@Stateless
@Remote
public class EmployeeDemoSessionBean {
    @PersistenceContext EntityManager entityManager;
    public Employee createEmployee(String firstName, String lastName) {
            Employee employee = new Employee();
            employee.setFirstName(firstName);
            employee.setLastName(lastName);
            entityManager.persist(employee);
            return employee;
    } ...
}
```

## And Smalltalk in the Enterprise?

NOVATEC
*Make IT happen!*

**Smalltalk in the enterprise spells: Do It Yourself!**

- No standard component model
- Very little declarativeness and meta-programming in applications
- No abstract standard for application infrastructure
- **Every project has to implement its own server infrastructure architecture**

## Do It Yourself – One Example

- Thread-Pooling
  - http://onsmalltalk.com/2010-07-28-a-simple-thread-pool-for-smalltalk (Ramon Leon)
  - „So, let's write a thread pool.“
  - James Robertson reply about BottomFeeder „What I should have done is what Ramon did - write a more general solution instead of burying it inside an app. „
- It seems it it too simple to write necessary infrastructure ad-hoc

**Not everyone is able to write a good thread-pooling framework for high loads**

## Why Smalltalk in the Enterprise? Why Bother?

Complex business concept modelling

Rapid domain changes

Translating between domains

And there is a lot of valuable legacy Smalltalk code…

Under pressure to integrate in JEE or go away

## Smalltalk Enterprise Edition 1.0

Do not try to reinvent or replace the JEE server!

Model architecture and infrastructure abstractions of JEE

Interface with JEE to make Smalltalk components first class citizens of JEE

**Turn Smalltalk into a "shadow container" of Smalltalk components**

## Benefits of a Smalltalk Server Application Architecture

Be part of the overall enterprise architecture and application design

Improve the server qualities of Smalltalk

Simplify the JEE-Smalltalk interface

Attract standardized server development on a broader scale

- Make 3rd party tools and framework development more attractive

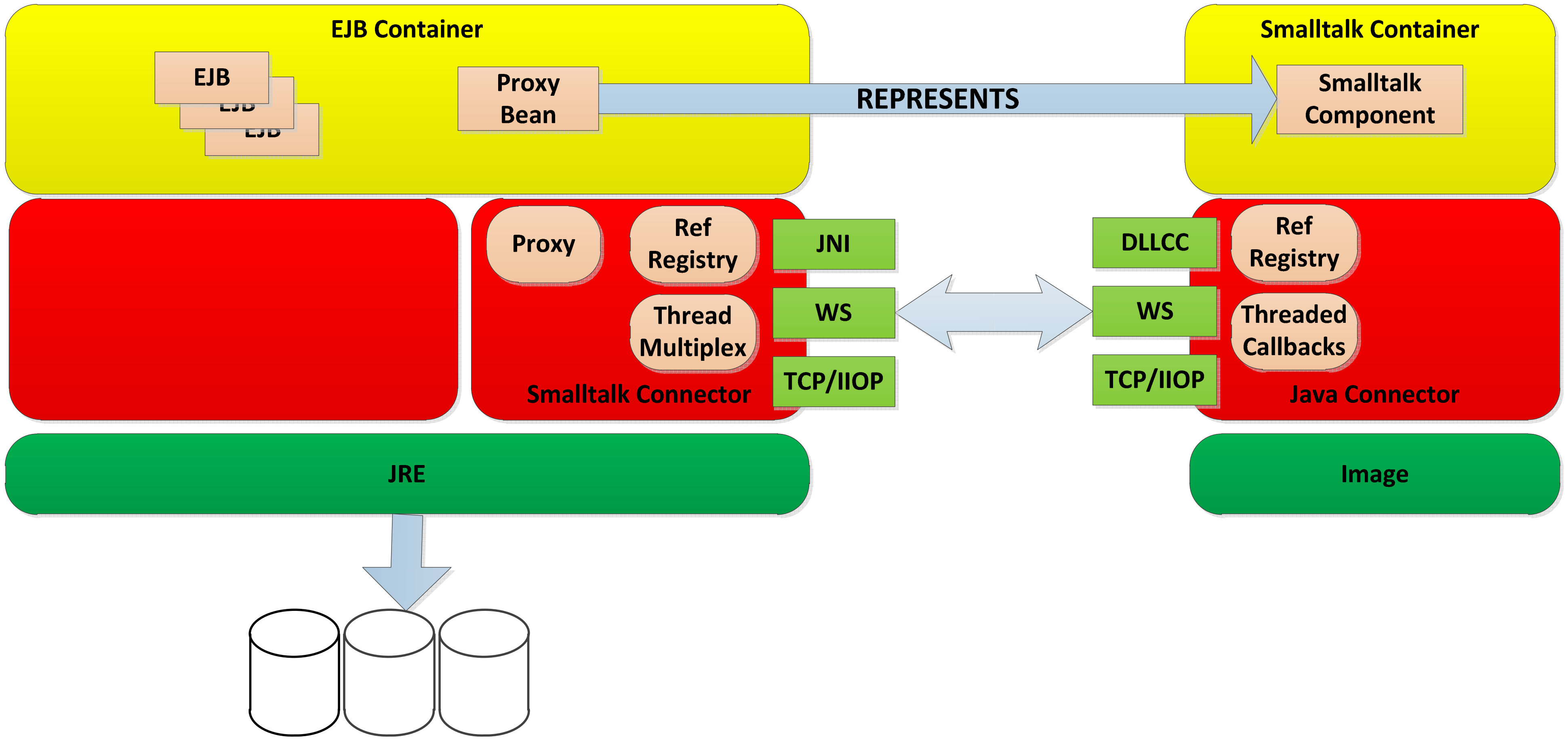
## Smalltalk Enterprise Edition 1.0 - TODO

- Introduce a component model
- Add more meta-programming using pragmas or similar
- Provide standard abstract frameworks for server tasks
- Abstract model of factories with dependency injection
- Abstract model of containers

- Bean-level interface between JEE and Smalltalk

# Smalltalk EE – Architecture Sketch (Java calling ST)

NOVATEC
*Make IT happen!*



## Which Smalltalk Components in Version 1.0?

- Session beans for threaded services
  - Stateless for massively concurrent single tasks
  - Stateful for workflow oriented tasks
- Smalltalk should export session beans
- Proxy Java Bean (generated) only interface to use a Smalltalk component
  - Transactional control and security through this proxy
- JCA (Java Connector Architecture) option for further transactional and security integration
- Persistence is a topic for V1.1

## Which Components - Sketch

- Smart defaults, optional types for remoted properties and services
- Standard life-cycle callbacks available

```
<bean: 'HelloWorld' stateful: true remote: true local: true>
Smalltalk defineClass: #HelloWorldBean
    superclass: #{Core.Object}
    indexedType: #none
    private: false
    instanceVariableNames: 'name '
    classInstanceVariableNames: ''
    imports: ''
    category: ''
```

```
name
    <propertyType: #string>
    ^name

name: aString
    name := aString

sayHello
    <serviceParameterTypes: #() returnType: #string)>
    ^'Hello World to ', self name
```

## Which Containers?

- Container = Factory + Maintenance of bean per bean kind
- Consistently based on
  - Dependency injection for bean construction
  - Meta-programming controlled through pragmas or deployment descriptors
- Creation of bean instances through a lookup mechanism
- Referencing and finding bean
- Handling the general life-cycle

- Concrete containers may come in a wide variety
  - Pooling
  - Threaded
  - Load limiting like persisting inactive components
  - Bean instrumentation like logging or transactions

# Component Deployment Descriptors - Sketch

```
Smalltalk defineClass: #MockupContainer
        superclass: #{SEE.BeanContainer}
        ....
beanSetup
        <deploymentDescriptor: #SampleContainer>
        self bean: #HelloWorld with:
                [:bean |

                        bean implementor: HelloWorldBean.
                        bean isStateful.
                        bean isRemote.
                        bean isLocal.
                        bean property: #name with:
                                        [:prop |
                                        prop getter: #name.
                                        prop setter: #name:.
                                        prop type: SEE.Types.String].
                        bean service: #sayHello with: [:svc | svc returnType: SEE.Types.String]]
```

## Which Container – Local Usage Sketch

```
context := LocalContainerContext for: #SampleContainer.

bean := context lookup: #HelloWorld.

bean name: 'Barcelona'.

Transcript show: bean sayHello; cr.
```

## Which Container – Local Usage Sketch cont.

NOVATEC
*Make IT happen!*

```
greetESUG
        <inject: #HelloWorld into: #hello>
        | hello|
        hello name: 'Barcelona'.
        Transcript show: hello sayHello; cr
```

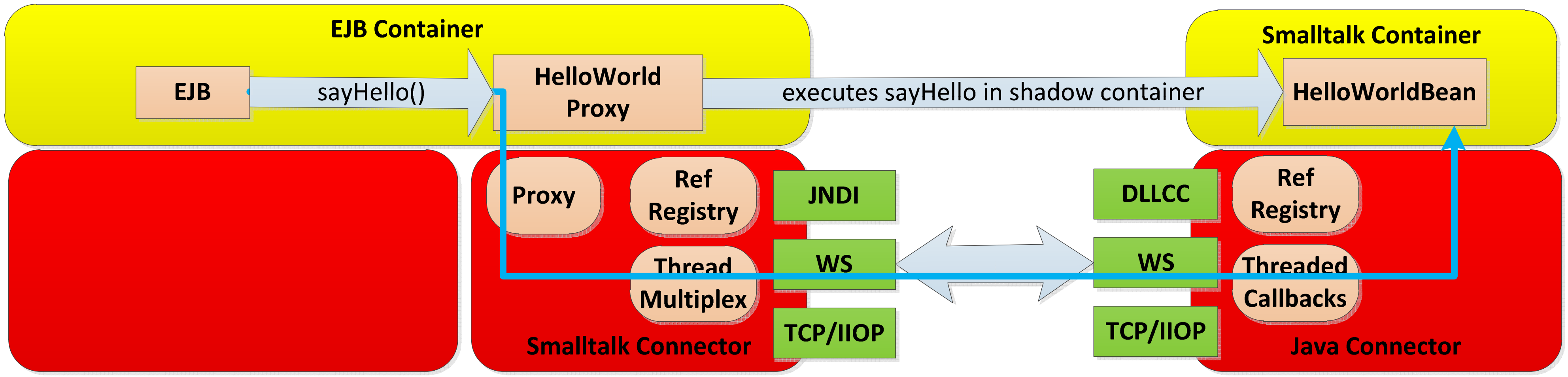
- No local context, no lookup
- Injected component changable in deployment descriptor

## Interfacing with JEE - Principles

NOVATEC
*Make IT happen!*

- Remoted Smalltalk component represented by local Java EJB
- Imported Java beans represented by local Smalltalk components
- Cross-container communication only through connector
- Clients talk to the EJB server, never to Smalltalk directly

- Smalltalk is a "shadow container" behind the EJB server



## Interfacing with JEE - Options

- Using JNI to run EJB- and Smalltalk-VMs in the same process space
  - JNIPort or JavaConnect via VMasDLL (VisualWorks)
  - Starting the Java server from Smalltalk via DLLCC is NO OPTION
  - Threading a serious issue
- Using web-services as the transport layer
  - Possibly the simplest and slowest option
- Using a fast custom IIOP or someone please implement RMI

- Add JCA for more complete transaction and security integration

## Summary

NOVATEC
*Make IT happen!*

- Java's success in the enterprise has created good progress in standard enterprise application server infrastructure
- Smalltalk is lacking a similar platform/vendor independent infrastructure
- This makes Smalltalk inacceptable for enterprise applications by common standards
- JEE improved through proper abstractions of technical infrastructure and by making good use of meta-programming
- Smalltalk should add a standardized server application architecture
- A JEE connector will allow Smalltalk to be used as a shadow container behind a EJB server

- Give existing Smalltalk applications a longer life
- Introduce new opportunities for Smalltalk projects as part of a JEE project