



Quercus Software Engineering Group
Department of Computer Science
University of Extremadura (Spain)

AspectCCM: An aspect-oriented extension of the Corba Component Model

Pedro J. Clemente (jclemente@unex.es)
Juan Hernández (juanher@unex.es)
Juan M. Murillo (juanmamu@unex.es)
Miguel A. Pérez (toledano@unex.es)
Fernando Sánchez (fernando@unex.es)

Detecting the Problems of crosscutting in CBSE (CCM context).

Define Code-Tangling and its repercussions in CBS.

The problems are focused on the composition of the final software.

Provoke the appearance of problems to reuse components.

Provoke the appearance of problems to adapt the final systems to new requirements.

We solve these problems using Aspect Oriented Programming.

We present a solution for CCM Components: AspectCCM.

The process includes all phases of component life cycle.

The process separates the aspect involved in a system and builds the final system with them.

AspectCCM deletes or decreases the appearance of crosscutting in CBS.



Introduction

Crosscutting

Finding

CBS

Problems

AOP

AspectCCM

Specification

Implementation

Package

Assembly

Conclusions

Future Works



Introduction

Crosscutting

AOP

AspectCCM

Conclusions

Future Works

CBSE has been generating wide concern in recent years.

It is very attractive for software development companies.

Building applications = Assembling independent and reusable software modules called *components*.

A **component** is a software composition unit that specifies a set of interfaces and a set of requirements.

A *component* is thought to be developed, acquired and incorporated into the system.

It can also be composed with other components independently in time and space.

This new paradigm involves the assembly/composition of prefabricated pieces of software.

Why should components be used?

It reduces the cost of software development.

It promises the flexibility, reliability and reusability of applications.

Introduction

Crosscutting

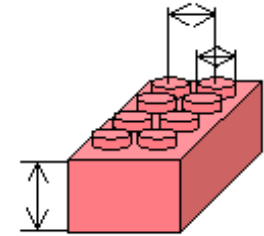
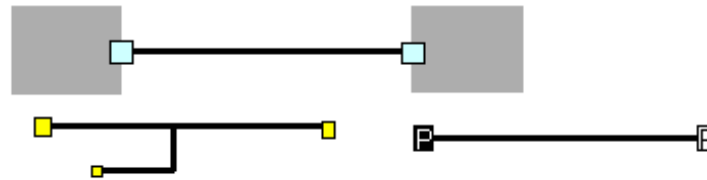
AOP

AspectCCM

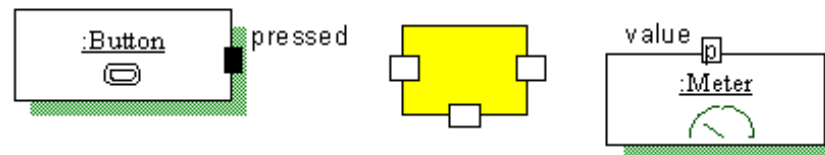
Conclusions

Future Works

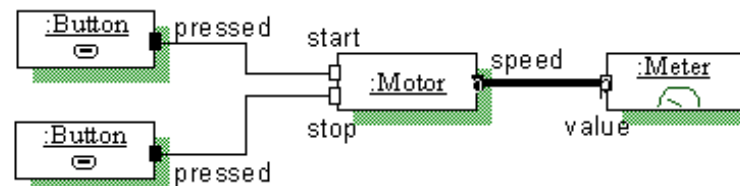
- Define *connectors* between components



- Design *components* that work in the domain



- Rapidly assemble components into *products*



Desmond D'Suoza
at Catalysis

Finding *crosscutting* in CBS (I)

Department of Computer Science
University of Extremadura(Spain)

Introduction

Crosscutting

Finding

CBS

Problems

AOP

AspectCCM

Conclusions

Future Works

During the component-based system specification, the interfaces it provides and those it requires must be described.

A component specification describes the necessities of the problem domain.

```
interface AccountManager {
    void OpenAccount(in string name, in string pass);
    long Balance(in string name, in string pass);
    void AddBalance(int string name, in string pass, in long
balance);
    void SubBalance(int string name, in string pass, in long
balance);
};
interface Autentication {
    boolean CheckLogin(in string name, in string pass);
};
```

```
component ServerBank {
    attribute string name;
    provides AccountManager for_clients;
    uses Autentication for_clients_login;
};
component ServerCheck {
    attribute string name;
    provides Autentication for_clients_check;
};
component ClientBank {
    attribute string name;
    uses AccountManager to_server;
};
```

Finding *crosscutting* in CBS (II)



Department of Computer Science
University of Extremadura (Spain)

Introduction

Crosscutting

Finding

CBS

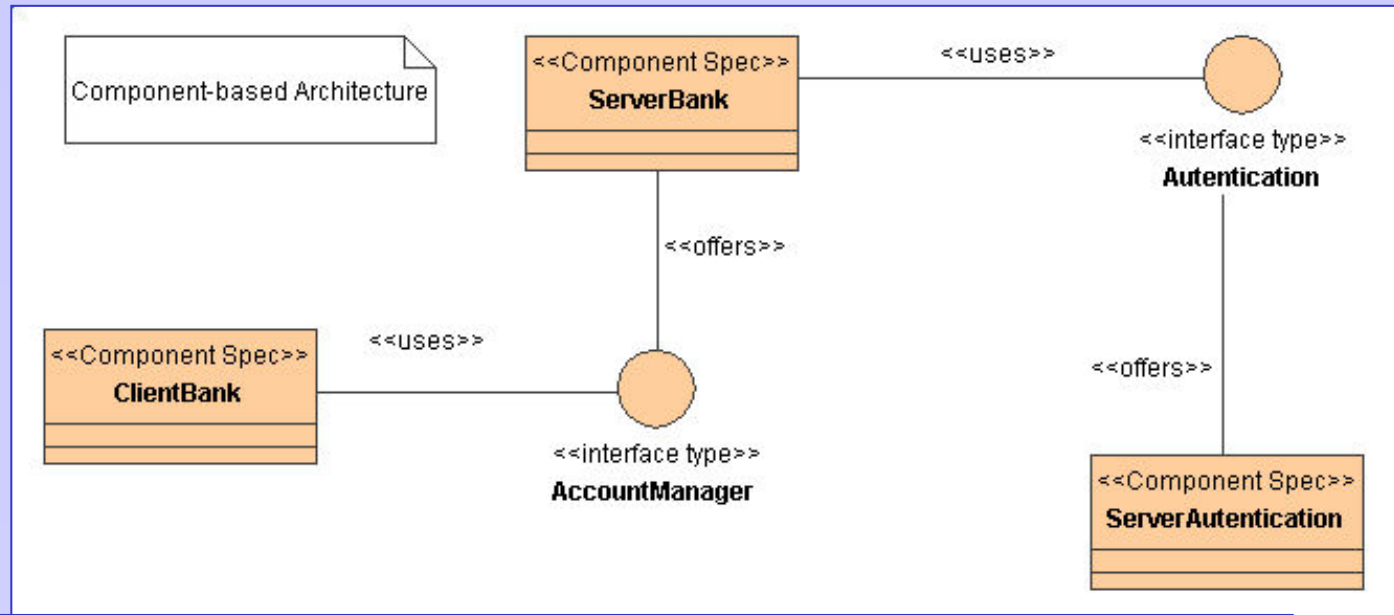
Problems

AOP

AspectCCM

Conclusions

Future Works



```

public int Balance(String name, String pass){
    Authentication for_clients_login =
        get_connection_for_clients_login();
    if(for_clients_login == null) { return 0; }
    Autorizado=for_clients_login.CheckLogin(name,pass);
    if (Autorizado==true) {
        textArea_.append("Balance of account: "+name+ "Balance: "+saldo);
        return saldo;}
    else {
        textArea_.append(" Balance of account: " +name + " non authorized");
        return -1; }
}
  
```

Crosscutting

Finding *crosscutting* in CBS (II)



Department of Computer Science
University of Extremadura(Spain)

Introduction

Crosscutting

Finding

CBS

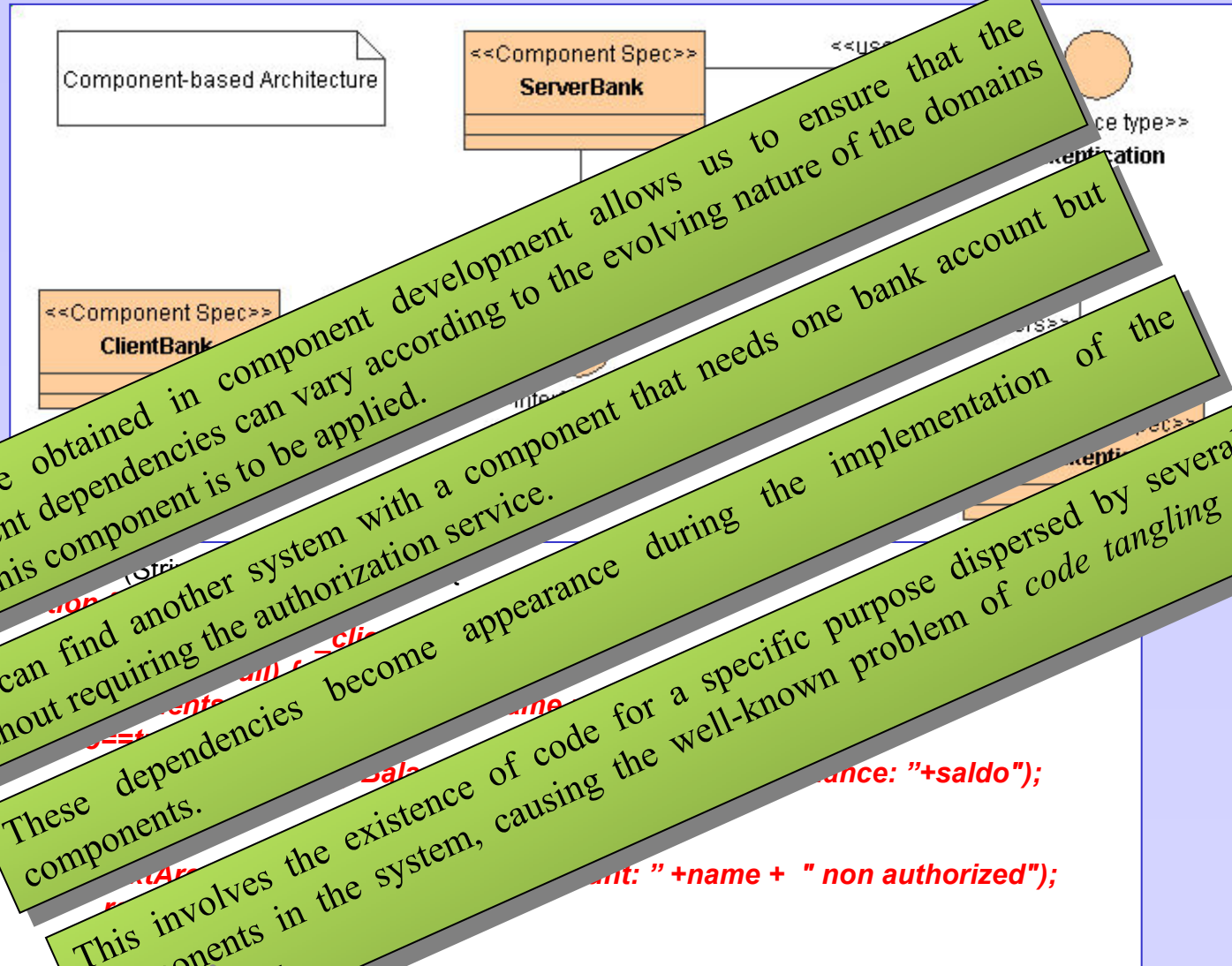
Problems

AOP

AspectCCM

Conclusions

Future Works



Experience obtained in component development allows us to ensure that the component dependencies can vary according to the evolving nature of the domains where this component is to be applied.

We can find another system with a component that needs one bank account but without requiring the authorization service.

These dependencies become appearance during the implementation of the components.

This involves the existence of code for a specific purpose dispersed by several components in the system, causing the well-known problem of *code tangling* or *crosscutting*.

Crosscutting

Finding *crosscutting* in CBS (III)



Department of Computer Science
University of Extremadura(Spain)

Introduction

Crosscutting

Finding

CBS

Problems

AOP

AspectCCM

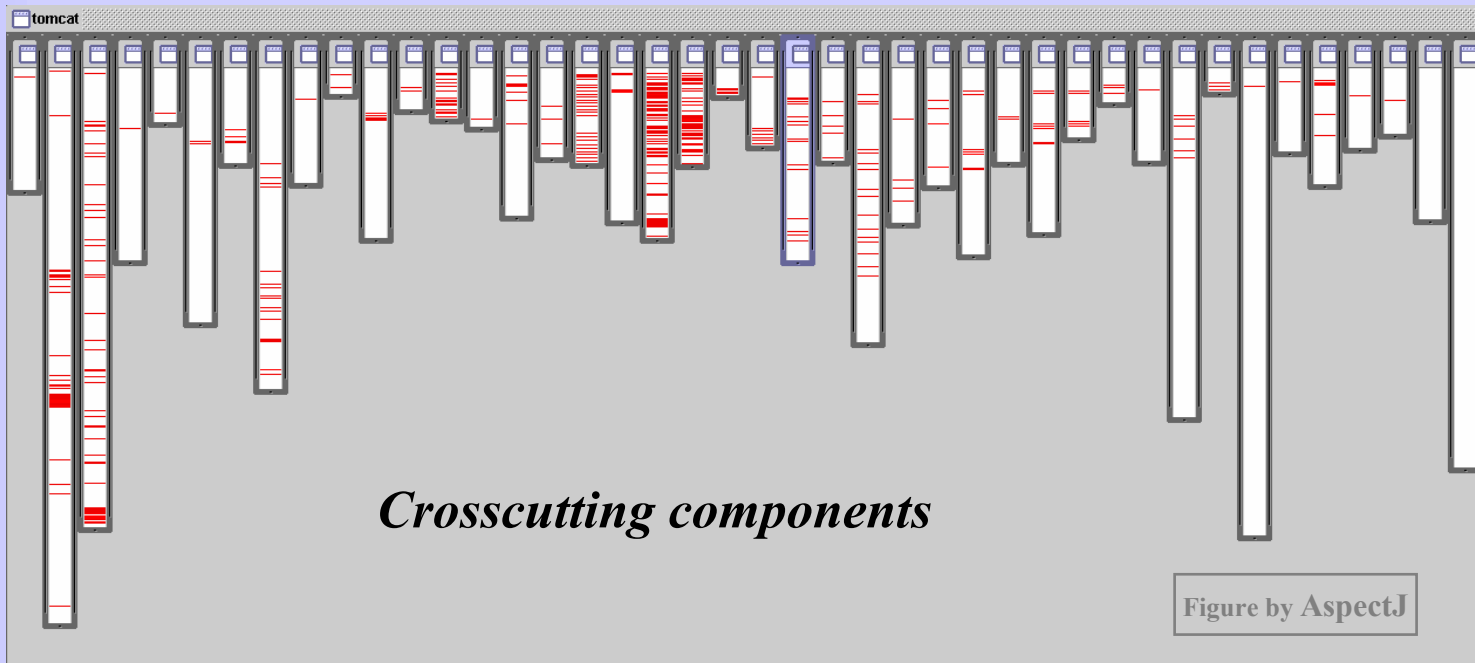
Conclusions

Future Works

This situation unmistakably supposes that any change in the component dependences involves recoding the component.

This situation of *code tangling* increases the dependences among the components.

Introduces new opportunities for making programming mistakes and, as a result, the components are less reusable, flexible and adaptable.



Introduction

Crosscutting

AOP

AspectCCM

Conclusions

Future Works

Where can we find **crosscutting** during component-based software development? .

Mainly the interactions or interconnections among components describe *crosscutting* characteristics in the components.

AOP is a technique that tries to capture and delete this crosscutting. This means separating the different aspects involved in a system into different components.

AOP improves flexibility, adaptability and reusability of the elements used to compose the final software.

AOP has already been applied with success in the areas of concurrence, coordination, replication, etc.

AOP must also provide these characteristics to Component-based Development.



Introduction

Crosscutting

AOP

AspectCCM

Specification

Implementation

Package

Assembly

Conclusions

Future Works

AspectCCM

Uses Aspect Oriented Programming to eliminate or decrease the **crosscutting** problem at the CBS

Produces a connection between two software engineering technologies like CBSE & AOP.

Covers all the phases of CCM component development.

interfaces specification

component specification

components implementation

packaging

assembly and deployment

CBSE and AOP complement each other since the solutions provided in the scope of AOP to solve the **crosscutting** problem.

Specification (I)



Department of Computer Science
University of Extremadura(Spain)

Introduction

Crosscutting

AOP

AspectCCM

Specification

Implementation

Package

Assembly

Conclusions

Future Works

The objective of component specification is to offer a useful mechanism to decide which component to use at each moment during software design.

During the component-based system specification, the interfaces it provides and those it requires must be described.

However, the dependences description causes the appearance of **crosscutting** in implementation phase.

Is it necessary to define the components dependences from their definition?

Component dependences

Intrinsic

It is defined when the description and use of this dependency is vital to make the component functional,

Non-Intrinsic

It is defined when its use depends on the framework or the context in which a component is developed.

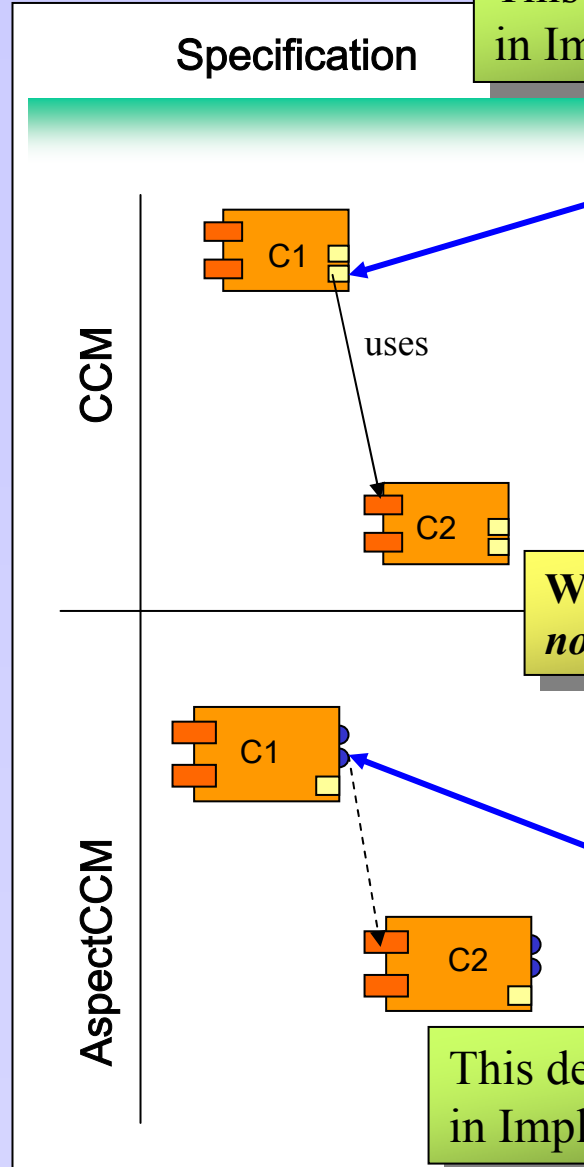
Specification (II)



Department of Computer Science
University of Extremadura(Spain)

This dependence is implemented in Implementation Phase

Define *Uses*



We can to describe *intrinsic* and *non-intrinsic* dependencies.

Define *UsesAspect* in IDL3

This dependence is not implemented in Implementation Phase

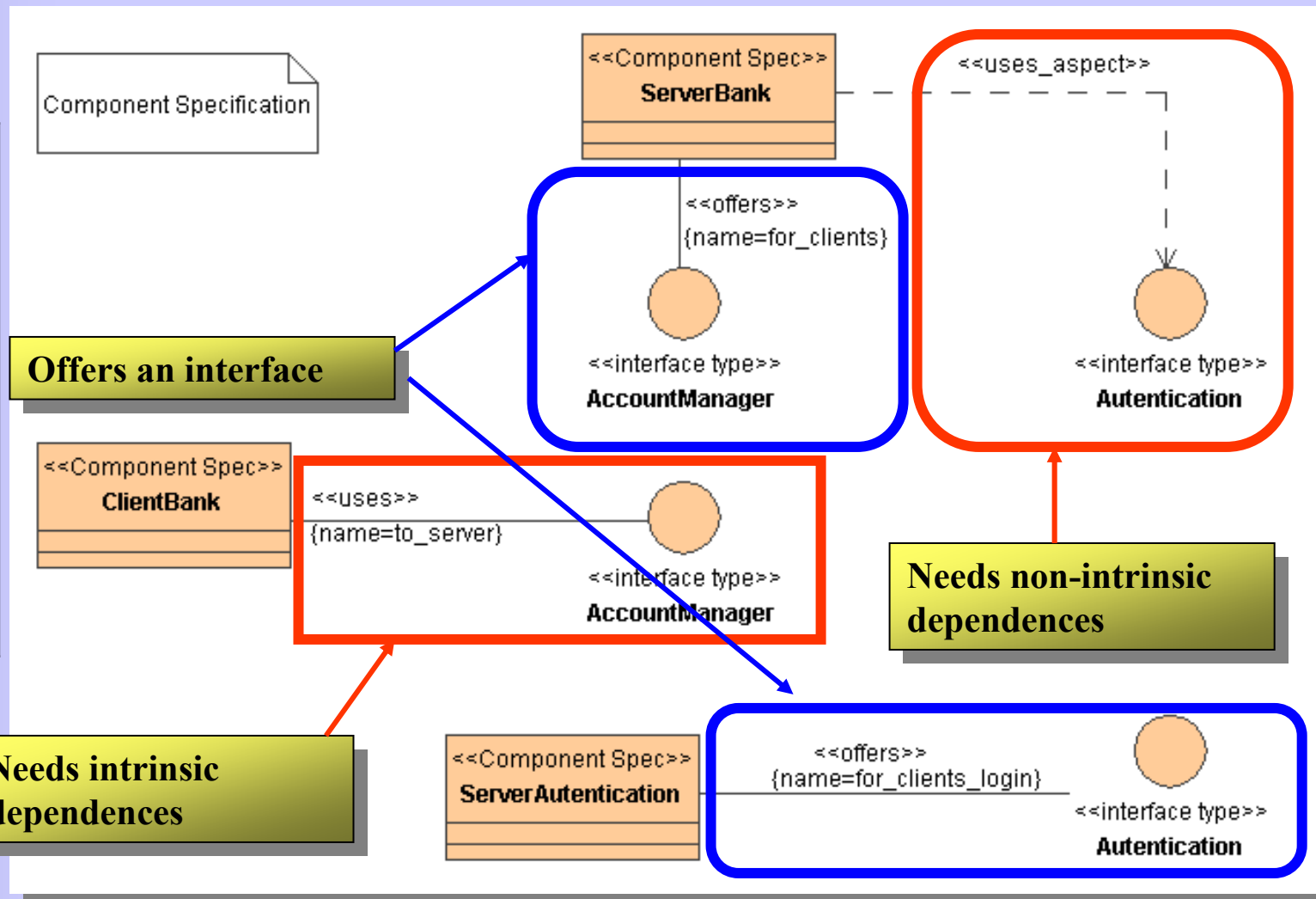
- Introduction
- Crosscutting
- AOP
- AspectCCM
- Specification
- Implementation
- Package
- Assembly
- Conclusions
- Future Works

Specification (III)



Department of Computer Science
University of Extremadura(Spain)

- Introduction
- Crosscutting
- AOP
- AspectCCM
- Specification
- Implementation
- Package
- Assembly
- Conclusions
- Future Works



Example:UML diagram for the intrinsic and non-intrinsic dependences definition



Example of component definition using IDL3 and *uses_aspects*

```

interface AccountManager {
    void OpenAccount(in string name, in string pass);
    long Balance(in string name, in string pass);
    void AddBalance(int string name, in string pass, in long balance);
    void SubBalance(int string name, in string pass, in long balance);
};

interface Autentication {
    boolean CheckLogin(in string login, in string password);
};

component ServerBank {
    attribute string name;
    provides AccountManager for _clients;
    uses_aspect Autentication;
};

component ServerCheck {
    attribute string name;
    provides Autentication for _clients_check;
};

component ClientBank {
    attribute string name;
    uses AccountManager to _server;
};
  
```

Define *UsesAspect* in IDL3 →
non-intrinsic dependences

Define *Uses* in IDL3 →
intrinsic dependences

Introduction

Crosscutting

AOP

AspectCCM

Specification

Implementation

Package

Assembly

Conclusions

Future Works

Implementation



Department of Computer Science
University of Extremadura(Spain)

Introduction

Crosscutting

AOP

AspectCCM

Specification

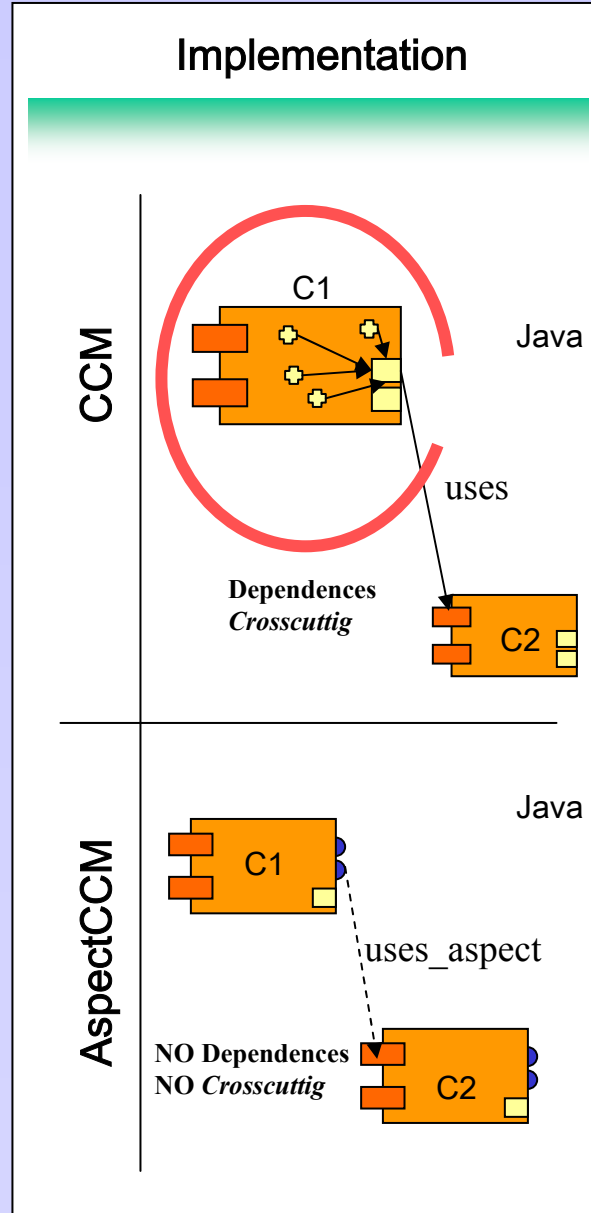
Implementation

Package

Assembly

Conclusions

Future Works



During the implementation of these methods, dependencies can be used in the component implementation through the uses clause.



Crosscutting

All those dependencies defined through the uses_aspect clause are applied throughout the design phases of software components



NO Crosscutting

Package



Department of Computer Science
University of Extremadura(Spain)

Introduction

Crosscutting

AOP

AspectCCM

Specification

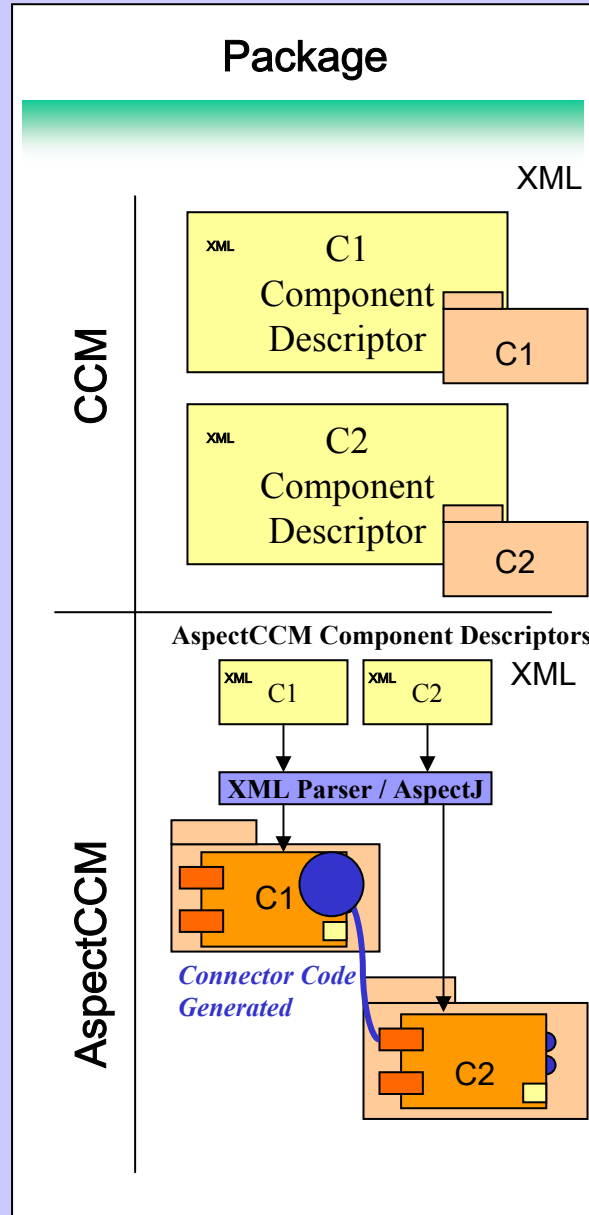
Implementation

Package

Assembly

Conclusions

Future Works



The packaging phase the most suitable to describe when and where the dependencies should be applied

This situation allow us obtain more flexible component-based system.

We extend Component Descriptor

This component description is pre-processed.

It is generated a code for aspect (AspectJ code) which interconnect the components.

Apply the restrictions and dependencies specifies in Component Descriptor

Package



Department of Computer Science
University of Extremadura(Spain)

Introduction

Crosscutting

AOP

AspectCCM

Specification

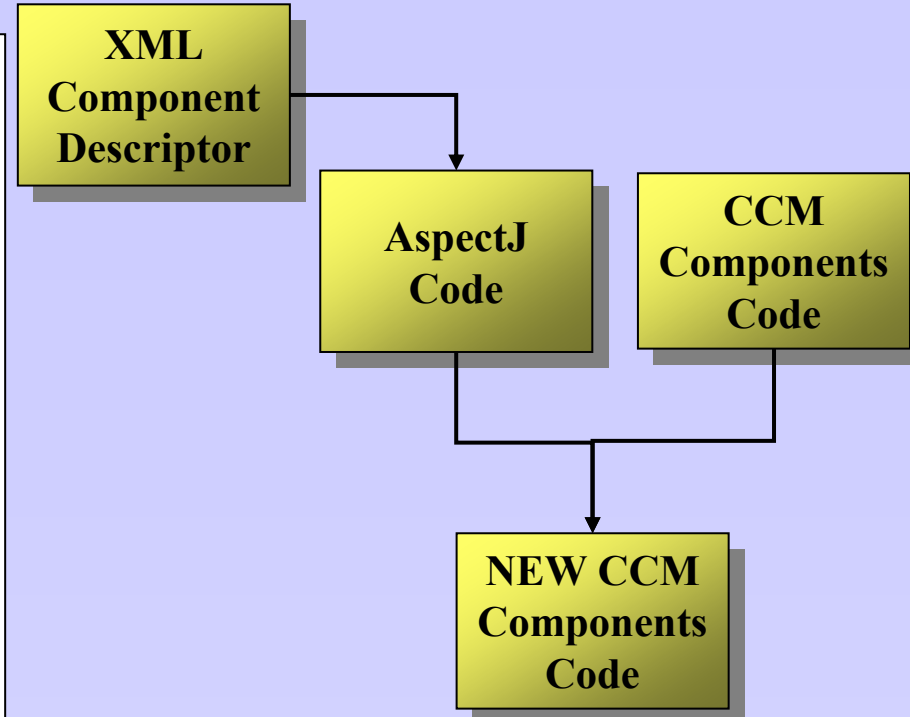
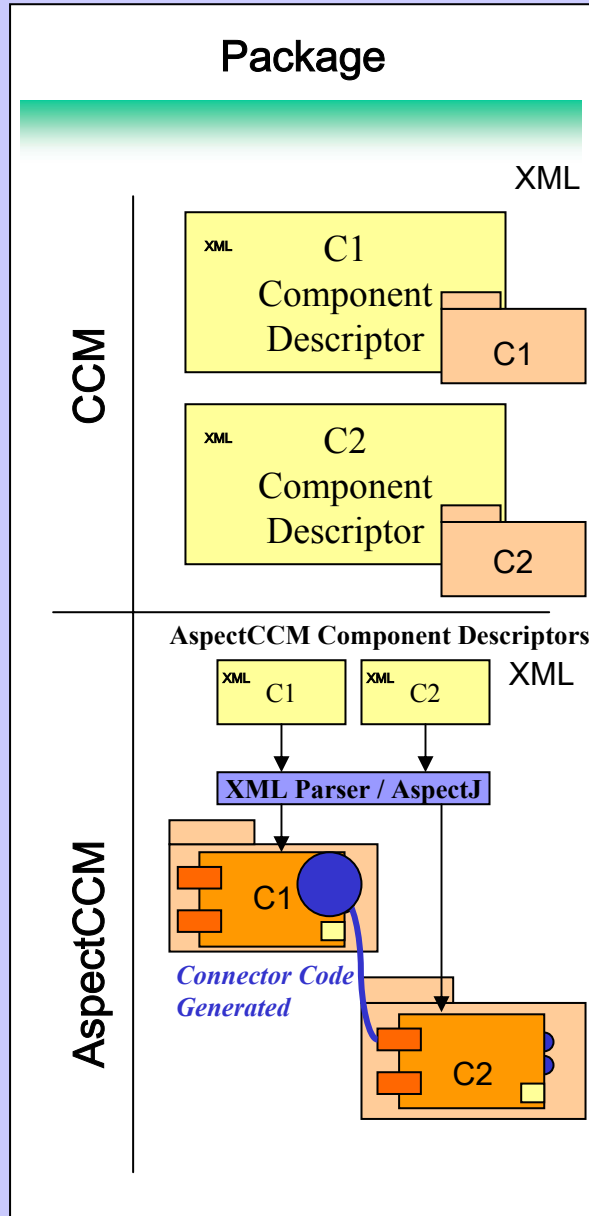
Implementation

Package

Assembly

Conclusions

Future Works



Advantages

Adaptability of the systems

Reusability of the components

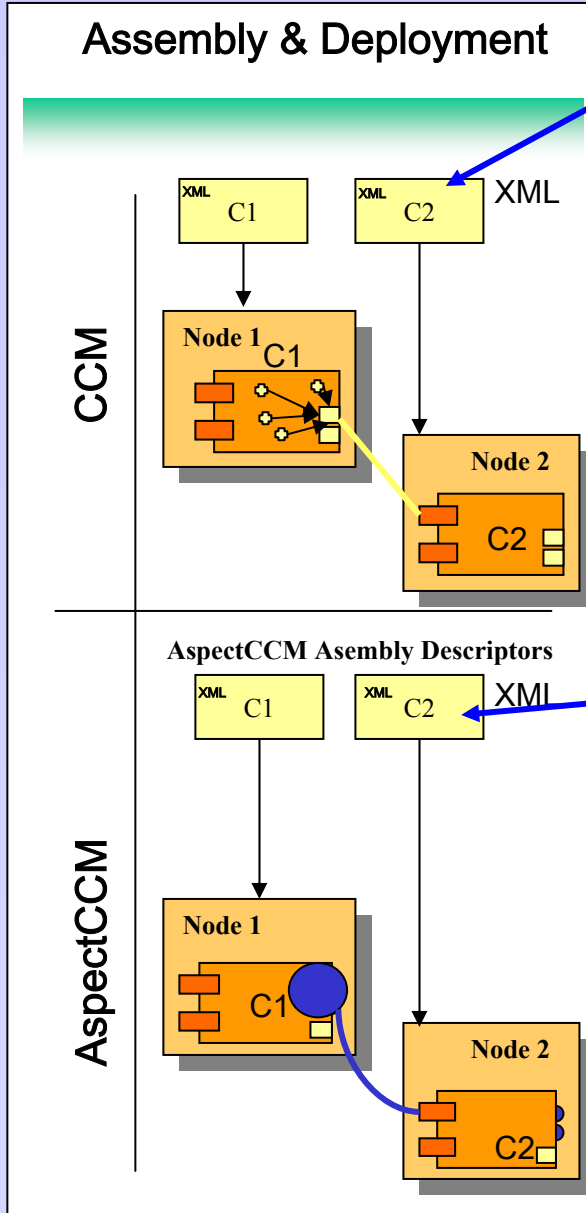
Change the requirements, new components and not recode these components

Assembly & Deployment



Department of Computer Science
University of Extremadura(Spain)

- Introduction
- Crosscutting
- AOP
- AspectCCM
- Specification
- Implementation
- Package
- Assembly
- Conclusions
- Future Works



The descriptor specifies the location of the components and their instances

The components can be assembled along various machines

We extend the Assembly Descriptor with a new directive *connectaspects*

Similar to *connections*

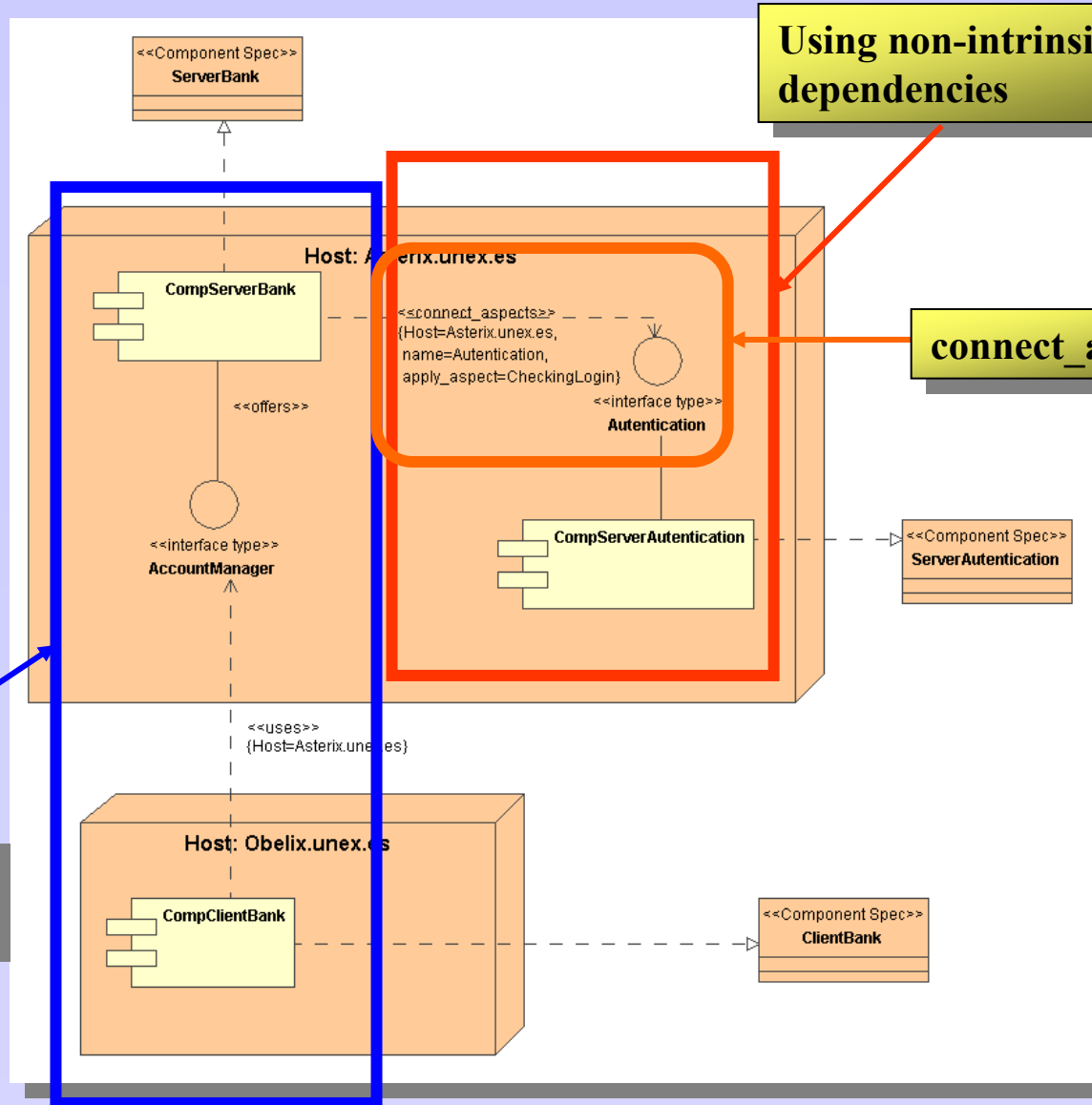
Integrated the non-intrinsic dependences & components locations

Assembly & Deployment



Department of Computer Science
University of Extremadura(Spain)

- Introduction
- Crosscutting
- AOP
- AspectCCM
- Specification
- Implementation
- Package
- Assembly
- Conclusions
- Future Works



Using non-intrinsic dependencies

connect_aspects

Using intrinsic dependencies

Assembly and deployment description of a component based system



Introduction

Crosscutting

AOP

AspectCCM

Conclusions

Future Works

We have presented a joined CBSE and AOP proposal in which two of the recent tendencies in software system development are united: AspectCCM.

We have expanded the life cycle of a CCM component-based system through techniques of aspect-oriented programming

For it we have detached every one of the stages in the CCM component development.

Dependences description model among components is expanded (intrinsic and non-intrinsic dependences)

AspectCCM reduces the appearance of crosscutting in the interconnection of component that form the final systems.

AspectCCM increases the component reusability and systems adaptability.



Introduction

Crosscutting

AOP

AspectCCM

Conclusions

Future Works

We are working on the CCM implementation carried out by the GOAL group. We want to extend our proposal to others CCM implementations.

Optimize XML descriptors and the AspectJ code generation .

We are working in a system that it would adapt the interfaces component during interconnection process.

**Thanks for your
attention**



Quercus Software Engineering Group
Department of Computer Science
University of Extremadura (Spain)

AspectCCM: An aspect-oriented extension of the Corba Component Model

Presented by **Pedro José Clemente Martín**