Component Patterns

Architecture and Applications with EJB

Markus Völter, Oliver Stuch
MATHEMA AG

{markus.voelter|oliver.stuch}@mathema.de

MATHEMA

Roadmap

- Patterns and Pattern Languages
- Basic Principles
- Core Patterns

Patterns and Pattern Languages

Patterns and Pattern Languages

- patterns have become part of the mainstream
  - patterns for software design
  - patterns for software architecture
  - organizational patterns
  - pedagogical patterns

What is a pattern?

Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain software configuration which allows these forces to resolve themselves.

Jim Coplien

- Forces
- Problem
- Solution
- Structure
- QWAN

What is a pattern language?

- Systematic collection of patterns
  - Has a language-wide goal
  - Is generative in nature (generates the „whole“)
  - has to be applied in a specific way
  - each pattern must define its place in this sequence

Form of the patterns here

- Alexandrian Form
- Pattern consists of
  - **Name**
  - Context
  - **Problem**
  - Body
  - **Solution**
  - Resulting Context
  - *Examples*

COMPONENT HOME **

A Core Technical Infrastructure Pattern

You have decomposed the functionality into COMPONENTS. Your application is assembled of collaborating, loosely coupled components, i.e. one COMPONENT uses the services of another COMPONENT.

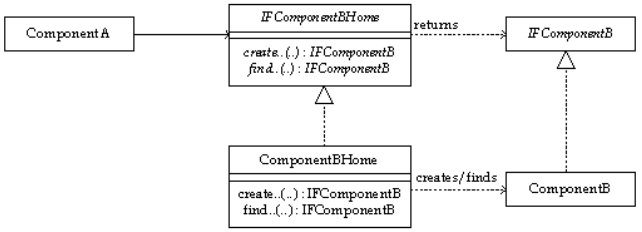
***

A COMPONENT needs to find (or create new) instances of other COMPONENTS. You might not know the exact procedure of how to create or find such COMPONENTS, because this depends largely on the technical concerns (which you do not care about because they are the responsibility of the CONTAINER.) You do not want to code this knowledge into your COMPONENTS.

For example, an instance of a required COMPONENT can live in another CONTAINER which might even reside on another machine. Depending on the component technology, the COMPONENT might even be implemented in another programming language.

Therefore:

For each Component, provide a management interface that can be used by other clients to create and find other COMPONENT instances. It can provide different ways (operations) how to create and find component instances, depending on the COMPONENT type (ENTITY, SERVICE, PROCESS).



***

The Component Home is also an interface. I.e. the CONTAINER is free to implement it, in a way that suits its technical requirements. The steps required to implement the interface can be arbitrarily complex. This is usually done during COMPONENT INSTALLATION.

When creating a component (bean) in EJB, the programmer has to define a home interface in addition to the bean's interface. Depending on the component type, he can (must) specify several create, remove, and find operations. The create and find operations can be overloaded with different signatures, they serve as constructors for the component. Note that the programmer never really implements the home interface. Depending on the bean type and whether container managed or bean managed persistence is used, the programmer has to implement some of these LIFECYCLE CALLBACK operations in the bean class itself. The implementation of the home itself is done by the CONTAINER, to allow for INSTANCE POOLING and PASSIVATION.

MATHEMA

This pattern language contains...

- architectural and design patterns

Architecture:
*The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the **externally visible** properties of those components, and the relationships among them.*

Basic Principles

Separation of Concerns

- Problem:
  - Technology is changing fast.
  - Your business is changing fast.
  - The changes happen at different speeds.
- Solution:
  - Separate functional and technical concerns
  - Implement them separately using different software artifacts.
  - Reuse and evolve each of them separately

Multitier Architecture

- Problem:
  - Enterprise applications should be scalable
  - They must be simple to deploy
  - Different user interfaces are required
  - The data should be stored centrally
- Solution:
  - Split the application system into several layers
  - Allow remote access to each of these layers
  - Introduce a specific layer for the business logic

Core Patterns

Component

- Context:
  - You have Separated Concerns. Functional part is “one chunk”.
- Problem:
  - Your business (requirements) is changing quickly.
  - Need reuse on enterprise level, i.e. on high granularity level
  - Want to evolve the parts of your system(s) independently
- **Solution:**
  - **Decompose your application into several components**
  - **They do not directly depend on other components**
  - **An application consists of loosely coupled components**

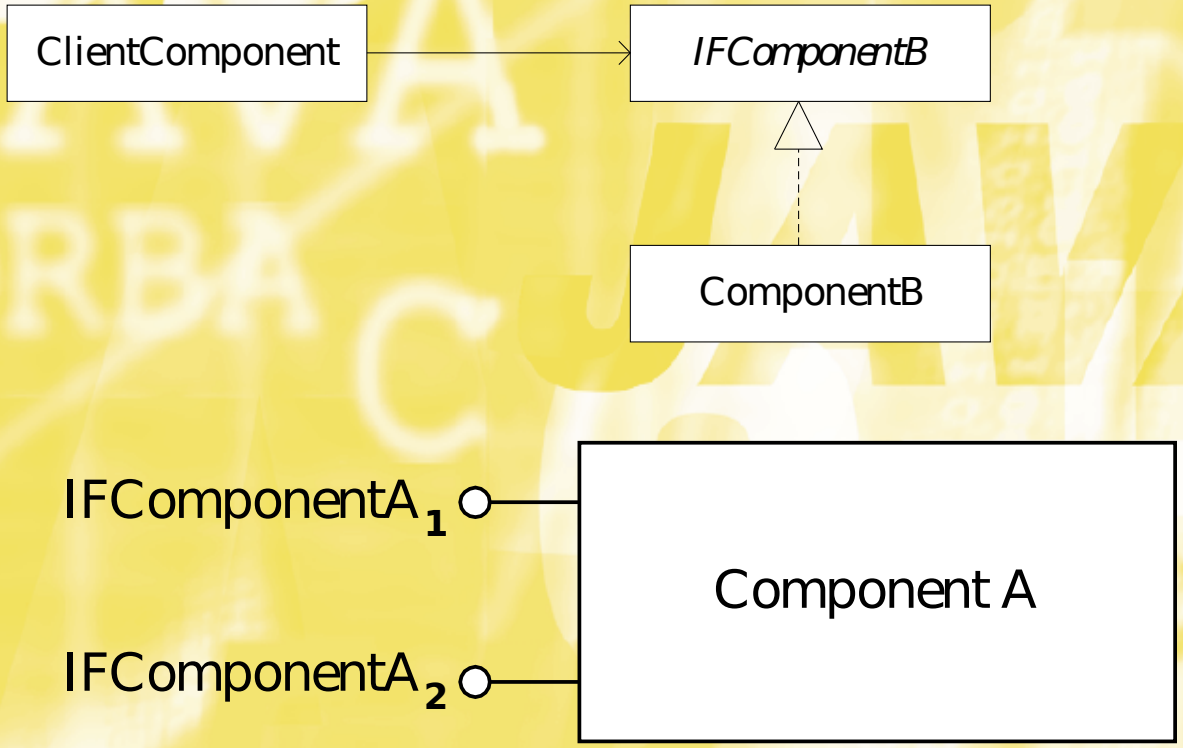
Component


Monolithic Application with features A, B, C
decompose
ComponentA
ComponentB
ComponentC

Component Interface

- Context:
  - You Decomposed functional requirements into Components. Now: "reassemble" app by letting components collaborate.
- Problem:
  - Components should not depend on other components' implementations
  - You do not even want to know how another component is implemented
- **Solution:**
  - **Define public interface to the component**
  - **Client accesses a Component using the interface only**
  - **Accessing this interface should be standardized**

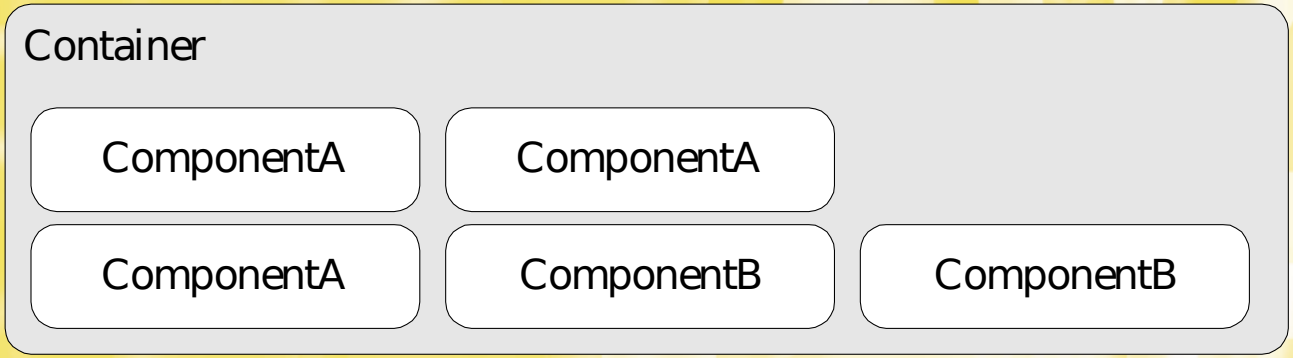
Component Interface



Container

- Context:
  - You have decided to Separate Concerns.
- Problem:
  - Components contain functional logic
  - Now you need something for the technical parts
  - Need to recombine functional and technical requirements into a complete application.
- **Solution:**
  - **Create a container for the Components.**
  - **Responsible to enforce technical requirements on the components**
  - **Uses standardized frameworks and other techniques such as code generation.**

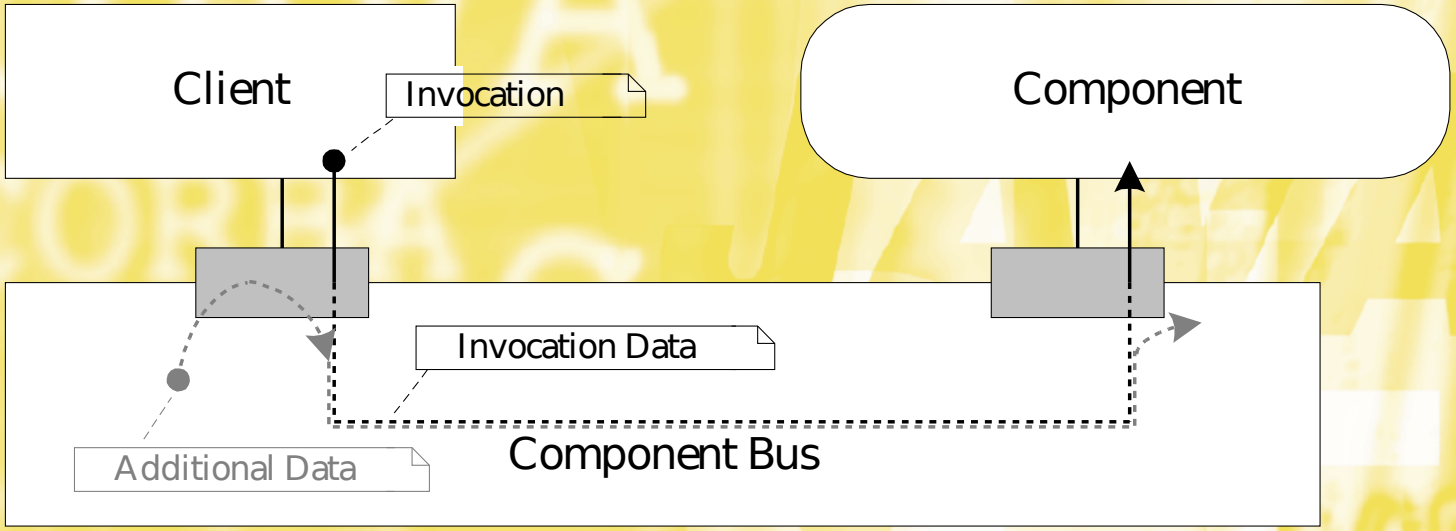
Container



Component Bus

- Context
  - You have a Container to host your Components.
- Problem
  - Containers (and therefore, your components) usually reside on different machines that your client application
  - You don’t want to depend on te semantics of the underlying transport protocol.
- **Solution**
  - **Component bus, as logical communication infrastructure**
  - **Hide the underlying low-level transport protocol**
  - **The Container and the clients are attached to the bus.**

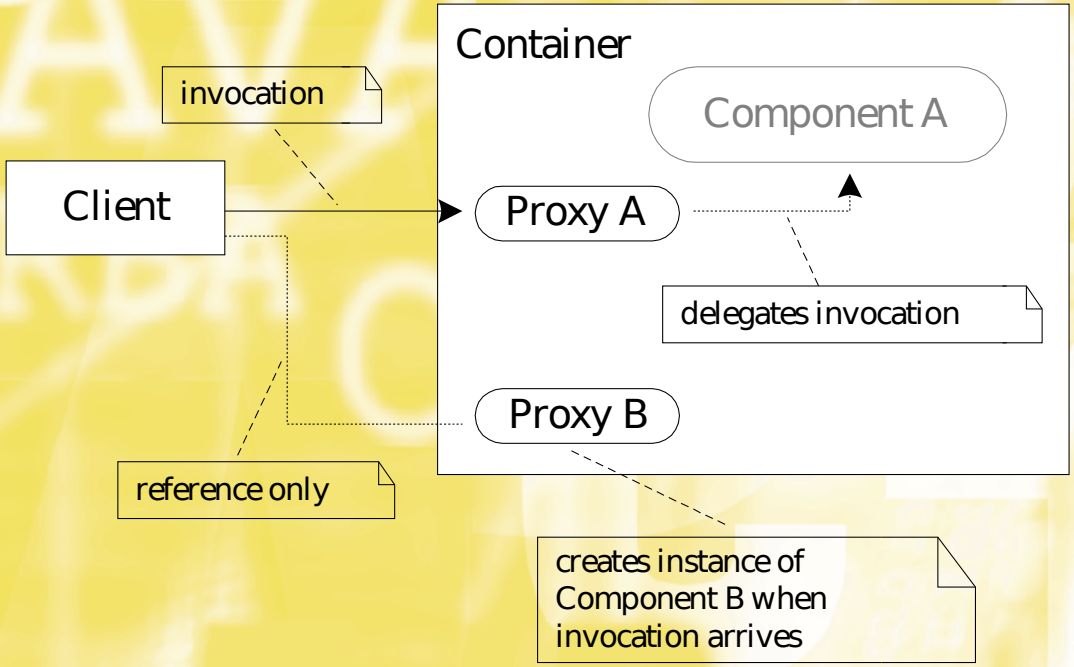
Component Bus



Distinguish Identities

- Context
  - Your design results in potentially many logical component identities, especially when you use Entity Components.
- Problem
  - Many logical component instances referenced by clients at the same time
  - Container will run into resource problems
- **Solution**
  - **Distinguish logical and physical identities**
  - **Clients never have a reference to the physical component instance, they use a Proxy**
  - **Container is free to assign physical instances to logical identities**

Distinguish Identities



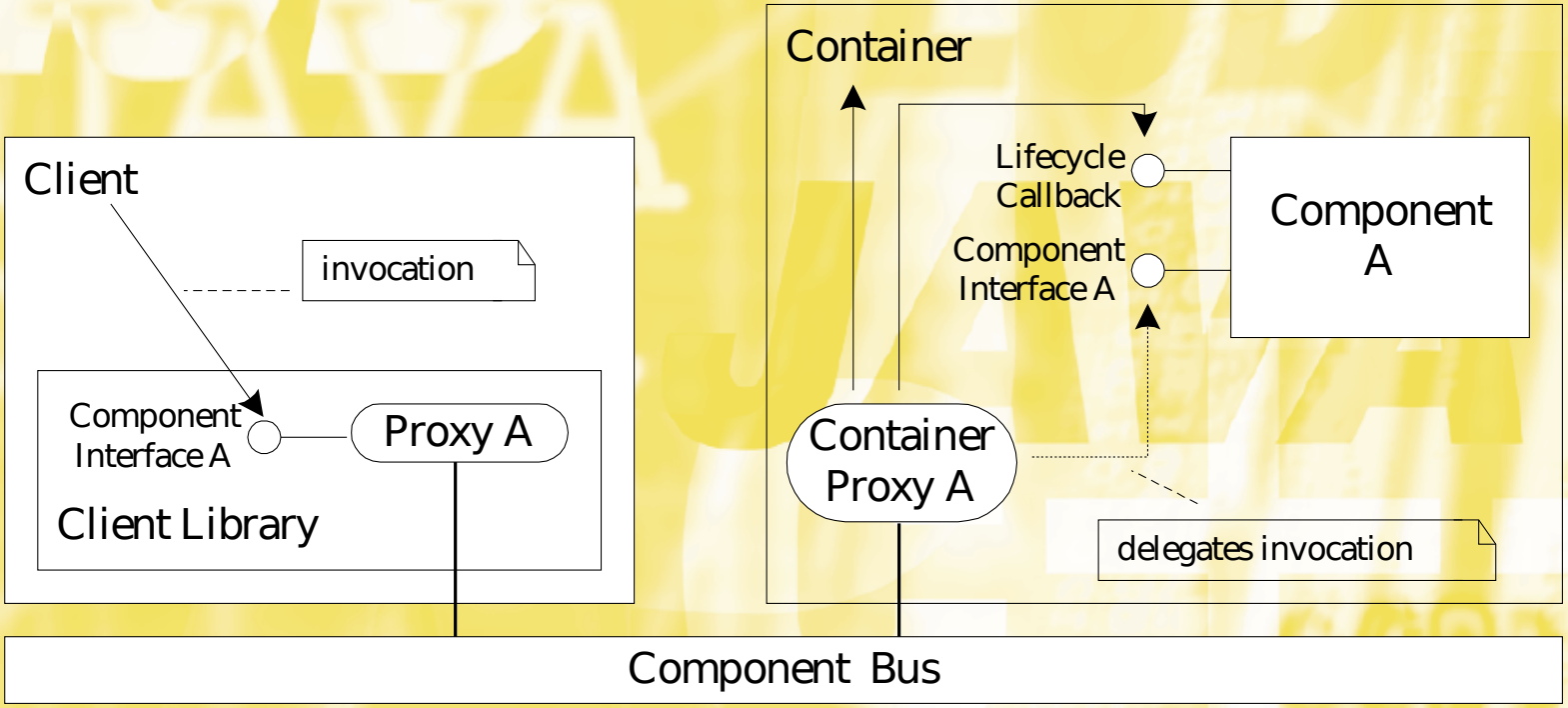
Lifecycle Callback

- Context
  - Your Components live in a Container and you Distinguish Identities.
- Problem
  - Physical component it has to change its logical identity
  - Component will have to be initialized after birth and it needs to return its resources before it dies
- **Solution**
  - **Provide a set of lifecycle callback operations.**
  - **The Container calls them whenever it feels it is necessary.**

Client Library

- Context
  - You are using a Component Bus to access your Components in the Container.
- Problem
  - The client application needs to access the Component Bus
  - Needs to know the interfaces of the components
  - Every method invocation must contain security and other information
  - Specific marshalling code may also be necessary
- **Solution**
  - **Create a client library upon Component Installation**
  - **Contains all the interfaces, and other generated code**

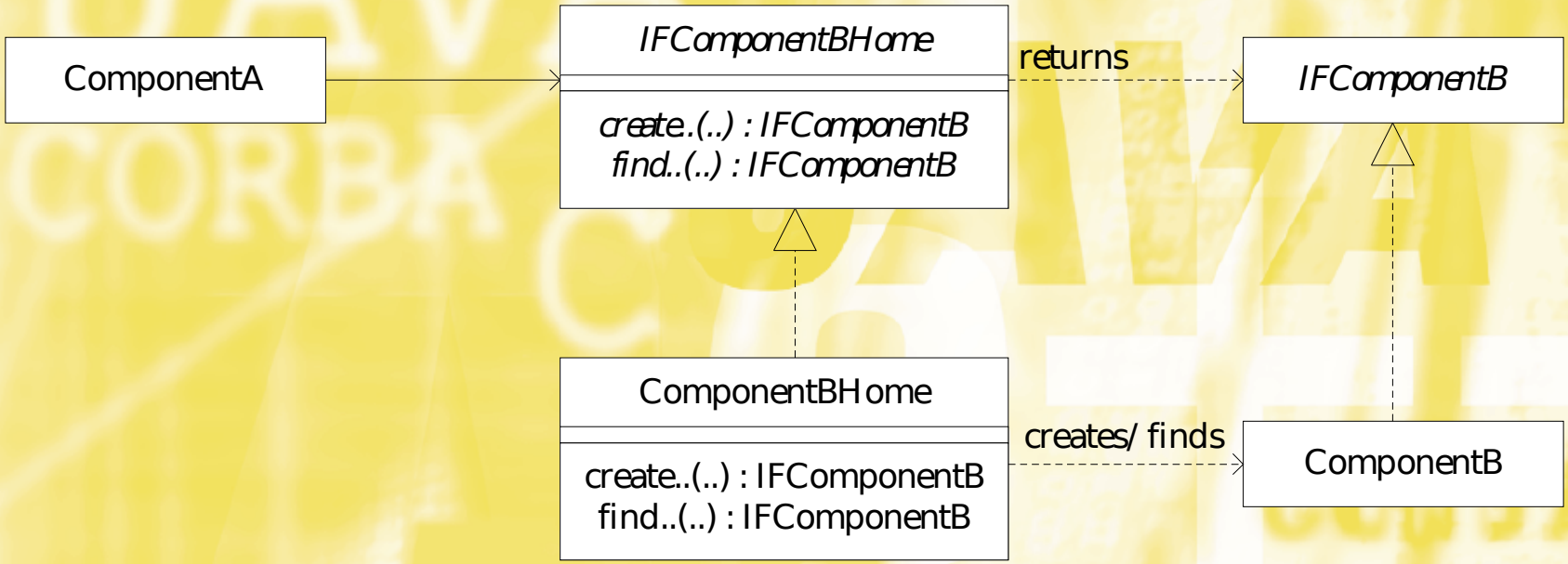
COLLABORATION



Component Home

- Context
  - Your application is assembled of collaborating, loosely coupled components, i.e. one Component uses the services of another Component.
- Problem
  - A Component needs to find (or create new) instances of other Components
  - You might not know the exact procedure to do this
  - Technical concern: Don’t want that in component code
- **Solution**
  - **For each Component, provide a management interface**
  - **It can provide different ways (operations) how to create and find component instances**

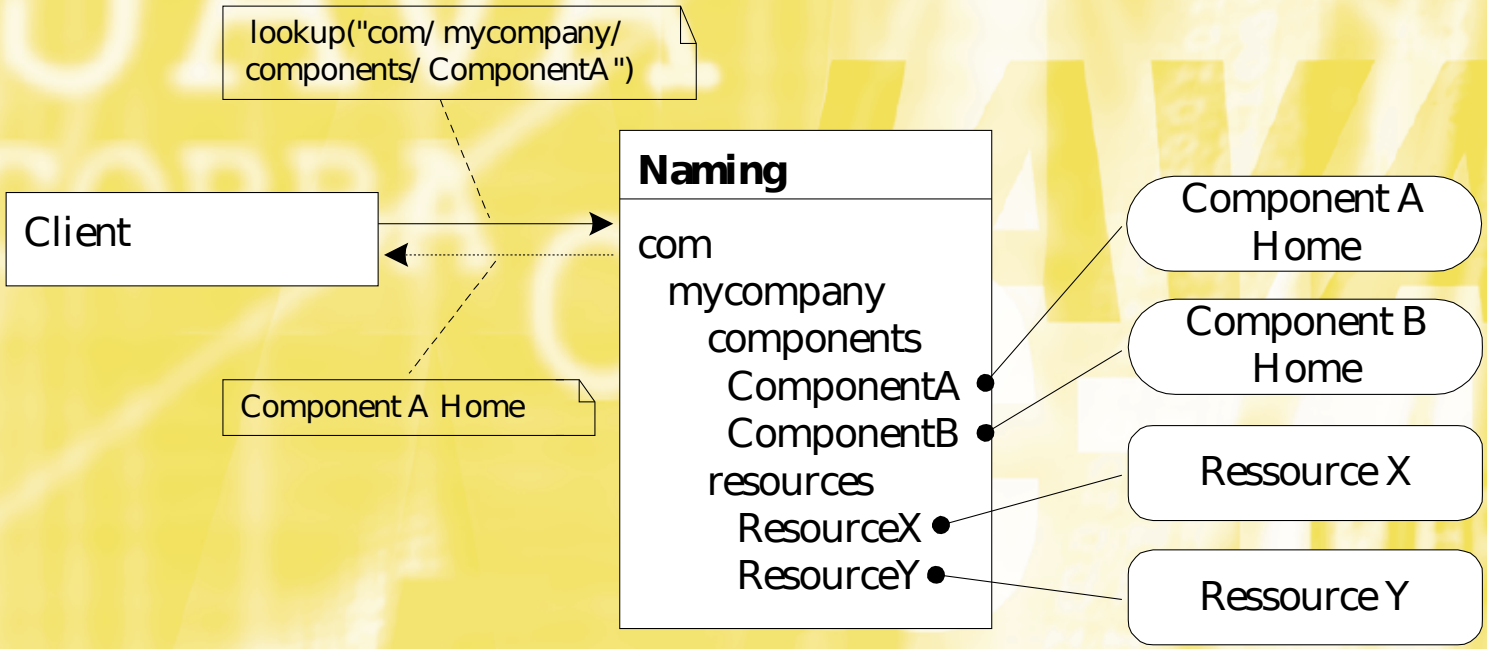
Component Home



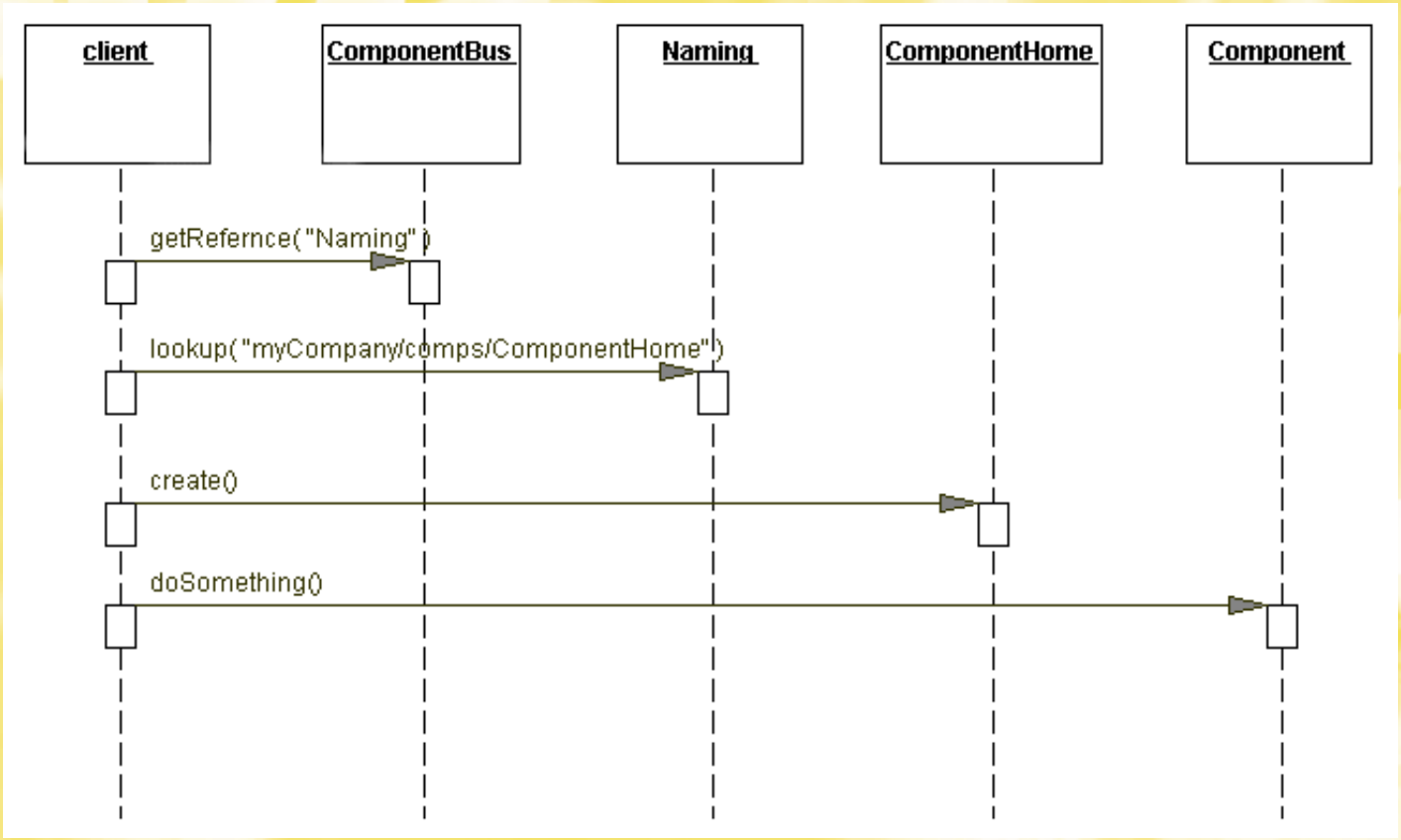
Naming

- Context
  - You have provided a Component Home for your components to manage the instances of a specific component type. You are using Managed Resources in the Container.
- Problem
  - To access the Component Home or Managed Resources, you need to get the reference of the home or of the service.
- **Solution**
  - **Provide a naming service which maps names to object references**
  - **Can be used uniformly for any kind of object/component/resource**
  - **It can be accessed by clients using a well-known object reference.**

Naming


lookup("com/ mycompany/ components/ ComponentA")
Client
Component A Home
Naming
com
mycompany
components
ComponentA
ComponentB
resources
ResourceX
ResourceY
Component A Home
Component B Home
Ressource X
Ressource Y

COLLABORATION



Annotations

- Context
  - You have implemented a Container. Programmer programs functional aspects only.
- Problem
  - Programmers will need a way to control the behavior of the components in the Container
  - You need a way to tell the Container how it should handle certain aspects
- **Solution**
  - **Allow the developer to annotate the components**
  - **The Container is free to implement the Annotations**
  - **Uses standardized frameworks and other techniques such as code generation.**

Component Installation

- Context
  - You express your technical requirements regarding a Component using Annotations and the Container provides a way to implement these.
- Problem
  - Annotations state technical concerns declaratively
  - Code required to realize these at runtime
  - Consistency has to be checked in advance
- **Solution**
  - **Include an explicit installation step for Components**
  - **Provide your Component's code and the Annotations to the Container, Container creates necessary code**

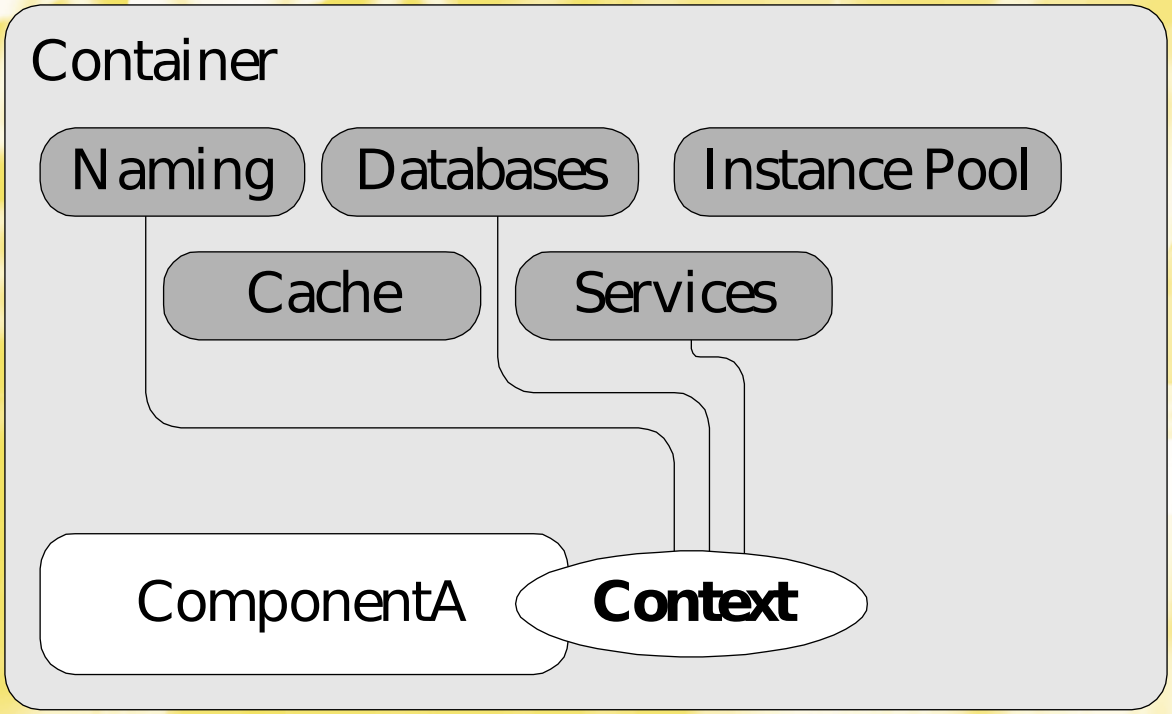
Interception

- Context
  - You have created a Container which serves as a place to live for the Components. You use Annotations to tell the Container how it should handle a Component. You use the concept of Distinguishing Identities.
- Problem
  - Container has to implement the behaviour specified in the Annotations
  - How can a Container insert specific code into prebuilt Components
- **Solution**
  - **Allow the Container to intercept any request before it reaches the destination Component**
  - **Provide a standardized interface for interceptors.**

Component Context

- Context
  - Your application is decomposed into Components which are executed in a Container.
- Problem
  - Components need to access resources outside of itself (in the Container).
  - Need to control some aspects of the Container (e.g. Tx state)
- **Solution**
  - **Supply each component instance with a Component Context at the beginning of its lifecycle**
  - **Context provides operations, with which the Component can access the environment**

Component Context



Configuration Parameters

- Context
  - You want to reuse your Components. To achieve reuse, you need a certain degree of variability in your Component implementation.
- Problem
  - you need a way to “pass (configuration) information to the Component”
  - This information must be accessible from within the Component
- **Solution**
  - **The Component Context should allow the Component to access configuration parameters**
  - **defined for the Component during Component Installation.**

Instance Pooling

- Context
  - You run your Components in a Container. You Distinguish Identities and provide Lifecycle Callbacks operations in your Component.
- Problem
  - Physical instance creation in the Container is expensive.
  - It is therefore useful to minimize the number of creations and destructions
  - especially in the case of Entity Components
- **Solution**
  - **Use instance pooling together with Lifecycle Callbacks**
  - **Keep a number of component instances ready**
  - **Let them "become" different logical instances at different times.**

Passivation

- Context
  - You run your Components in a Container. You Distinguish Identities and provide Lifecycle Callbacks operations in your Component.
- Problem
  - Session Components need to be accessible as long as a client does not destroy them
  - They might not be used for a very long time in between invocations
- **Solution**
  - **Allow the Container to remove unused Component instances temporarily from memory**
  - **Attributes are stored persistently and are reloaded upon reactivation.**

Managed Resource

- Context
  - You run your Components in a Container, and they can access parts of the world outside using a Component Context.
- Problem
  - Components will need to access several external resources
  - You do not know, how many physical component instances you will have
  - You do not want to limit the portability of your components by depending on the type or location of a specific resource
- **Solution**
  - **Let the Container manage resources**
  - **It creates pools for every configured resource**
  - **Access Managed Resources in Naming using logical names only**
  - **Let the Container do the mapping to real Naming name**

Invocation Context

- Context
  - You are using a Container to take care of the technical requirements. Among other things, the Container's job to manage transactions and security.
- Problem
  - Container needs to know more than just the name of the invoked method and the arguments when an operation is called
  - This cannot be supplied with a normal method call
- **Solution**
  - **Include an invocation context with each operation**
  - **It can be inserted or created by using Interception**
  - **The context can include any kind of information, only the Container must know how to handle it**

Thank you...

Questions, Critique ?