A Metamodel for Modelling of Component-Based Systems with Mobile Architecture

Marek Rychlý

Department of Information Systems
Faculty of Information Technology
Brno University of Technology
(Czech Republic)

19^{th} International Conference
on Information Systems Development,
August 25–27, 2010

Outline

1. Introduction
   - Component-Based Development (CBD)
   - State of the Art and Motivation
2. A Metamodel for Modelling of CBS with Mobile Architecture
   - A Four-layer Modelling Architecture and (E)MOF
   - The Metamodel – Components and Interfaces
   - An Example of a CBS Model and Behavioural Description
3. Summary and Future Work

Obsah

1. Introduction
   - Component-Based Development (CBD)
   - State of the Art and Motivation
2. A Metamodel for Modelling of CBS with Mobile Architecture
   - A Four-layer Modelling Architecture and (E)MOF
   - The Metamodel – Components and Interfaces
   - An Example of a CBS Model and Behavioural Description
3. Summary and Future Work

Component-Based Development (CBD)

Definition (Software Component)

A software component is a unit of composition with contractually specified **interfaces** and explicit **context dependencies** only. It can be deployed independently and is **subject to composition** by third parties.

[Clemens Szyperski, Component Software: . . . , 2002]

1. **Primitive components**
2. **Composite components**

The architecture of component-based system can evolve:

1. **Static architectures**
2. **Dynamic architectures**
3. **Mobile architectures**

Component-Based Development (CBD)

Definition (Software Component)

A software component is a unit of composition with contractually specified **interfaces** and explicit **context dependencies** only. It can be deployed independently and is **subject to composition** by third parties.

[Clemens Szyperski, Component Software: . . . , 2002]

1. **Primitive components**
   (realised directly, beyond the scope of architecture description)
2. **Composite components**

The architecture of component-based system can evolve:

1. **Static architectures**
2. **Dynamic architectures**
3. **Mobile architectures**

Component-Based Development (CBD)

Definition (Software Component)

A software component is a unit of composition with contractually specified **interfaces** and explicit **context dependencies** only. It can be deployed independently and is **subject to composition** by third parties.
[Clemens Szyperski, Component Software: . . . , 2002]

1. **Primitive components**
2. **Composite components**
(decomposable on systems of subcomponents at the lower level)

The architecture of component-based system can evolve:

1. **Static architectures**
2. **Dynamic architectures**
3. **Mobile architectures**

Component-Based Development (CBD)

Definition (Software Component)

A software component is a unit of composition with contractually specified **interfaces** and explicit **context dependencies** only. It can be deployed independently and is **subject to composition** by third parties.

[Clemens Szyperski, Component Software: . . . , 2002]

1. **Primitive components**
2. **Composite components**

The architecture of component-based system can evolve:

1. Static architectures
2. Dynamic architectures
3. Mobile architectures

Component-Based Development (CBD)

Definition (Software Component)

A software component is a unit of composition with contractually specified **interfaces** and explicit **context dependencies** only. It can be deployed independently and is **subject to composition** by third parties.

[Clemens Szyperski, Component Software: . . . , 2002]

1. **Primitive components**
2. **Composite components**

The architecture of component-based system can evolve:

1. **Static architectures**
   (they do not evolve, a fixed structure described at a design-time;
   e.g. component models Wright, Darwin, ACME language, UML diagrams)
2. **Dynamic architectures**
3. **Mobile architectures**

Component-Based Development (CBD)

Definition (Software Component)

A software component is a unit of composition with contractually specified **interfaces** and explicit **context dependencies** only. It can be deployed independently and is **subject to composition** by third parties.

[Clemens Szyperski, Component Software: . . . , 2002]

1. **Primitive components**
2. **Composite components**

The architecture of component-based system can evolve:

1. **Static architectures**
2. **Dynamic architectures**
   (they can evolve at a system's run-time, but it is described at design-time; components can be created, removed, reconfigured, etc.;
   e.g. component models Fractal and SOFA)
3. **Mobile architectures**

Component-Based Development (CBD)

Definition (Software Component)

A software component is a unit of composition with contractually specified **interfaces** and explicit **context dependencies** only. It can be deployed independently and is **subject to composition** by third parties.

[Clemens Szyperski, Component Software: . . . , 2002]

1. **Primitive components**
2. **Composite components**

The architecture of component-based system can evolve:

1. **Static architectures**
2. **Dynamic architectures**
3. **Mobile architectures**
   (components can move at the run-time, according to functional requirements; e.g. component model SOFA 2.0)

State of the Art and Motivation

- Component models do not directly address component mobility.
  (the mobility should be utilised in basic operations, e.g. for binding of components' interfaces; it should not be a "special" feature, as in SOFA 2.0 component model)
- Components are strictly isolated from their controllers.
  (functional operations can not fire control operations and initiate reconfiguration)
- Mobility is not supported by formal bases and related models.
  (insufficient integration of formal bases and models that usually do not consider component mobility; e.g. in Wright/CSP, Darwin/Tracta, Fractal/Fractive, etc.)

State of the Art and Motivation

- Component models do not directly address component mobility.
  (the mobility should be utilised in basic operations, e.g. for binding of components' interfaces; it should not be a "special" feature, as in SOFA 2.0 component model)
- Components are strictly isolated from their controllers.
  (functional operations can not fire control operations and initiate reconfiguration)
- Mobility is not supported by formal bases and related models.
  (insufficient integration of formal bases and models that usually do not consider component mobility; e.g. in Wright/CSP, Darwin/Tracta, Fractal/Fractive, etc.)

State of the Art and Motivation

- Component models do not directly address component mobility.
  (the mobility should be utilised in basic operations, e.g. for binding of components' interfaces; it should not be a “special” feature, as in SOFA 2.0 component model)
- Components are strictly isolated from their controllers.
  (functional operations can not fire control operations and initiate reconfiguration)
- Mobility is not supported by formal bases and related models.
  (insufficient integration of formal bases and models that usually do not consider component mobility; e.g. in Wright/CSP, Darwin/Tracta, Fractal/Fractive, etc.)

Component Model Requirements

General requirements:

- functional interfaces can be (re)bound via control interfaces,
- mobile components can be moved into different contexts,
- (composite) components can change their functionality.

Three types of **component interfaces**:

1. functional interfaces,
2. control interfaces,
3. referring interfaces.

. . . .

The component model can be presented in two views:

logical view	×	**process view**
(metamodel)		(behavioural description)

Component Model Requirements

General requirements:

- functional interfaces can be (re)bound via control interfaces,
- mobile components can be moved into different contexts,
- (composite) components can change their functionality.

Three types of **component interfaces**:

1. functional interfaces,
2. control interfaces,
3. referring interfaces.

The component model can be presented in two views:

logical view (metamodel) × **process view** (behavioural description)

Component Model Requirements

General requirements:

- functional interfaces can be (re)bound via control interfaces,
- mobile components can be moved into different contexts,
- (composite) components can change their functionality.

Three types of **component interfaces**:

1. functional interfaces,
2. control interfaces,
3. referring interfaces.

The component model can be presented in two views:

logical view (metamodel) × **process view** (behavioural description)

Component Model Requirements

General requirements:

- functional interfaces can be (re)bound via control interfaces,
- mobile components can be moved into different contexts,
- (composite) components can change their functionality.

Three types of **component interfaces**:

1. functional interfaces,
   (providing and requiring a component's functional "business" operations)
2. control interfaces,
3. referring interfaces.

The component model can be presented in two views:

logical view (metamodel) × **process view** (behavioural description)

Component Model Requirements

General requirements:

- functional interfaces can be (re)bound via control interfaces,
- mobile components can be moved into different contexts,
- (composite) components can change their functionality.

Three types of **component interfaces**:

1. functional interfaces,
2. control interfaces,
   (operations for a component's life-cycle, binding of its interfaces, and its mobility)
3. referring interfaces.

The component model can be presented in two views:

logical view (metamodel) × **process view** (behavioural description)

Component Model Requirements

General requirements:

- functional interfaces can be (re)bound via control interfaces,
- mobile components can be moved into different contexts,
- (composite) components can change their functionality.

Three types of **component interfaces**:

1. functional interfaces,
2. control interfaces,
3. referring interfaces.
   (able to pass references of provided functional interfaces or whole components)

The component model can be presented in two views:

logical view × **process view**
(metamodel) (behavioural description)

Component Model Requirements

General requirements:

- functional interfaces can be (re)bound via control interfaces,
- mobile components can be moved into different contexts,
- (composite) components can change their functionality.

Three types of **component interfaces**:

1. functional interfaces,
2. control interfaces,
3. referring interfaces.

. . . .

The component model can be presented in two views:

logical view	$\times$	**process view**
(metamodel)		(behavioural description)

Component Model Requirements

General requirements:

- functional interfaces can be (re)bound via control interfaces,
- mobile components can be moved into different contexts,
- (composite) components can change their functionality.

Three types of **component interfaces**:

1. functional interfaces,
2. control interfaces,
3. referring interfaces.

. . . .

The component model can be presented in two views:

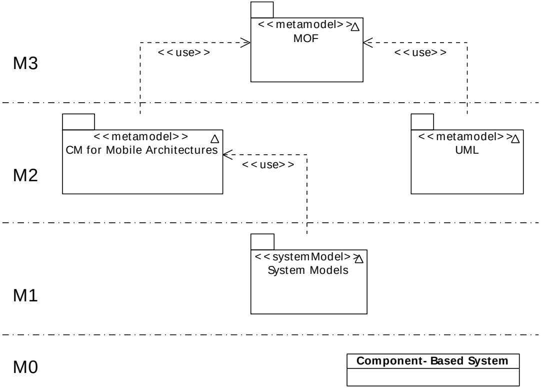
logical view	$\times$	**process view**
(metamodel)		(behavioural description)

Obsah

1. Introduction
   - Component-Based Development (CBD)
   - State of the Art and Motivation
2. A Metamodel for Modelling of CBS with Mobile Architecture
   - A Four-layer Modelling Architecture and (E)MOF
   - The Metamodel – Components and Interfaces
   - An Example of a CBS Model and Behavioural Description
3. Summary and Future Work

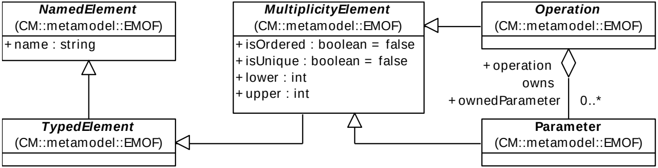
A Four-layer Modelling Architecture and (E)MOF I

The component model is described as **a metamodel in layer M2** of a four-layer modelling architecture, using OMG's Meta Object Facility.



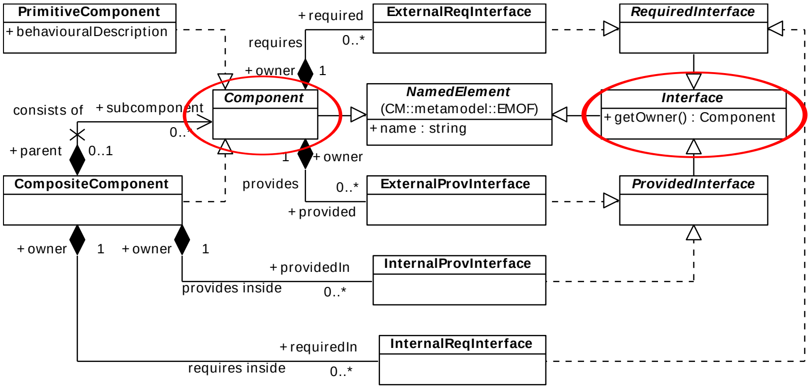
A Four-layer Modelling Architecture and (E)MOF II

To reuse well-established concepts of MOF, the component model's **metamodel extends Essential MOF** (EMOF) classes.



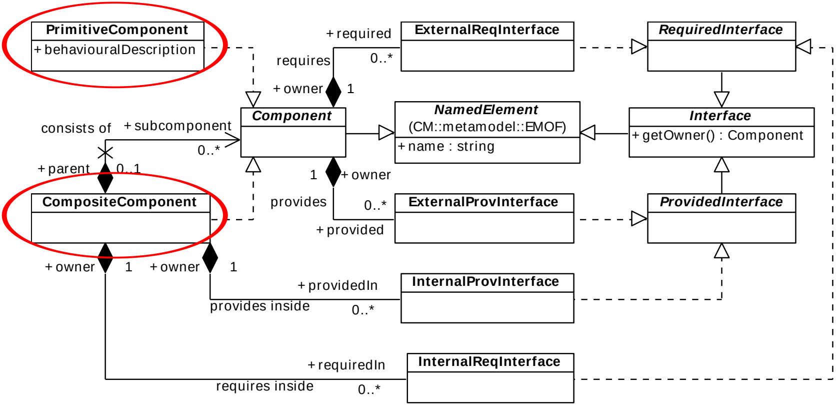
- All classes of the metamodel inherits (directly or indirectly) from class "EMOF::NamedElement".
- Interfaces of components are typed by "EMOF::TypedElement" and composed of operations "EMOF::Operation".

The Metamodel – Components and their Interfaces



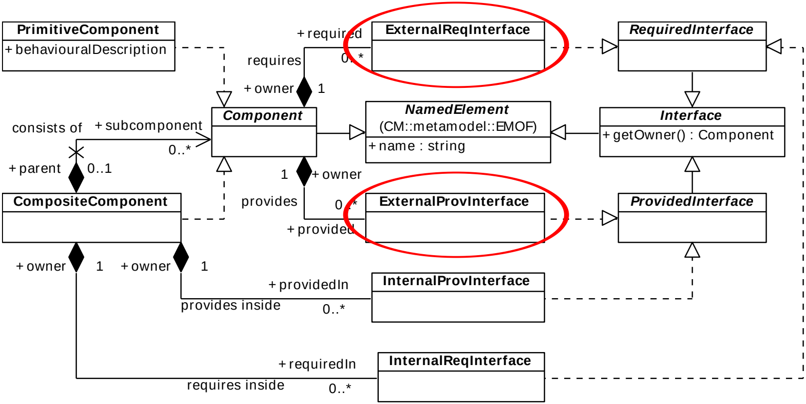
A component is represented by abstract class “Component”.
An interface is represented by abstract class “Interface”.

The Metamodel – Components and their Interfaces



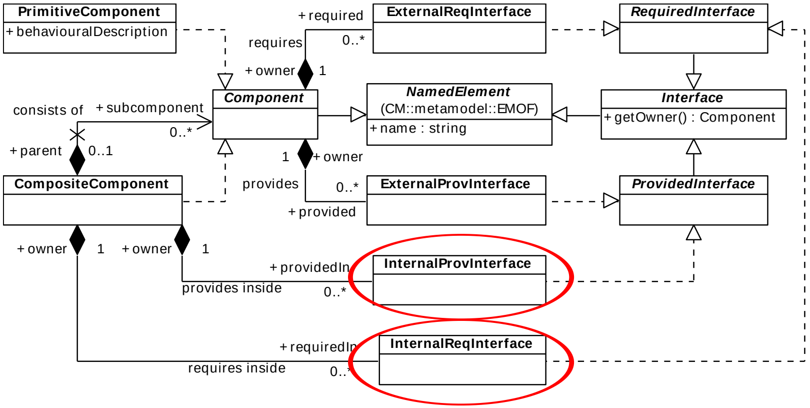
The component realisation is primitive (class "PrimitiveComponent") or composite (class "CompositeComponent"), which is composed of its subcomponents.

The Metamodel – Components and their Interfaces



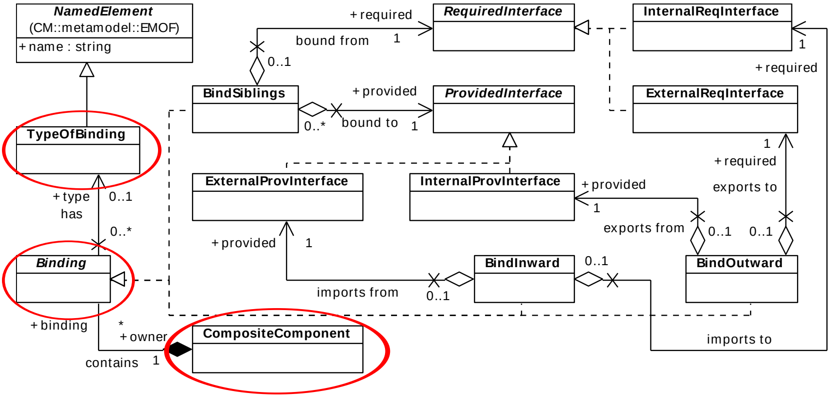
Each component can have several external required and provided interfaces (classes “ExternalReqInterface” and “ExternalProvInterface”, respectively).

The Metamodel – Components and their Interfaces



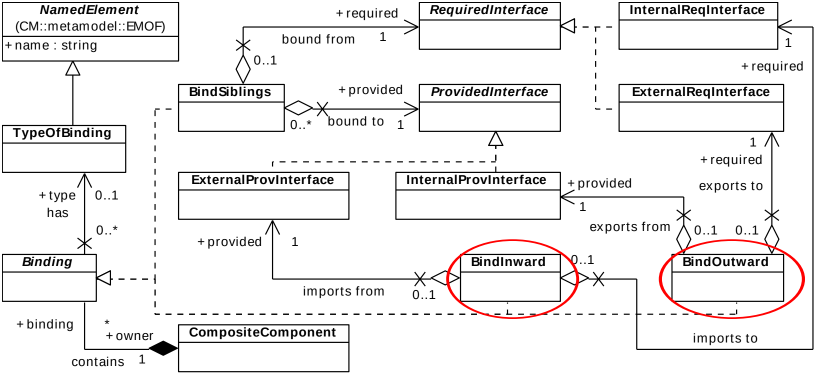
Moreover, composite components can have internal required and provided interfaces (classes “InternalReqInterface” and “InternalProvInterface”, respectively).

The Metamodel – Interfaces and their Bindings



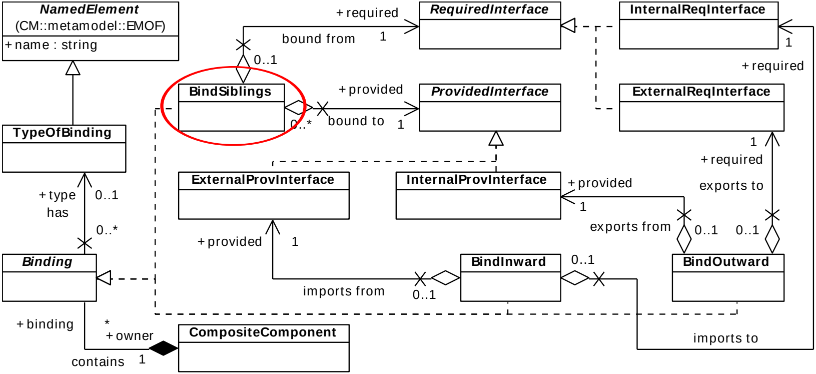
Each binding, represented by abstract class “Binding”, is owned by a composite component (class “CompositeComponent”) and typed by class “TypeOfBinding”.

The Metamodel – Interfaces and their Bindings



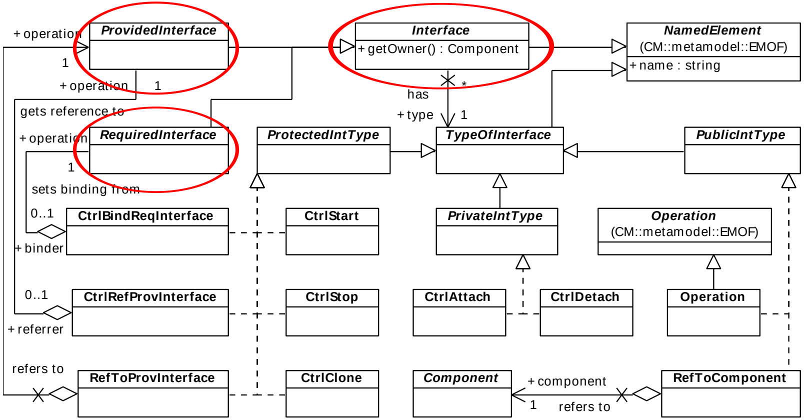
Interfaces can be bound to import (class “BindInward”) or export (class “BindOutward”) functionality into or out of a composite component, respectively, across its boundary.

The Metamodel – Interfaces and their Bindings



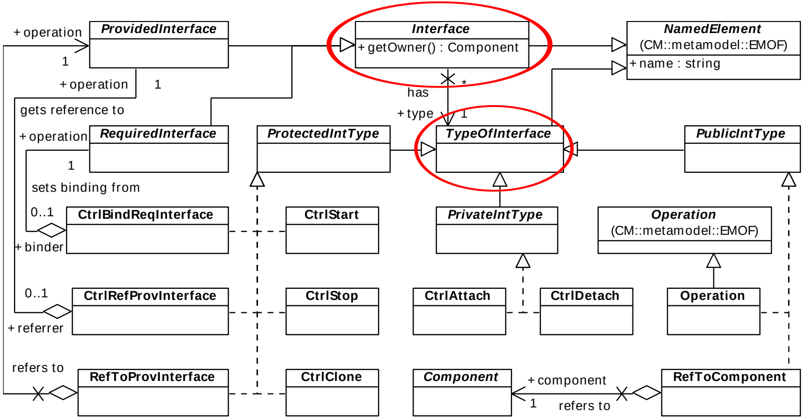
Interfaces of neighbouring components (the subcomponents of the same composite component) can be bound by class “BindSiblings”.

The Metamodel – Types of the Interfaces, Mobility



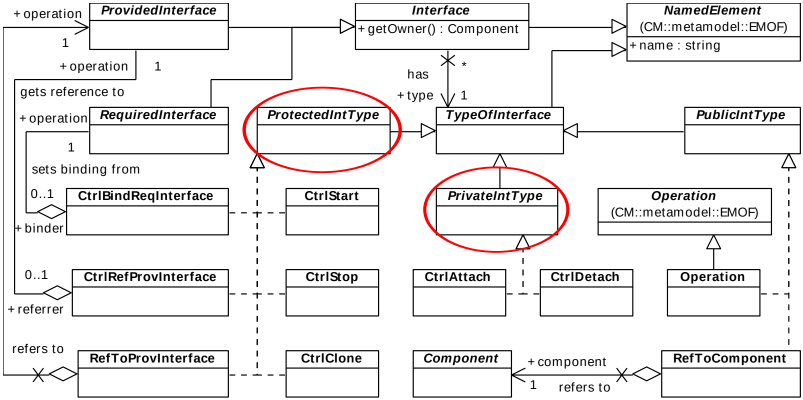
Interfaces can be realised as provided (class “ProvidedInterface”) or required interfaces (class “RequiredInterface”), according their roles in their bindings.

The Metamodel – Types of the Interfaces, Mobility



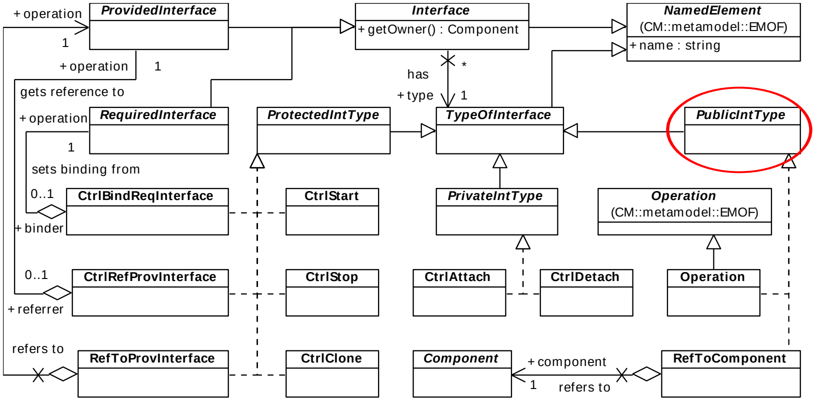
Moreover, the interfaces are typed by class “TypeOfInterface”), which describe functionality provided or required by them.

The Metamodel – Types of the Interfaces, Mobility



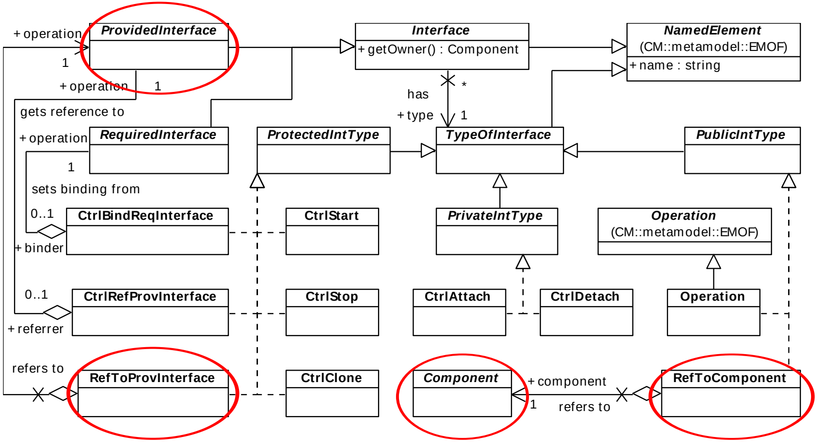
Private interfaces (class "PrivateIntType") are provided by a composite component to its subcomponents only. Together with protected interfaces (class "ProtectedIntType"), they can not be referred (i.e. used outside of their parent components).

The Metamodel – Types of the Interfaces, Mobility



Public interfaces (class "PublicIntType") can be used freely and pass across components boundaries. They support component mobility.

The Metamodel – Types of the Interfaces, Mobility



Interfaces of types represented by "RefToProvInterface" and "RefToComponent" are used to pass references to provided interfaces and components, respectively.

An Example of a Component-Based System Model

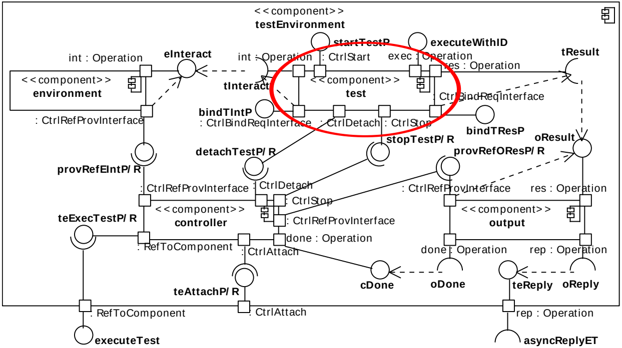
- The metamodel and UML are based on the same meta-metamodel MOF, although they are distinct in purpose and also in practice.
- However, we can utilise the notation of UML 2 component diagrams.

An Example of a Component-Based System Model

- The metamodel and UML are based on the same meta-metamodel MOF, although they are distinct in purpose and also in practice.
- However, we can utilise the notation of UML 2 component diagrams.

An Example of a Component-Based System Model

- The metamodel and UML are based on the same meta-metamodel MOF, although they are distinct in purpose and also in practice.
- However, we can utilise the notation of UML 2 component diagrams.



Component "testEnvironment" is able to receive component "test" (a test script) and to attach it as its sub-component via component "controller".

Behavioural Description of CBSs

- The component model's metamodel can describe a specific configuration of an architecture.
- To describe the architecture's evolution, we need the process view and appropriate formalism.

logical view	$\times$	**process view**
(metamodel)		(behavioural description)

- We use the polyadic π-calculus and a component's behaviour is described as a single π-calculus process abstraction.

Behavioural Description of CBSs

- The component model's metamodel can describe a specific configuration of an architecture.
- To describe the architecture's evolution, we need the process view and appropriate formalism.

logical view	$\times$	**process view**
(metamodel)		(behavioural description)

- We use the polyadic π-calculus and a component's behaviour is described as a single π-calculus process abstraction.

Behavioural Description of CBSs

- The component model's metamodel can describe a specific configuration of an architecture.
- To describe the architecture's evolution, we need the process view and appropriate formalism.

logical view	$\times$	**process view**
(metamodel)		(behavioural description)

- We use the polyadic π-calculus and a component's behaviour is described as a single π-calculus process abstraction.

Behaviour of a Component from the Example

Behaviour of interface references and binding, import and export, control of the component's life-cycle, and core behaviour of composite component "testEnvironment":

$$\begin{aligned}
TE_{comp} \overset{def}{=}\ & (s_0, s_1, p^g_{executeTest}, p^s_{asyncRepltET}).(p_{executeTest}, r_{teExecTest},\\
& p^s_{teExecTest}, r_{asyncRepltET}, p_{teReply}, p^g_{teReply}, p_{teAttach})\\
& (Ctrl_{Ifs}\langle p_{executeTest}, p^g_{executeTest}\rangle \mid Ctrl_{Ifs}\langle r_{teExecTest}, p^s_{teExecTest}\rangle\\
& \mid Ctrl_{Ifs}\langle r_{asyncRepltET}, p^s_{asyncRepltET}\rangle \mid Ctrl_{Ifs}\langle p_{teReply}, p^g_{teReply}\rangle\\
& \mid Ctrl_{EI}\langle p_{executeTest}, r_{teExecTest}\rangle \mid Ctrl_{EI}\langle p_{teReply}, r_{asyncRepltET}\rangle\\
& \mid Ctrl_{SS}\langle s_0, s_1, p_{teAttach}\rangle \mid TE'_{comp}\langle p_{teAttach}, p^s_{teExecTest}, p^g_{teReply}\rangle)\\
TE'_{comp} \overset{def}{=}\ & (p_{teAttach}, p^s_{teExecTest}, p^g_{teReply}).\ldots\\
& (Ctr\langle s^{ctr}_0, s^{ctr}_1, p^g_{cDone}, p^g_{teExecTest}, r_{teAttach}, r_{detachTest}, r_{stopTest},\\
& \quad r_{provRefElnt}, r_{provRefORes}\rangle \mid Env\langle s^{env}_0, s^{env}_1, p^g_{eInteract}\rangle\\
& \mid Out\langle s^{out}_0, s^{out}_1, p^g_{oResult}, p^s_{oDone}, p^s_{oReply}\rangle \mid \ldots)
\end{aligned}$$

Behaviour of a Component from the Example

Behaviour of interface references and binding, import and export, control of the component's life-cycle, and core behaviour of composite component "testEnvironment":

$$
\begin{aligned}
TE_{comp} \stackrel{def}{=}\ & (s_0, s_1, p^g_{executeTest}, p^s_{asyncRepltET}).(p_{executeTest}, r_{teExecTest},\\
& p^s_{teExecTest}, r_{asyncRepltET}, p_{teReply}, p^g_{teReply}, p_{teAttach})\\
& (Ctrl_{Ifs}\langle p_{executeTest}, p^g_{executeTest}\rangle \mid Ctrl_{Ifs}\langle r_{teExecTest}, p^s_{teExecTest}\rangle\\
& \mid Ctrl_{Ifs}\langle r_{asyncRepltET}, p^s_{asyncRepltET}\rangle \mid Ctrl_{Ifs}\langle p_{teReply}, p^g_{teReply}\rangle\\
& \mid Ctrl_{EI}\langle p_{executeTest}, r_{teExecTest}\rangle \mid Ctrl_{EI}\langle p_{teReply}, r_{asyncRepltET}\rangle\\
& \mid Ctrl_{SS}\langle s_0, s_1, p_{teAttach}\rangle \mid TE'_{comp}\langle p_{teAttach}, p^s_{teExecTest}, p^g_{teReply}\rangle)\\
TE'_{comp} \stackrel{def}{=}\ & (p_{teAttach}, p^s_{teExecTest}, p^g_{teReply}).\ldots\\
& (Ctr\langle s^{ctr}_0, s^{ctr}_1, p^g_{cDone}, p^g_{teExecTest}, r_{teAttach}, r_{detachTest}, r_{stopTest},\\
& \quad r_{provRefEInt}, r_{provRefORes}\rangle \mid Env\langle s^{env}_0, s^{env}_1, p^g_{eInteract}\rangle\\
& \mid Out\langle s^{out}_0, s^{out}_1, p^g_{oResult}, p^s_{oDone}, p^s_{oReply}\rangle \mid \ldots)
\end{aligned}
$$

Behaviour of a Component from the Example

Behaviour of interface references and binding, import and export, control of the component's life-cycle, and core behaviour of composite component "testEnvironment":

Out of scope of this presentation!

See: Marek Rychlý. "A case study on behavioural modelling of service-oriented architectures". *e-Informatica Software Engineering Journal*, 4(1):71–87, 2010.

Obsah

1 Introduction
- Component-Based Development (CBD)
- State of the Art and Motivation

2 A Metamodel for Modelling of CBS with Mobile Architecture
- A Four-layer Modelling Architecture and (E)MOF
- The Metamodel – Components and Interfaces
- An Example of a CBS Model and Behavioural Description

3 Summary and Future Work

Summary and Future Work

- Our approach consist of **the metamodel** and behavioural description.
  (the first implements a logical view, the second realises a process view of a CBS)
- The proposed metamodel is based on the (E)MOF.
  (several existing tools can be utilised, e.g. Eclipse Modeling Framework)
- It allows to describe a configuration of a CBS with mobile architecture.
  (primitive and composite components, their interfaces and bindings; to enable the components to provide their functionality and control evolution of the architecture)

Future work

- Integration with existing modelling tools.
- Design-time verification and model-checking, service and component modelling with constraints, etc.

Summary and Future Work

- Our approach consist of **the metamodel** and behavioural description.
  (the first implements a logical view, the second realises a process view of a CBS)
- The proposed metamodel is based on the (E)MOF.
  (several existing tools can be utilised, e.g. Eclipse Modeling Framework)
- It allows to describe a configuration of a CBS with mobile architecture.
  (primitive and composite components, their interfaces and bindings; to enable the components to provide their functionality and control evolution of the architecture)

Future work

- Integration with existing modelling tools.
- Design-time verification and model-checking, service and component modelling with constraints, etc.

Summary and Future Work

- Our approach consist of **the metamodel** and behavioural description.
  (the first implements a logical view, the second realises a process view of a CBS)
- The proposed metamodel is based on the (E)MOF.
  (several existing tools can be utilised, e.g. Eclipse Modeling Framework)
- It allows to describe a configuration of a CBS with mobile architecture.
  (primitive and composite components, their interfaces and bindings; to enable the components to provide their functionality and control evolution of the architecture)

Future work

- Integration with existing modelling tools.
- Design-time verification and model-checking, service and component modelling with constraints, etc.

Summary and Future Work

- Our approach consist of **the metamodel** and behavioural description.
  (the first implements a logical view, the second realises a process view of a CBS)
- The proposed metamodel is based on the (E)MOF.
  (several existing tools can be utilised, e.g. Eclipse Modeling Framework)
- It allows to describe a configuration of a CBS with mobile architecture.
  (primitive and composite components, their interfaces and bindings; to enable the components to provide their functionality and control evolution of the architecture)

Future work

- Integration with existing modelling tools.
- Design-time verification and model-checking, service and component modelling with constraints, etc.

Summary and Future Work

- Our approach consist of **the metamodel** and behavioural description.
  (the first implements a logical view, the second realises a process view of a CBS)
- The proposed metamodel is based on the (E)MOF.
  (several existing tools can be utilised, e.g. Eclipse Modeling Framework)
- It allows to describe a configuration of a CBS with mobile architecture.
  (primitive and composite components, their interfaces and bindings; to enable the components to provide their functionality and control evolution of the architecture)

Future work

- Integration with existing modelling tools.
- Design-time verification and model-checking, service and component modelling with constraints, etc.

Thank you for your attention!