Multi-level Contracts for Trusted Components

Mohamed Messabihi   Pascal André   Christian Attiogbé

COLOSS / LINA – UMR CNRS 6241
{Firstname.Lastname}@univ-nantes.fr

International Workshop on Component and Service Interoperability
June 29th, 2010, Málaga

Outline

1. Introduction
2. Multi-level Contracts in Component Model
3. Design and Verification Process using Multi-level Contracts
4. Experimentations with Kmelia/COSTO
5. Conclusion and Future Work

Outline

1 Introduction

2 Multi-level Contracts in Component Model

3 Design and Verification Process using Multi-level Contracts

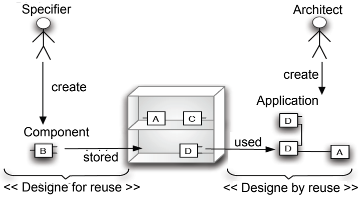
4 Experimentations with Kmelia/COSTO

5 Conclusion and Future Work

Introduction / context

Context: Trusted Component-Based Software Development

- *Commercial off-the-shelf* concept
- Trusted components and assemblies
- Various aspects (structure, behaviour, interaction...)



Goals:

- Models and techniques to specify and verify component-based systems
  - early in development phases, prior to implementation and deployment

Focus:

- Making explicit contracts at different level in component model for building trusted components and assemblies
  - Using assembly contracts to guarantee interoperability

Outline

1 Introduction

2 Multi-level Contracts in Component Model

3 Design and Verification Process using Multi-level Contracts

4 Experimentations with Kmelia/COSTO

5 Conclusion and Future Work

Using Multi-level Contracts in Component Model

What are contracts?

- In every day life:
  - Agreement between two or more parties
  - Establishing obligations or benefit of each of these parties
- A part of component definition

Definition (Component)

an unit of composition with **contractually** specified interfaces and explicit context dependencies only [Szyperski, 2002].

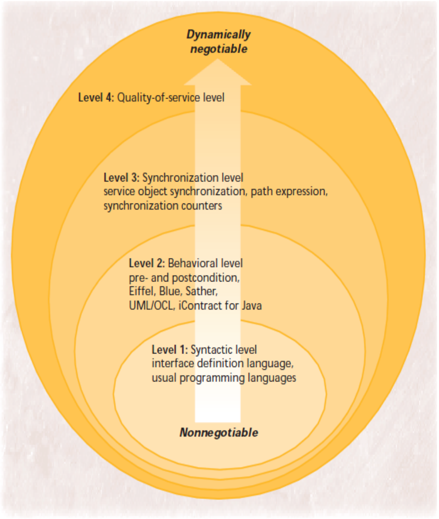
- Why are contracts useful?
  - Precision in specification & design
  - Making responsibilities explicit
  - Checking/Testing
  - Documentation

Using Multi-levels Contract in Component Model

Contract classification [Beugnard et al., 1999]

1. Syntactic contracts
2. Behavioural contracts
3. Synchronisation contracts
4. Quality of services contracts

	COM	SOFA	FRACTAL	Wright	CQM
Level 1	✓	✓	✓	✓	✓
Level 2	×	×	✓	×	×
Level 3	×	✓	×	✓	×
Level 4	×	×	×	×	✓

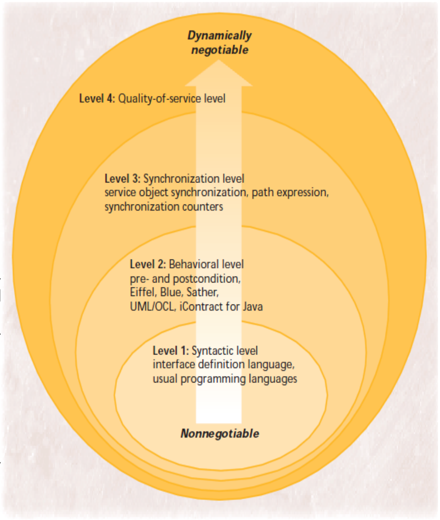

Dynamically negotiable
Level 4: Quality-of-service level
Level 3: Synchronization level
service object synchronization, path expression, synchronization counters
Level 2: Behavioral level
pre- and postcondition, Eiffel, Blue, Sather, UML/OCL, iContract for Java
Level 1: Syntactic level
interface definition language, usual programming languages
Nonnegotiable

Using Multi-levels Contract in Component Model

Contract classification [Beugnard et al., 1999]

1. Syntactic contracts
2. Behavioural contracts
3. Synchronisation contracts
4. Quality of services contracts

	COM	SOFA	FRACTAL	Wright	CQMI
Level 1	√	√	√	√	√
Level 2	×	×	√	×	×
Level 3	×	√	×	√	×
Level 4	×	×	×	×	√

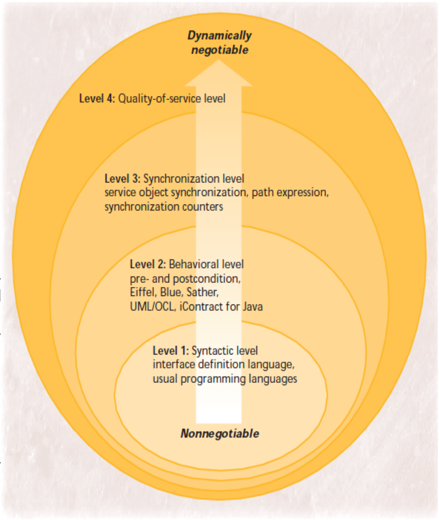

Dynamically negotiable
Level 4: Quality-of-service level
Level 3: Synchronization level
service object synchronization, path expression,
synchronization counters
Level 2: Behavioral level
pre- and postcondition,
Eiffel, Blue, Sather,
UML/OCL, iContract for Java
Level 1: Syntactic level
interface definition language,
usual programming languages
Nonnegotiable

Using Multi-levels Contract in Component Model

Contract classification [Beugnard et al., 1999]

1. Syntactic contracts
2. Behavioural contracts
3. Synchronisation contracts
4. Quality of services contracts

	COM	SOFA	FRACTAL	Wright	CQM
Level 1	✓	✓	✓	✓	✓
Level 2	✗	✗	✓	✗	✗
Level 3	✗	✓	✗	✓	✗
Level 4	✗	✗	✗	✗	✓


Dynamically negotiable
Level 4: Quality-of-service level
Level 3: Synchronization level service object synchronization, path expression, synchronization counters
Level 2: Behavioral level pre- and postcondition, Eiffel, Blue, Sather, UML/OCL, iContract for Java
Level 1: Syntactic level interface definition language, usual programming languages
Nonnegotiable

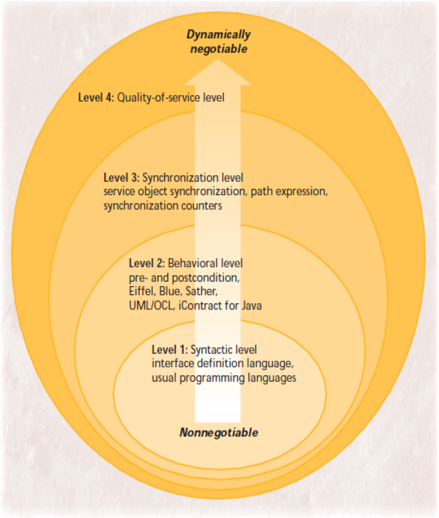
Using Multi-levels Contract in Component Model

Contract classification [Beugnard et al., 1999]

1. Syntactic contracts
2. Behavioural contracts
3. Synchronisation contracts
4. Quality of services contracts

	COM	SOFA	FRACTAL	Wright	CQM
Level 1	✓	✓	✓	✓	✓
Level 2	×	×	✓[a]	×	×
Level 3	×	✓	×	✓	×
Level 4	×	×	×	×	✓

[a] Using CCL-J in ConFract extention


Dynamically negotiable
Level 4: Quality-of-service level
Level 3: Synchronization level service object synchronization, path expression, synchronization counters
Level 2: Behavioral level pre- and postcondition, Eiffel, Blue, Sather, UML/OCL, iContract for Java
Level 1: Syntactic level interface definition language, usual programming languages
Nonnegotiable

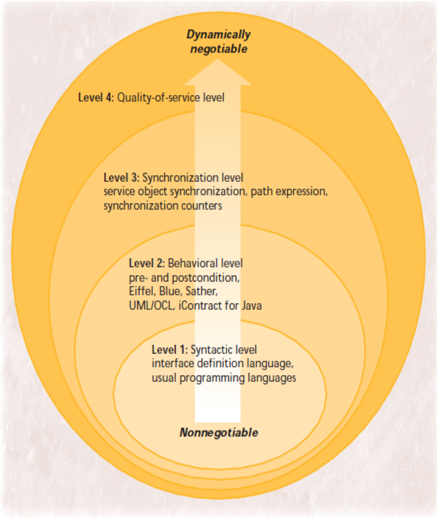
Using Multi-levels Contract in Component Model

Contract classification [Beugnard et al., 1999]

1. Syntactic contracts
2. Behavioural contracts
3. Synchronisation contracts
4. Quality of services contracts

	COM	SOFA	FRACTAL	Wright	CQM
Level 1	✓	✓	✓	✓	✓
Level 2	×	×	✓[a]	×	×
Level 3	×	✓	×	✓	×
Level 4	×	×	×	×	✓

[a] Using CCL-J in ConFract extention


Dynamically negotiable
Level 4: Quality-of-service level
Level 3: Synchronization level service object synchronization, path expression, synchronization counters
Level 2: Behavioral level pre- and postcondition, Eiffel, Blue, Sather, UML/OCL, iContract for Java
Level 1: Syntactic level interface definition language, usual programming languages
Nonnegotiable

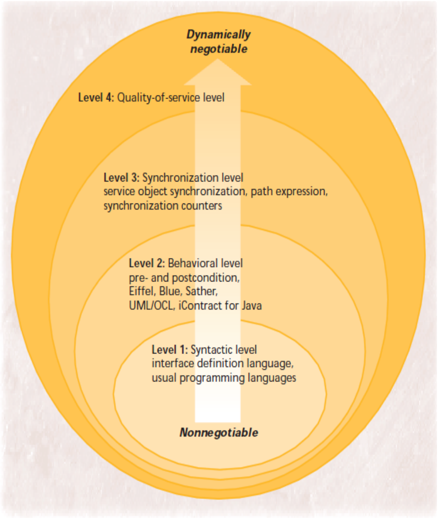
Using Multi-levels Contract in Component Model

Contract classification [Beugnard et al., 1999]

1. Syntactic contracts
2. Behavioural contracts
3. Synchronisation contracts
4. Quality of services contracts

	COM	SOFA	FRACTAL	Wright	CQM
Level 1	✓	✓	✓	✓	✓
Level 2	✗	✗	✓[a]	✗	✗
Level 3	✗	✓	✗	✓	✗
Level 4	✗	✗	✗	✗	✓

[a] Using CCL-J in ConFract extention



Dynamically negotiable
Level 4: Quality-of-service level
Level 3: Synchronization level
service object synchronization, path expression, synchronization counters
Level 2: Behavioral level
pre- and postcondition, Eiffel, Blue, Sather, UML/OCL, iContract for Java
Level 1: Syntactic level
interface definition language, usual programming languages
Nonnegotiable

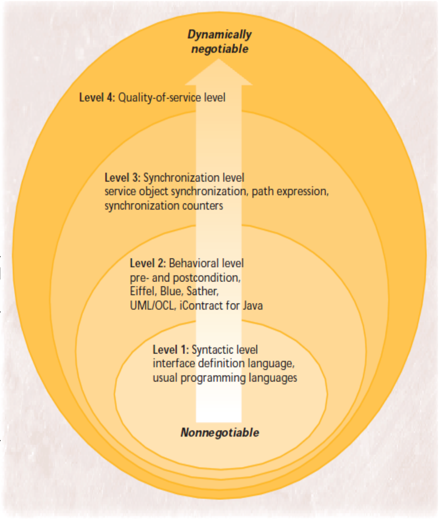
Using Multi-levels Contract in Component Model

Contract classification [Beugnard et al., 1999]

1. Syntactic contracts
2. Behavioural contracts
3. Synchronisation contracts
4. Quality of services contracts

	COM	SOFA	FRACTAL	Wright	CQMI
Level 1	✓	✓	✓	✓	✓
Level 2	×	×	✓[a]	×	×
Level 3	×	✓	×	✓	×
Level 4	×	×	×	×	✓

[a] Using CCL-J in ConFract extention



Dynamically negotiable
Level 4: Quality-of-service level
Level 3: Synchronization level service object synchronization, path expression, synchronization counters
Level 2: Behavioral level pre- and postcondition, Eiffel, Blue, Sather, UML/OCL, iContract for Java
Level 1: Syntactic level interface definition language, usual programming languages
Nonnegotiable

No one covers more than two levels

Using Multi-levels Contract in Component Model

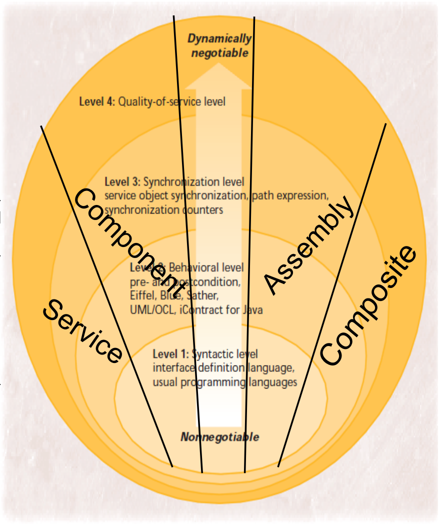
Contract classification [Beugnard et al., 1999]

1. Syntactic contracts
2. Behavioural contracts
3. Synchronisation contracts
4. Quality of services contracts

	COM	SOFA	FRACTAL	Wright	CQMI
Level 1	√	√	√	√	√
Level 2	×	×	√[a]	×	×
Level 3	×	√	×	√	×
Level 4	×	×	×	×	√

[a] Using CCL-J in ConFract extention

Different contexts for making contracts



Dynamically negotiable
Level 4: Quality-of-service level
Level 3: Synchronization level
service object synchronization, path expression, synchronization counters
Level 2: Behavioral level
pre- and postcondition, Eiffel, Blue, Sather, UML/OCL, iContract for Java
Level 1: Syntactic level
interface definition language, usual programming languages
Nonnegotiable
Component
Service
Assembly
Composite

Using Multi-levels Contract in Component Model

Crossing contexts and contracts = Multilevel Contracts

	Component model level			
Contract levels	Service	Component	Assembly	Composite
Syntactic	type checking	interface, type checking	signature matching, service dependencies	promotion, observability
Behaviour	functional correctness	invariant preservation	pre/post compliance	pre/post compliance
Synchronisation	deadlock freedom	protocol	behavioural compatibility	
QoS	-	-	-	-
Properties	Correctness	Consistency	Interoperability	Encapsulation

Illustration with Kmelia component model

Architecture levels in Kmelia [André et al., 2009]

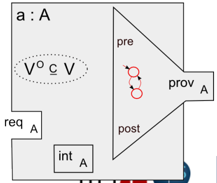
Service component "functionality"
- Interface = sub-services
- *Assertions = pre-/postconditions*
- Dynamic behaviour = eLTS

Component abstract and non executable
- State space with an *Invariant*
- Interface = required + provided services

Assembly Links between provided/required services

Composition encapsulation and promotion

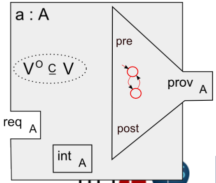
```
Provided service1 ()
    Interface   <Interface descr>
    Pre         <Predicate>
    Post        <Predicate>
    Behaviour
        init    q_0
        final   q_f
        { ...,
        q_i - - label - - > q_j,
        ... }
end
Required service2 () ...
```



Architecture levels in Kmelia [André et al., 2009]

Service	component "functionality"
	• Interface = sub-services
	• *Assertions = pre-/postconditions*
	• Dynamic behaviour = eLTS
Component	abstract and non executable
	• State space with an *Invariant*
	• Interface = required + provided services
Assembly	Links between provided/required services
Composition	encapsulation and promotion

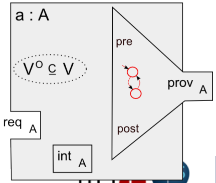
```
Provided service1 ()
      Interface    <Interface descr>
      Pre          <Predicate>
      Post         <Predicate>
      Behaviour
            init      q_0
            final     q_f
            { ...,
            q_i - - label - - > q_j,
            ... }
end
Required service2 () ...
```



Architecture levels in Kmelia [André et al., 2009]

Service	component "functionality"
	• Interface = sub-services
	• *Assertions = pre-/postconditions*
	• Dynamic behaviour = eLTS
Component	abstract and non executable
	• State space with an *Invariant*
	• Interface = required + provided services
Assembly	Links between provided/required services
Composition	encapsulation and promotion

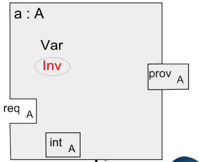
```
Provided service1 ()
      Interface   <Interface descr>
      Pre         <Predicate>
      Post        <Predicate>
      Behaviour
            init    q_0
            final   q_f
            { ...,
            q_i - - label - - > q_j,
            ... }
end
Required service2 () ...
```



Architecture levels in Kmelia [André et al., 2009]

Service	component "functionality"
	• Interface = sub-services • *Assertions = pre-/postconditions* • Dynamic behaviour = eLTS
Component	abstract and non executable
	• State space with an *Invariant* • Interface = required + provided services
Assembly	Links between provided/required services
Composition	encapsulation and promotion

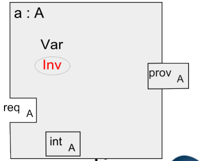
```
Component compo_name
    Interface       <Interface descr.>
    Types           < Type Defs >
    Variables       <Var list>
    Invariant       <Predicate>
    Initialisation
        ...         // var. assignments
    Services
        ...
end
```



Architecture levels in Kmelia [André et al., 2009]

Service	component "functionality"
	• Interface = sub-services • *Assertions = pre-/postconditions* • Dynamic behaviour = eLTS
Component	abstract and non executable
	• State space with an *Invariant* • Interface = required + provided services
Assembly	Links between provided/required services
Composition	encapsulation and promotion

```
Component compo_name
    Interface       <Interface descr.>
    Types           < Type Defs >
    Variables       <Var list>
    Invariant       <Predicate>
    Initialisation
        ...         // var. assignments
    Services
        ...
end
```



Architecture levels in Kmelia [André et al., 2009]

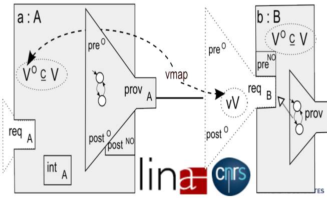
Service component "functionality"
- Interface = sub-services
- *Assertions = pre-/postconditions*
- Dynamic behaviour = eLTS

Component abstract and non executable
- State space with an *Invariant*
- Interface = required + provided services

Assembly Links between provided/required services

Composition encapsulation and promotion

```
ASSEMBLY assembly_name
    Components
                <Compos>
    Links
                < Links>
        context mapping
                <Predicats>
        ...
End_links
END
```



Architecture levels in Kmelia [André et al., 2009]

Service component "functionality"

- Interface = sub-services
- *Assertions = pre-/postconditions*
- Dynamic behaviour = eLTS

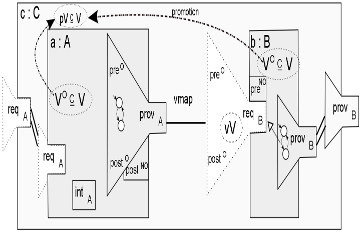
Component abstract and non executable

- State space with an *Invariant*
- Interface = required + provided services

Assembly Links between provided/required services

Composition encapsulation and promotion

```
COMPOSITION composite_name
    ASSEMBLY assembly_name
        <...>
    End
        PROMOTION
        Links
                <...>
        Variables
                <...>
END
```



Outline

1 Introduction

2 Multi-level Contracts in Component Model

3 Design and Verification Process using Multi-level Contracts

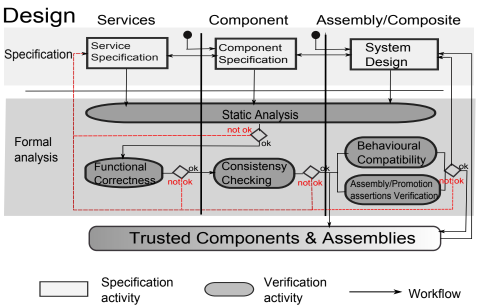
4 Experimentations with Kmelia/COSTO

5 Conclusion and Future Work

Multi-levels Contracts Design and Verification Process

Making explicit contracts in component-based development process

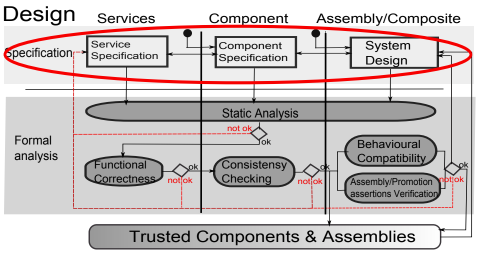
1. Service
2. Component
3. Assembly

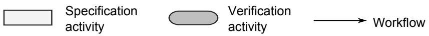


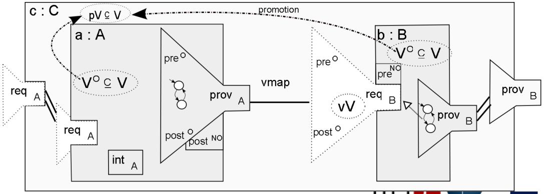
Multi-levels Contracts Design and Verification Process

Making explicit contracts in component-based development process

1. Service
   -
   -
   -
2. Component
   -
   -
   -
3. Assembly
   -
   -
   -



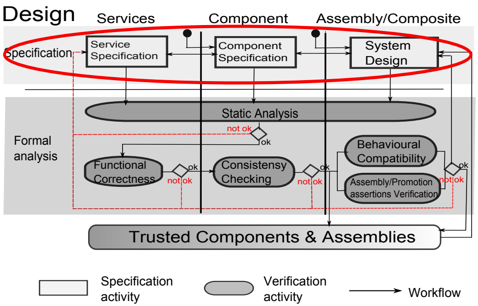


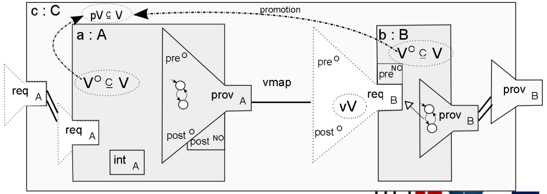


Multi-levels Contracts Design and Verification Process

Making explicit contracts in component-based development process

1. Service contract
2. Component contract
3. Assembly contract

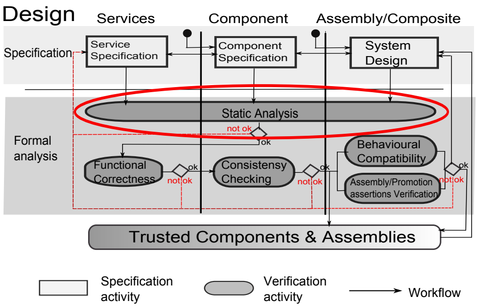


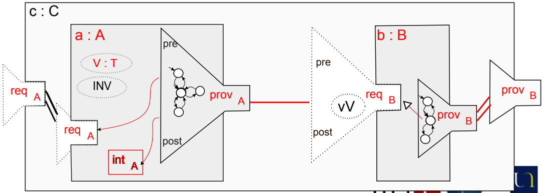


Multi-levels Contracts Design and Verification Process

Making explicit contracts in component-based development process

1. Service contract
   - Service dependency
   -
   -
2. Component contract
   - Service accessibility
   -
   -
3. Assembly contract
   - Static interoperability
   -
   -

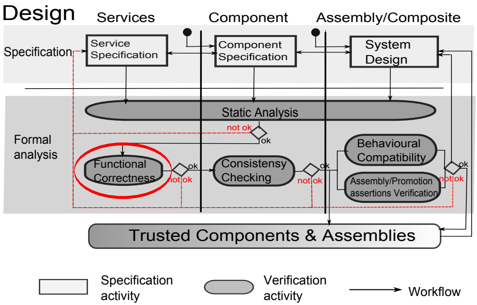


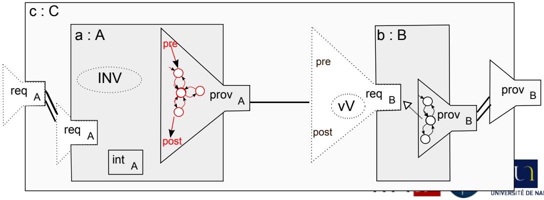


Multi-levels Contracts Design and Verification Process

Making explicit contracts in component-based development process

1. Service contract
   - Service dependency
   - *Functional correctness*
   -
2. Component contract
   - Service accessibility
   -
   -
3. Assembly contract
   - Static interoperability
   -
   -

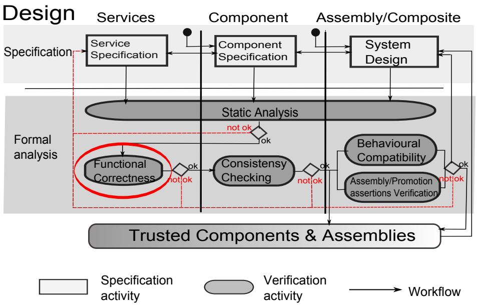


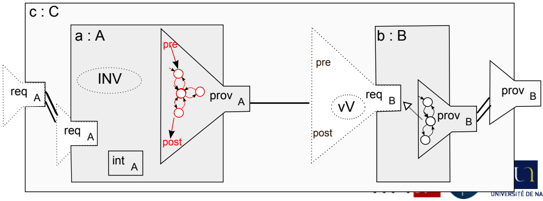


Multi-levels Contracts Design and Verification Process

Making explicit contracts in component-based development process

1. Service contract
   - Service dependency
   - *Functional correctness*
   - *Behavioural consistency*
2. Component contract
   - Service accessibility
   -
   -
3. Assembly contract
   - Static interoperability
   -
   -

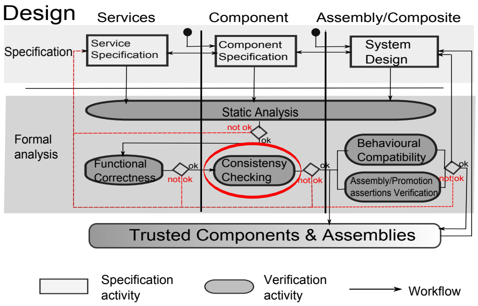


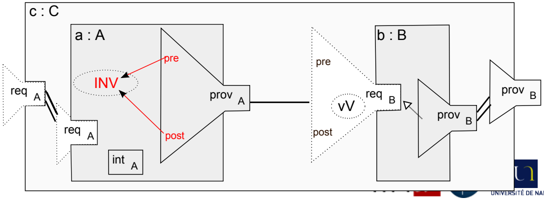


Multi-levels Contracts Design and Verification Process

Making explicit contracts in component-based development process

1. Service contract
   - Service dependency
   - *Functional correctness*
   - *Behavioural consistency*
2. Component contract
   - Service accessibility
   - *Component consistency*
   -
3. Assembly contract
   - Static interoperability
   -
   -

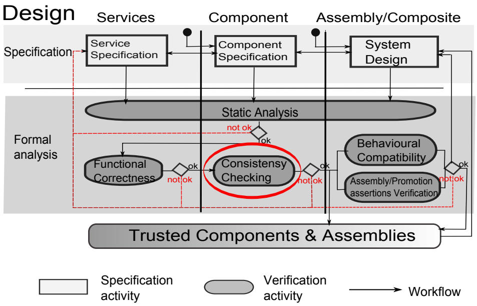


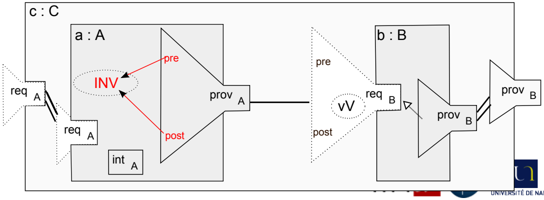


Multi-levels Contracts Design and Verification Process

Making explicit contracts in component-based development process

1. Service contract
   - Service dependency
   - *Functional correctness*
   - *Behavioural consistency*
2. Component contract
   - Service accessibility
   - *Component consistency*
   - *Protocol correctness*
3. Assembly contract
   - Static interoperability
   -
   -

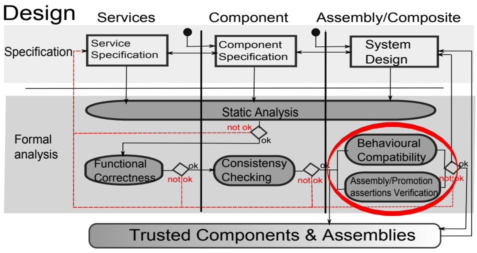


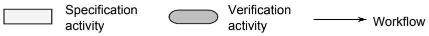


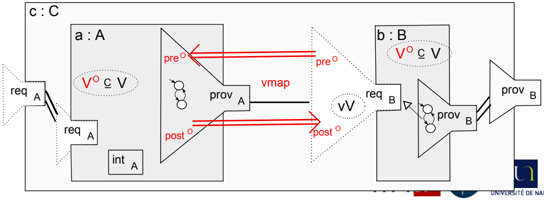
Multi-levels Contracts Design and Verification Process

Making explicit contracts in component-based development process

1. Service contract
   - Service dependency
   - *Functional correctness*
   - *Behavioural consistency*
2. Component contract
   - Service accessibility
   - *Component consistency*
   - *Protocol correctness*
3. Assembly contract
   - Static interoperability
   - *Service assertions compliance* on an assembly link.
   -



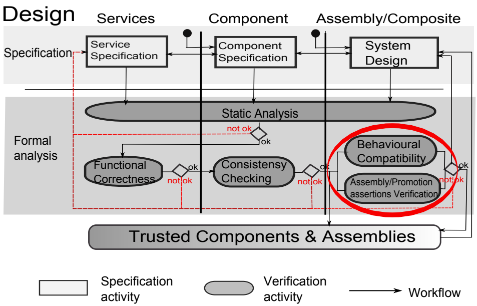


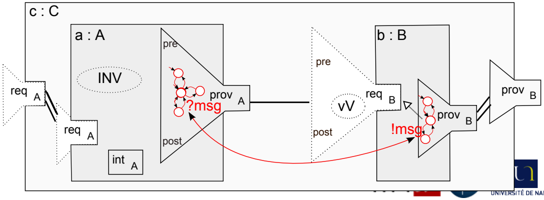


Multi-levels Contracts Design and Verification Process

Making explicit contracts in component-based development process

1. Service contract
   - Service dependency
   - *Functional correctness*
   - *Behavioural consistency*
2. Component contract
   - Service accessibility
   - *Component consistency*
   - *Protocol correctness*
3. Assembly contract
   - Static interoperability
   - *Service assertions compliance* on an assembly link.
   - *Behavioural compatibility* between the linked services in an assembly.





Outline

1 Introduction

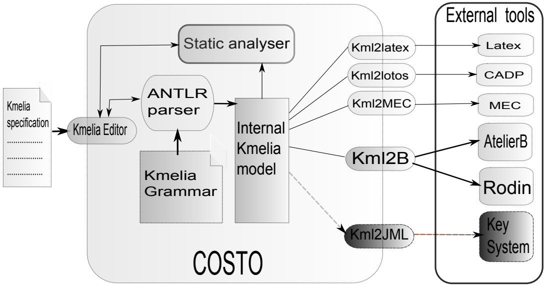
2 Multi-level Contracts in Component Model

3 Design and Verification Process using Multi-level Contracts

4 Experimentations with Kmelia/COSTO

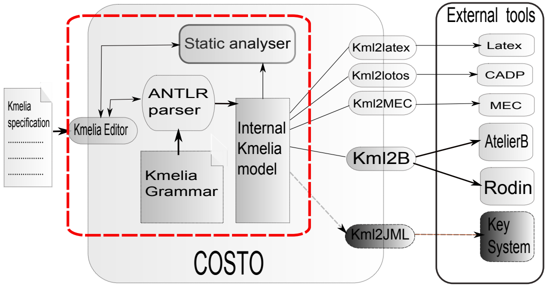
5 Conclusion and Future Work

COSTO Framework Overview



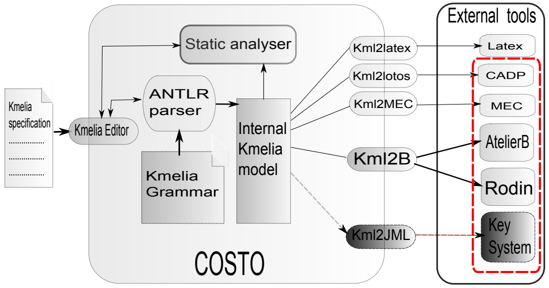
- COSTO is a Toolbox made of Eclipe-based plugins, dedicated to the specification and formal analysis of Kmelia Components.
- COSTO manages the Kmelia specifications and handles the verification of the primary properties (syntax, types, observability, signature matching, services dependency).
- Verifications of complex properties such as deadlock freeness, component or assembly consistency are delegated to other more appropriate tools.

COSTO Framework Overview



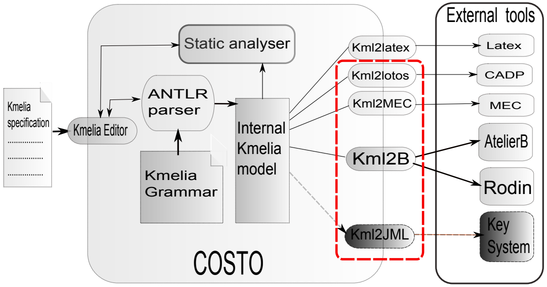
- COSTO is a Toolbox made of Eclipe-based plugins, dedicated to the specification and formal analysis of Kmelia Components.
- COSTO manages the Kmelia specifications and handles the verification of the primary properties (syntax, types, observability, signature matching, services dependency).
- Verifications of complex properties such as deadlock freeness, component or assembly consistency are delegated to other more appropriate tools.

COSTO Framework Overview



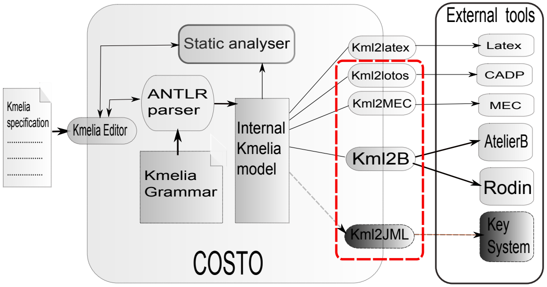
- COSTO is a Toolbox made of Eclipe-based plugins, dedicated to the specification and formal analysis of Kmelia Components.
- COSTO manages the Kmelia specifications and handles the verification of the primary properties (syntax, types, observability, signature matching, services dependency).
- Verifications of complex properties such as deadlock freeness, component or assembly consistency are delegated to other more appropriate tools.

COSTO Framework Overview



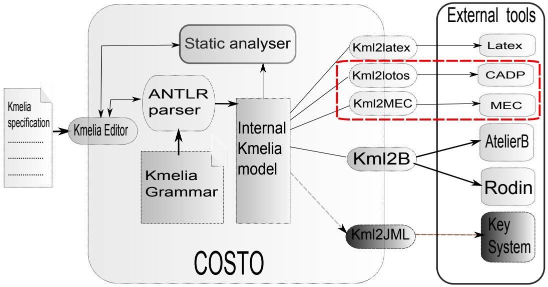
- The parts of the Kmelia specifications involved in the target property are extracted and translated into the input formalism of the target tool.
- The property is checked under the external tool. Currently:
  - Behavioural compatibility experimented with LOTOS/CADP and MEC
  - Component consistency and assembly assertions compliance experimented with AtelierB and Rodin
  - Functional correctness experimented with the Key tool

COSTO Framework Overview



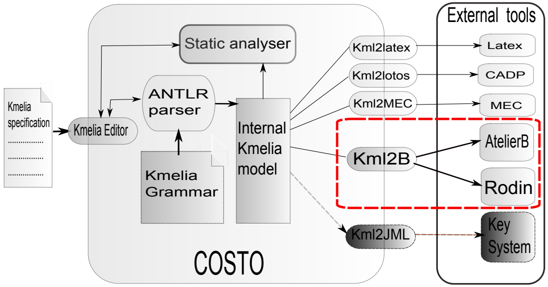
- The parts of the Kmelia specifications involved in the target property are extracted and translated into the input formalism of the target tool.
- The property is checked under the external tool. Currently:
  - Behavioural compatibility experimented with LOTOS/CADP and MEC
  - Component consistency and assembly assertions compliance experimented with AtelierB and Rodin
  - Functional correctness experimented with the Key tool

COSTO Framework Overview



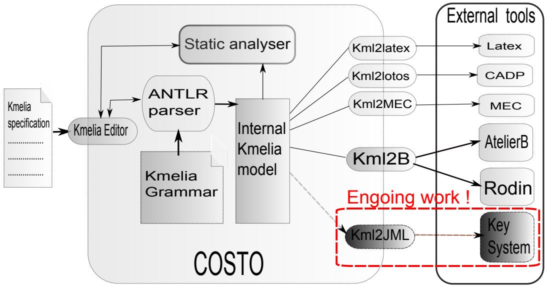
- The parts of the Kmelia specifications involved in the target property are extracted and translated into the input formalism of the target tool.
- The property is checked under the external tool. Currently:
  - Behavioural compatibility experimented with LOTOS/CADP and MEC
  - Component consistency and assembly assertions compliance experimented with AtelierB and Rodin
  - Functional correctness experimented with the Key tool

COSTO Framework Overview



- The parts of the Kmelia specifications involved in the target property are extracted and translated into the input formalism of the target tool.
- The property is checked under the external tool. Currently:
  - Behavioural compatibility experimented with LOTOS/CADP and MEC
  - Component consistency and assembly assertions compliance experimented with AtelierB and Rodin
  - Functional correctness experimented with the Key tool

COSTO Framework Overview



- The parts of the Kmelia specifications involved in the target property are extracted and translated into the input formalism of the target tool.
- The property is checked under the external tool. Currently:
  - Behavioural compatibility experimented with LOTOS/CADP and MEC
  - Component consistency and assembly assertions compliance experimented with AtelierB and Rodin
  - Functional correctness experimented with the Key tool (under experimentation)

Outline

1 Introduction

2 Multi-level Contracts in Component Model

3 Design and Verification Process using Multi-level Contracts

4 Experimentations with Kmelia/COSTO

5 Conclusion and Future Work

Conclusion

- Making explicit contract at different level in component model (service, component, assembly, composite)
- A process development based on contract checking
- A mechanisation of this process based on integrating existing tools such as theorem-provers or model-checkers

Conclusion

- Making explicit contract at different level in component model (service, component, assembly, composite)
- A process development based on contract checking
- A mechanisation of this process based on integrating existing tools such as theorem-provers or model-checkers

Conclusion

- Making explicit contract at different level in component model (service, component, assembly, composite)
- A process development based on contract checking
- A mechanisation of this process based on integrating existing tools such as theorem-provers or model-checkers

Perspectives

Short term (actually ongoing work)

- Follow the experimentation with the Key tool (new COSTO plugin)
- Enable the feedback to the specification step from the results of external tools

Medium term

- Using contracts for testing component code

Long term

- Apply these ideas and techniques to heterogeneous component and service models

Perspectives

Short term (actually ongoing work)

- Follow the experimentation with the Key tool (new COSTO plugin)
- Enable the feedback to the specification step from the results of external tools

Medium term

- Using contracts for testing component code

Long term

- Apply these ideas and techniques to heterogeneous component and service models

Perspectives

Short term (actually ongoing work)

- Follow the experimentation with the Key tool (new COSTO plugin)
- Enable the feedback to the specification step from the results of external tools

Medium term

- Using contracts for testing component code

Long term

- Apply these ideas and techniques to heterogeneous component and service models

Thanks for your attention!

References I

André, P., Ardourel, G., Attiogbé, C., and Lanoix, A. (2009).
Using Assertions to Enhance the Correctness of Kmelia Components and their Assemblies.
In *6th International Workshop on Formal Aspects of Component Software(FACS 2009)*, LNCS, pages –.
to appear.

Beugnard, A., Jézéquel, J.-M., Plouzeau, N., and Watkins, D. (1999).
Making components contract aware.
Computer, 32(7):38–45.

Szyperski, C. (2002).
Component Software: Beyond Object-Oriented Programming.
Addison Wesley Publishing Company/ACM Press.
ISBN 0-201-74572-0.