Multi-level Contracts for Trusted Components

Vers le test de contrats

Pascal André Gilles Ardourel

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

Sur une base du Workshop WCSI, 2010,
avec Christian Attiogbé et Mohamed Messabihi

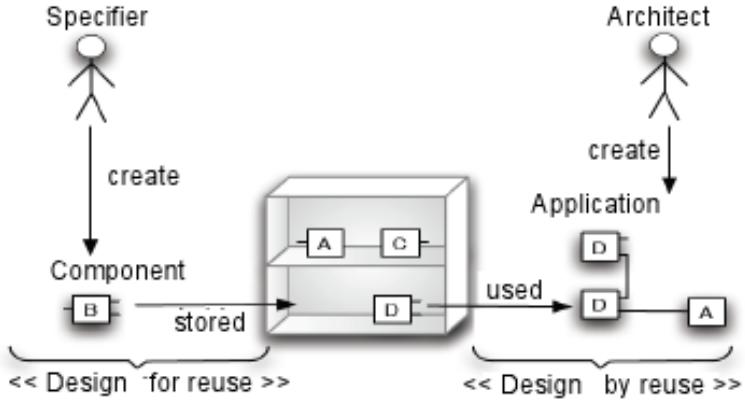
Outline

Outline

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

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
    - correct reuse
    - correct composition (services, components)
  - Documentation

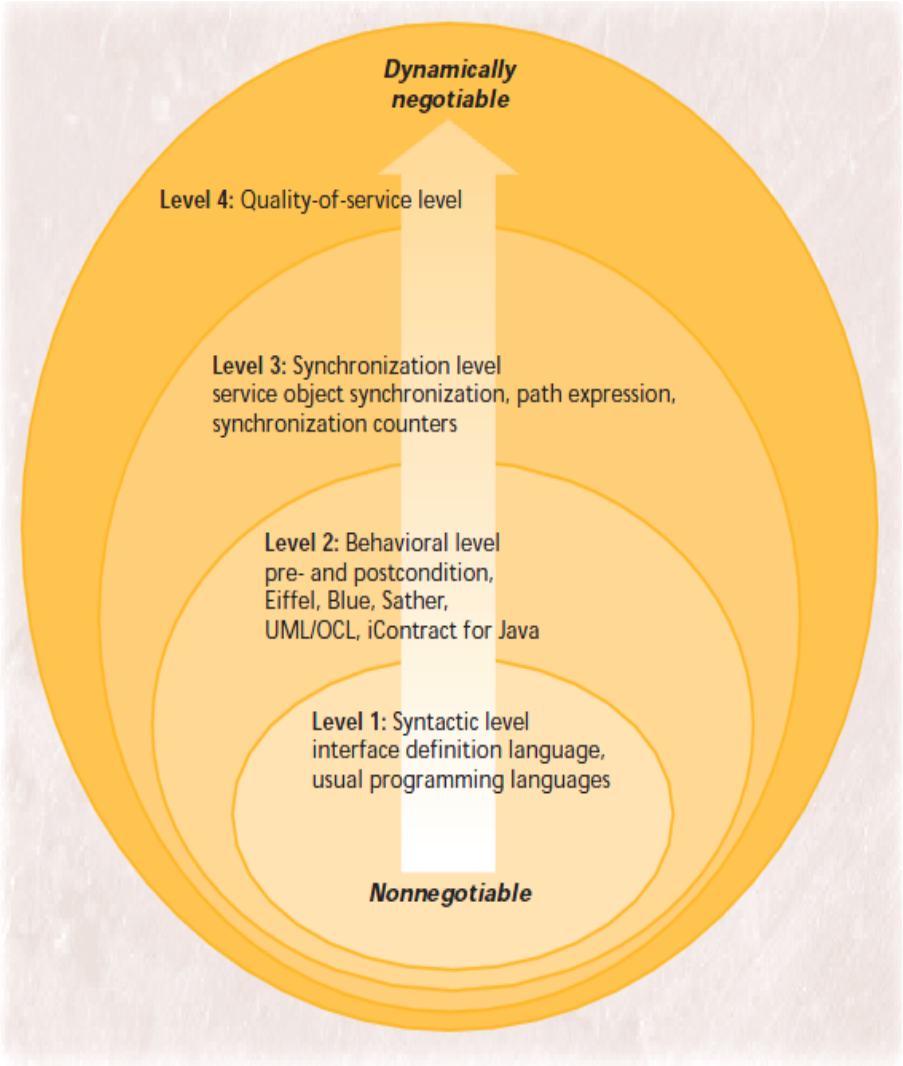
Using Multi-levels Contract in Component Model

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	×	×	×	×	√

[a] Using CCL-J in ConFract extention

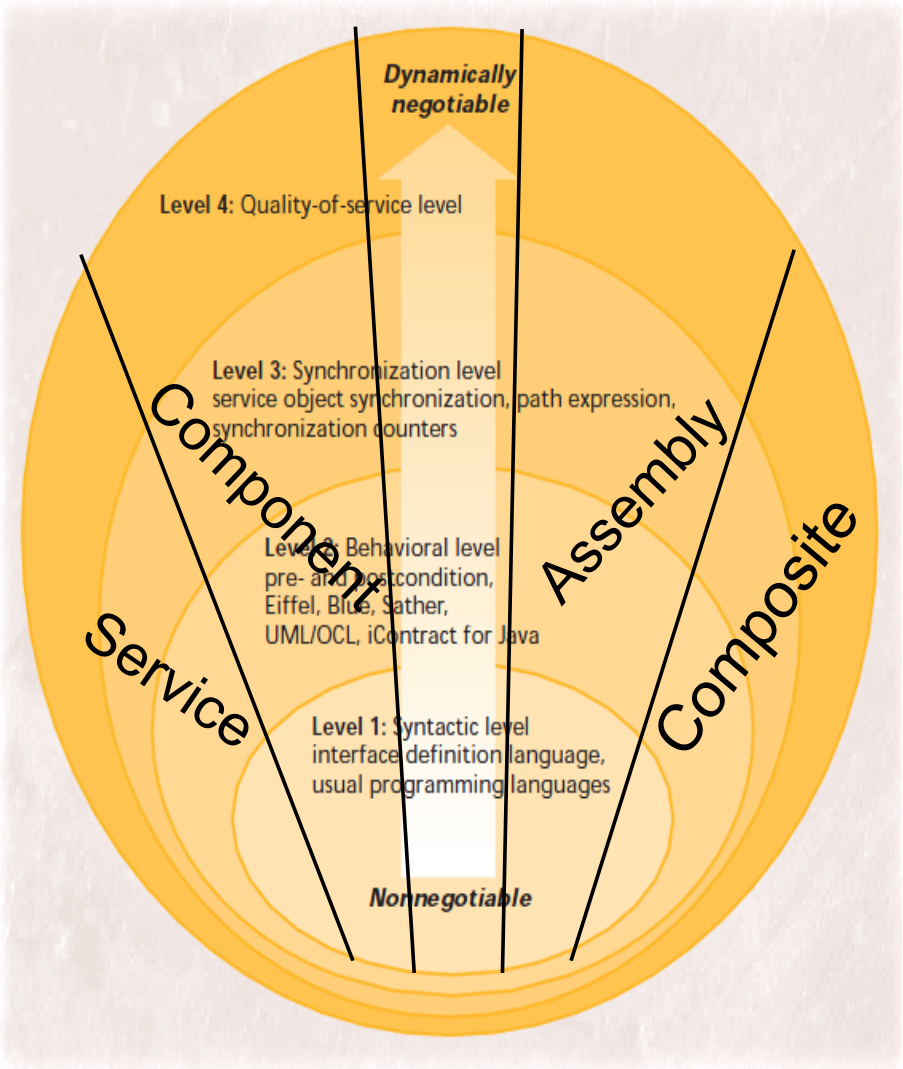


No one covers more than two levels

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



Different contexts for making contracts

Using Multi-levels Contract in Component Model

Crossing contexts and contracts = Multilevel Contracts

Contract level	Component model level			
	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

Model levels in Kmelia

Service component "functionality"

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

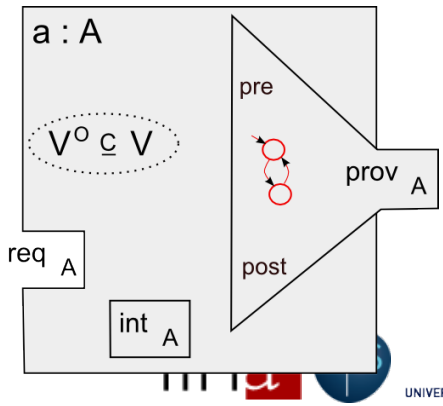
Component abstract and non executable

- 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 () ...
```



[André et al., 2009]

Model levels in Kmelia

Service component "functionality"

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

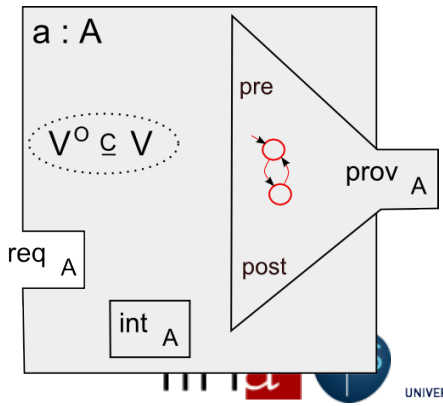
Component abstract and non executable

- 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 () ...
```



[André et al., 2009]

Model levels in Kmelia

Service component "functionality"

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

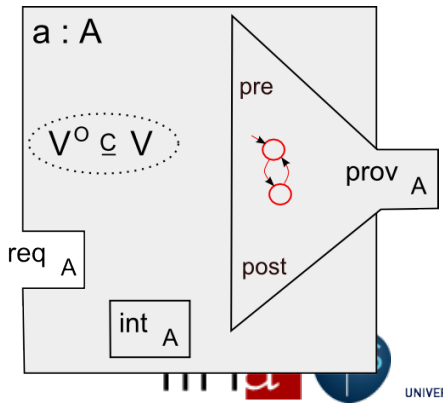
Component abstract and non executable

- 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 () ...
```



[André et al., 2009]

Model levels in Kmelia

Service component "functionality"

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

Component abstract and non executable

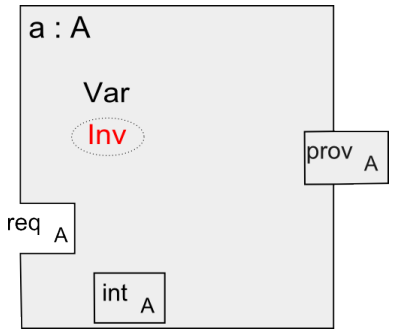
- State space with an *Invariant*
- Interface = required + provided services

Assembly Links between provided/required services

Composition encapsulation and promotion

[André et al., 2009]

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



Model levels in Kmelia

Service component "functionality"

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

Component abstract and non executable

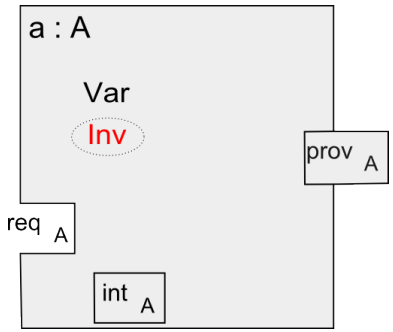
- State space with an *Invariant*
- Interface = required + provided services

Assembly Links between provided/required services

Composition encapsulation and promotion

[André et al., 2009]

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



Model levels in Kmelia

Service component "functionality"

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

Component abstract and non executable

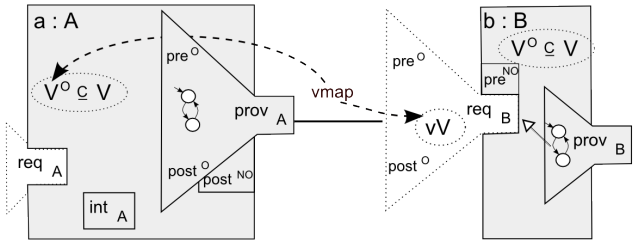
- State space with an *Invariant*
- Interface = required + provided services

Assembly Links between provided/required services

Composition encapsulation and promotion

[André et al., 2009]

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



Model levels in Kmelia

Service component "functionality"

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

Component abstract and non executable

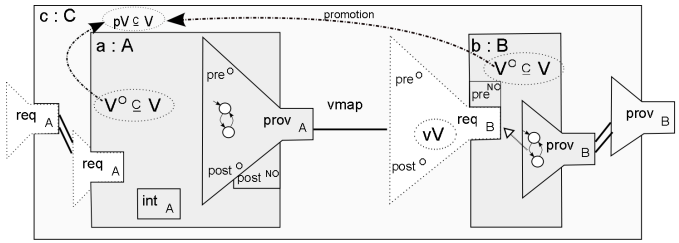
- State space with an *Invariant*
- Interface = required + provided services

Assembly Links between provided/required services

Composition encapsulation and promotion

[André et al., 2009]

COMPOSITION composite
ASSEMBLY
<...>
End
PROMOTION
Links
< ...>
Variables
< ...>
END

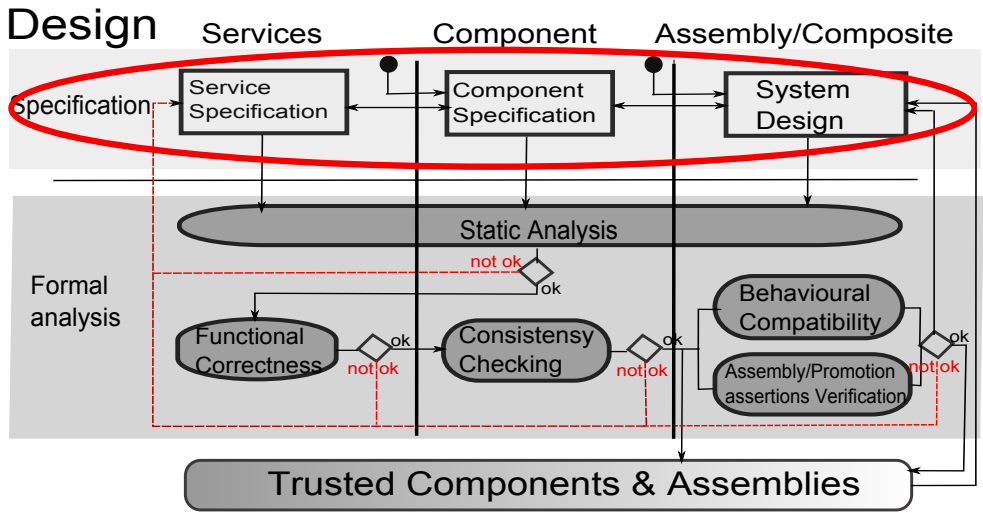


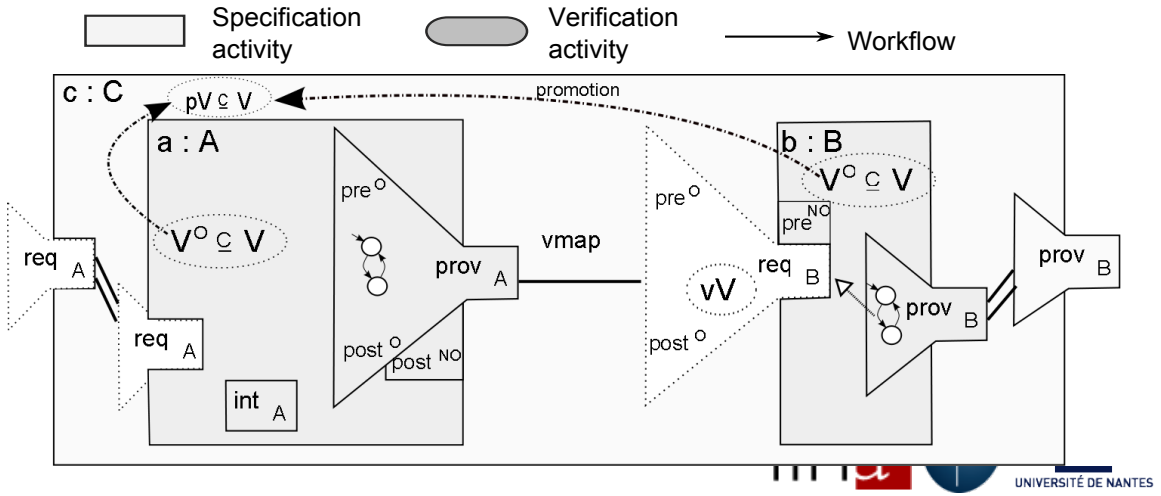
Outline

Multi-levels Contracts Design and Verification Process

Making explicit contracts in component-based development process

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

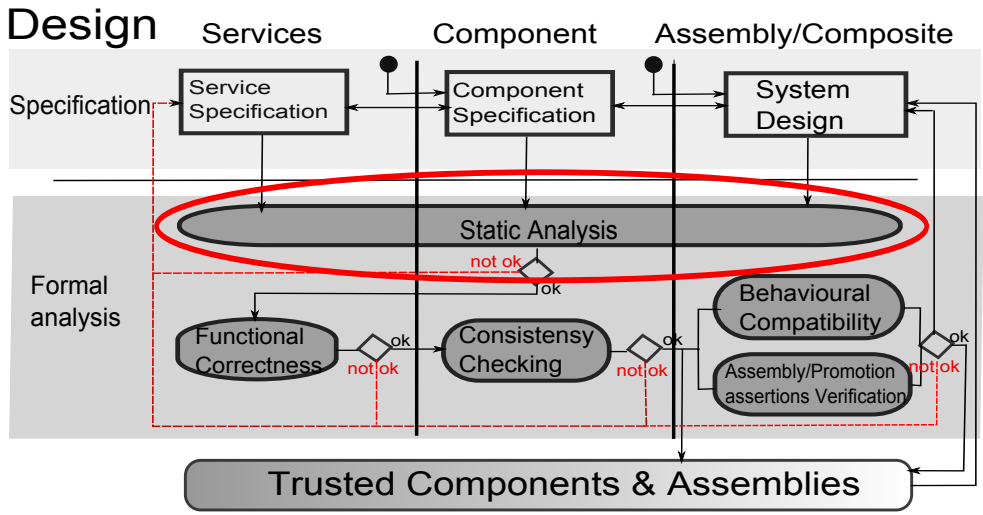


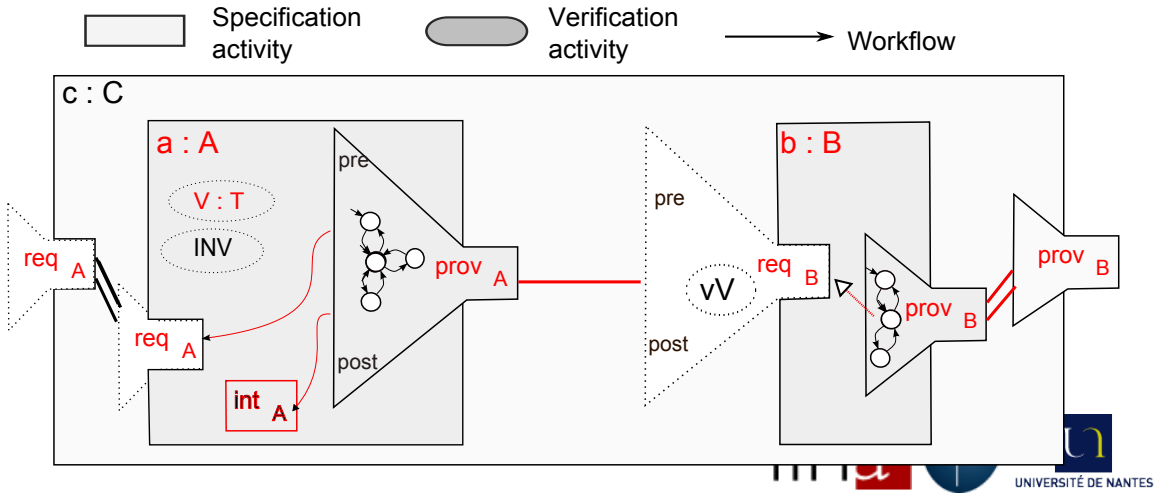


Multi-levels Contracts Design and Verification Process

Making explicit contracts in component-based development process

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

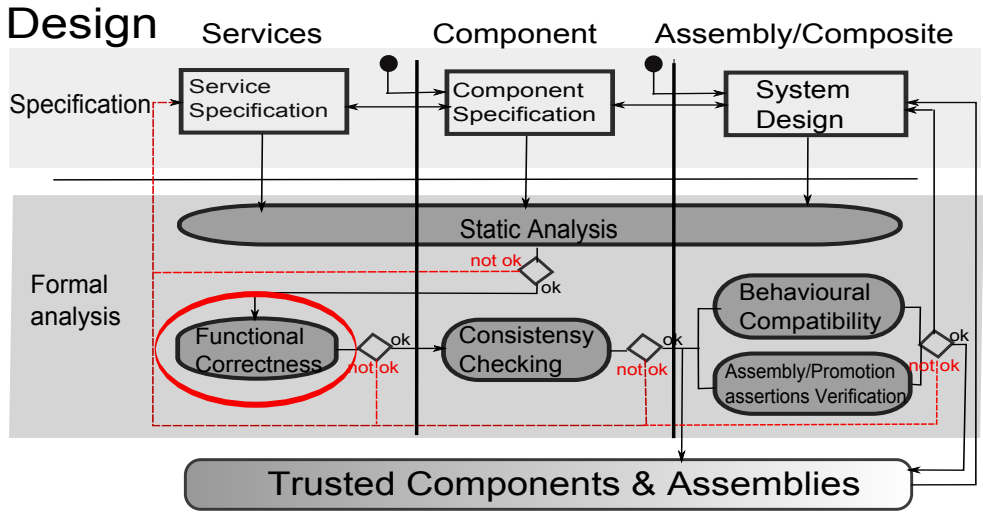


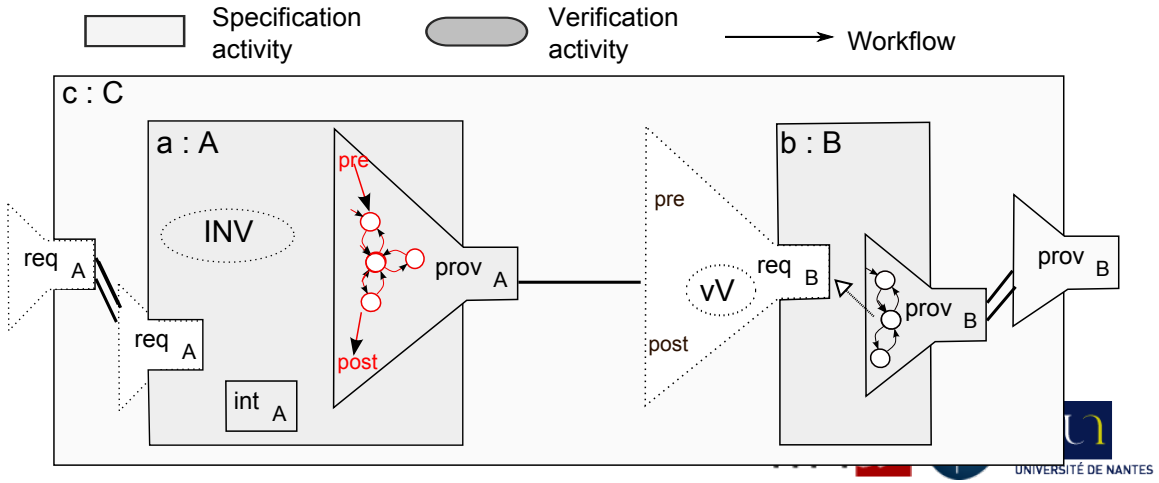


Multi-levels Contracts Design and Verification Process

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
   -
   -

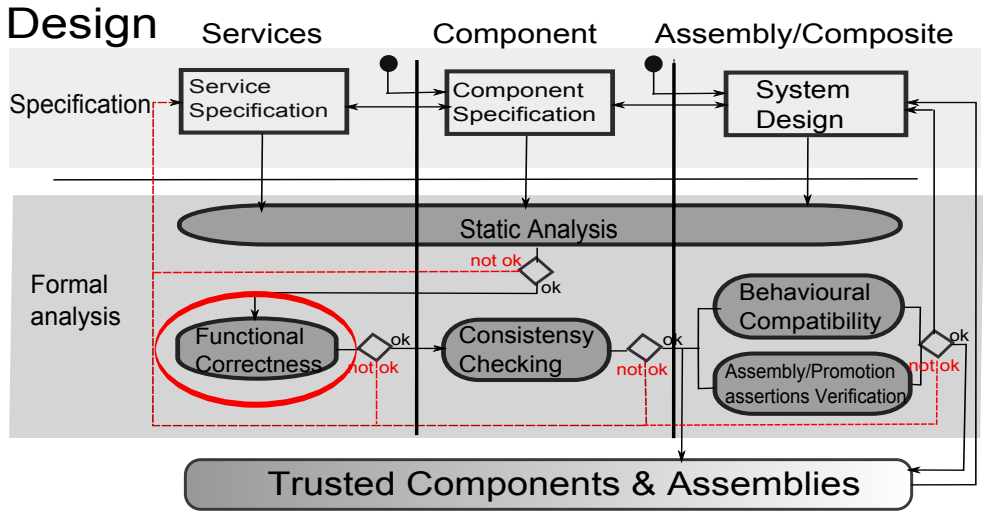


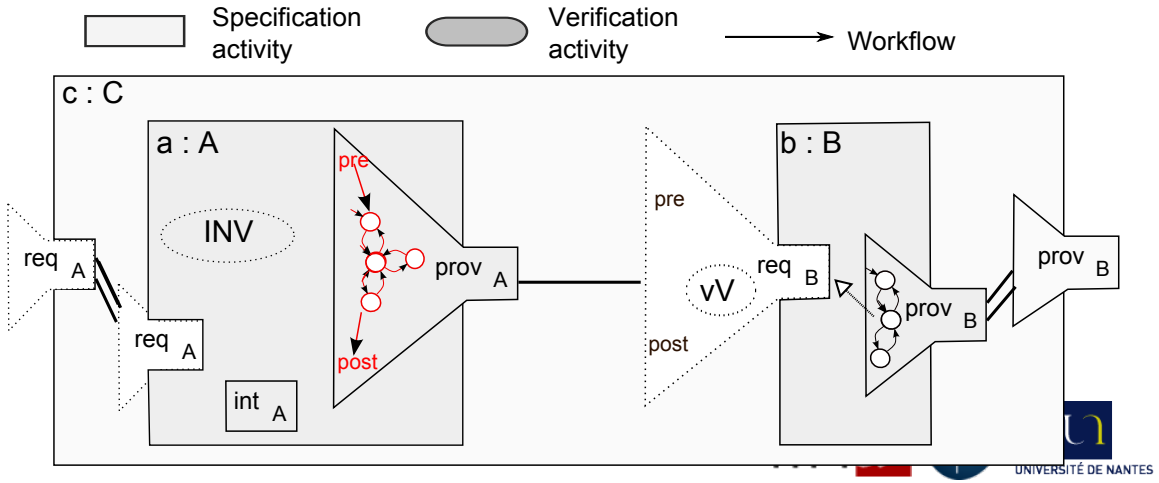


Multi-levels Contracts Design and Verification Process

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
   -
   -

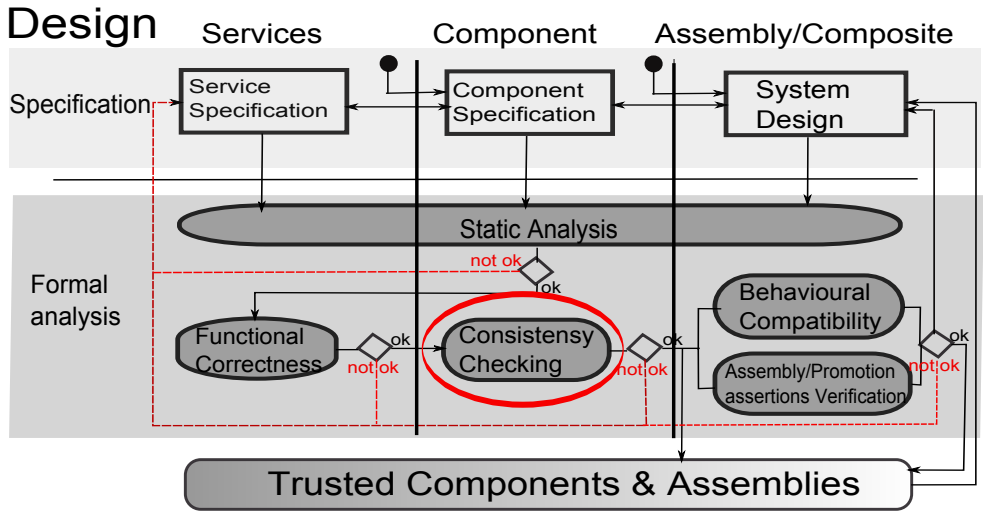


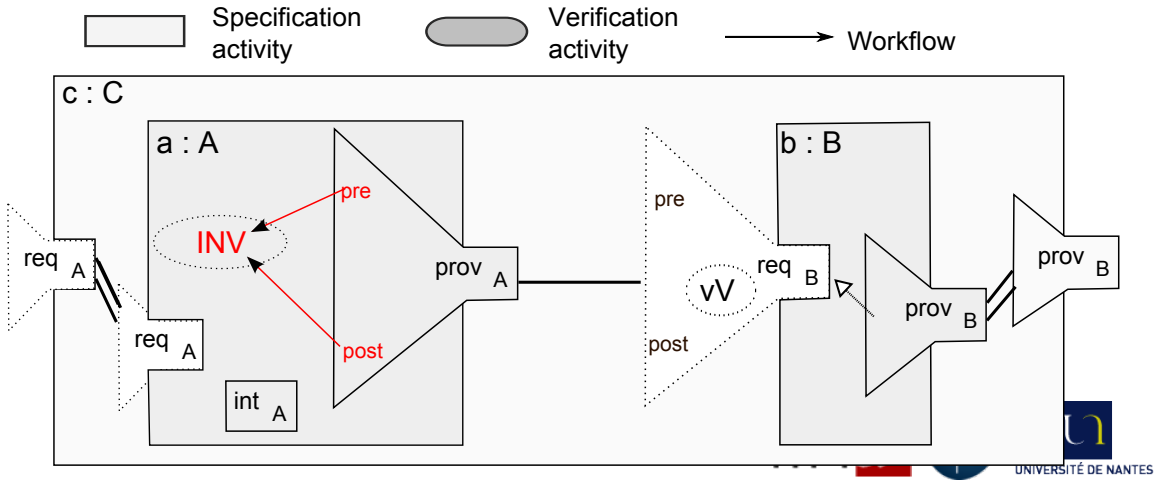


Multi-levels Contracts Design and Verification Process

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
   -
   -

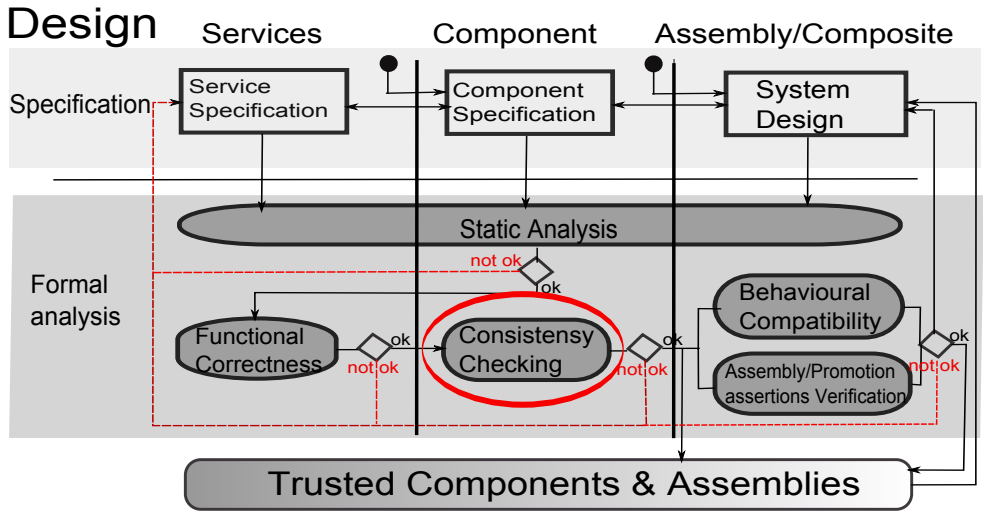


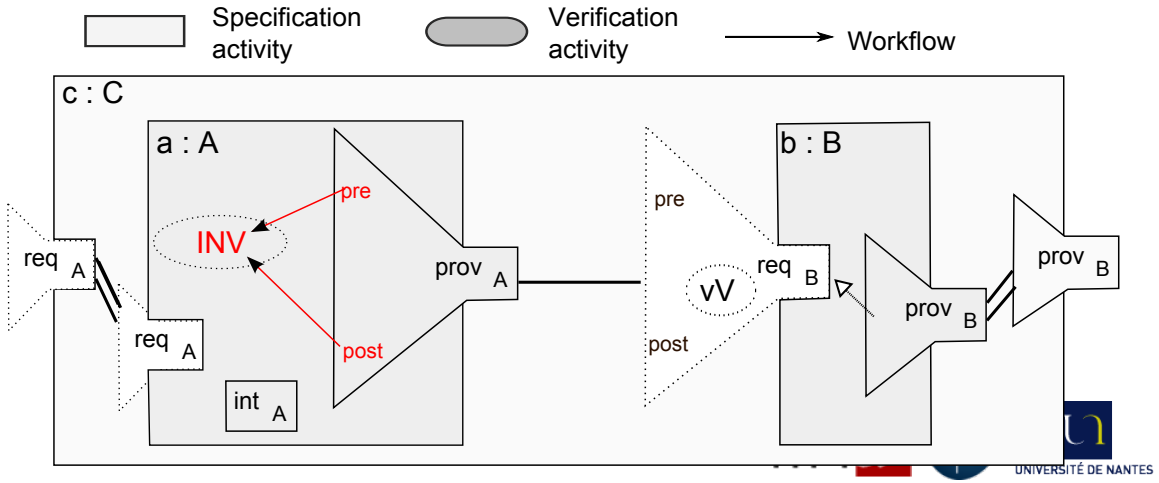


Multi-levels Contracts Design and Verification Process

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
   -
   -

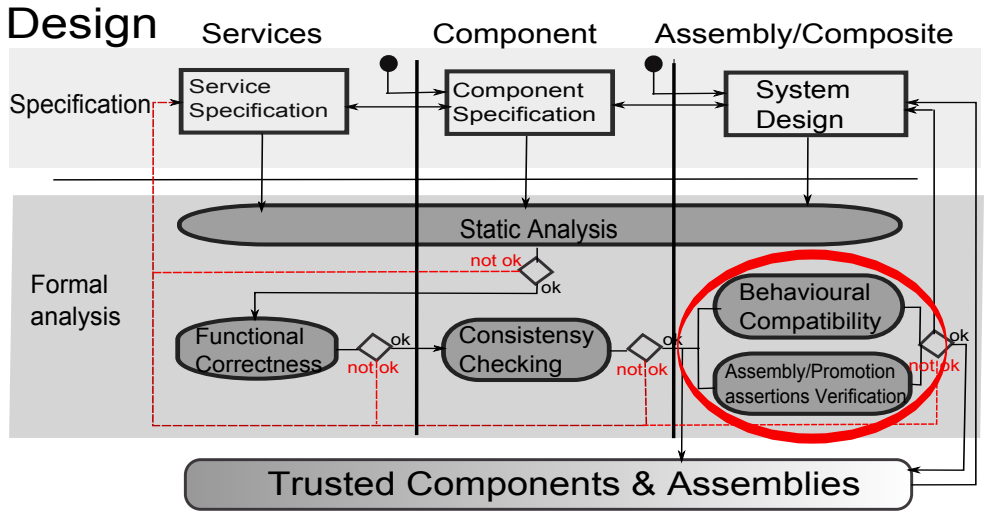


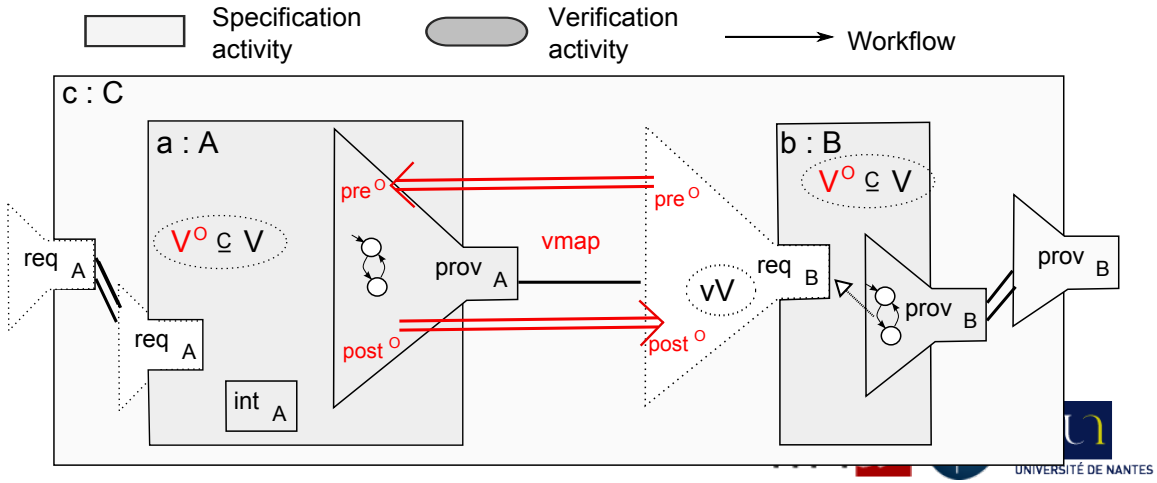


Multi-levels Contracts Design and Verification Process

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.
   -

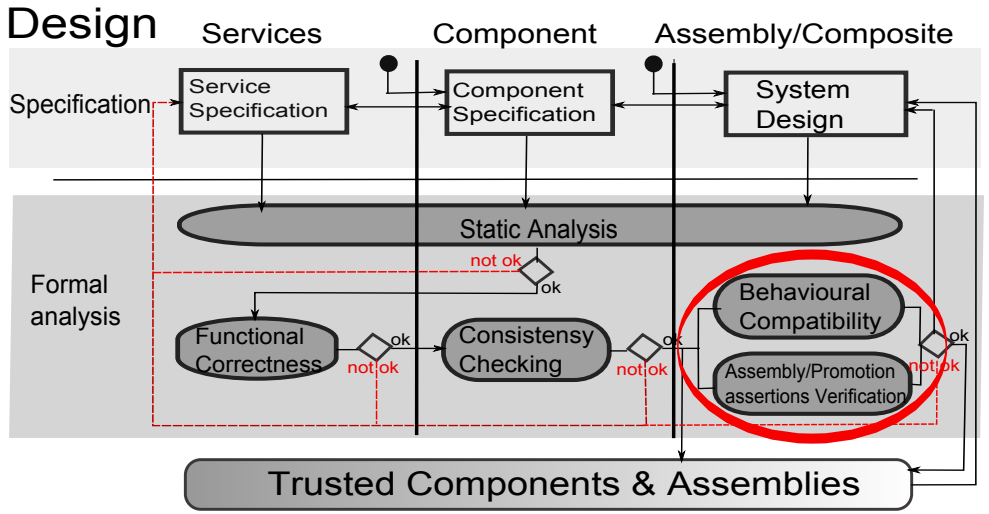


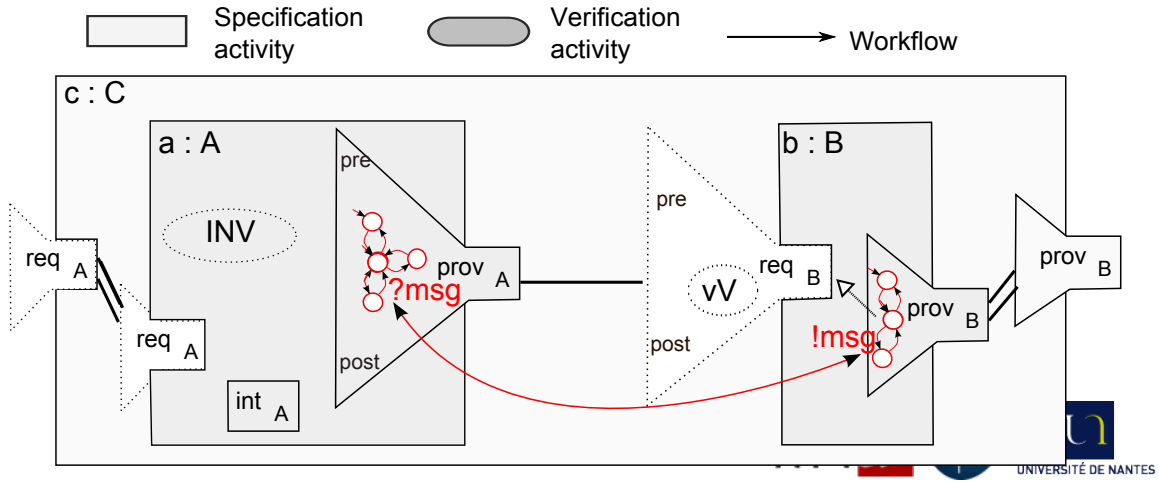


Multi-levels Contracts Design and Verification Process

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

COSTO Tool Overview

COSTO / Kmelia : a Platform to Specify and Verify Component and Service Software

AeLoS Team LINA - UMR 6241 CNRS - Université de Nantes – EMN - France

SPECIFICATION

VERIFICATION

Overview

The COSTO toolbox is an Eclipse platform that consists of:

- a core module with an ANTLR-based parser and an API to access the Kmelia (internal) model,
- several verification and exportation modules,
- a set of eclipse plugins.

Figure 1 shows a sample of the kind of errors (typing, observability, incompleteness of the mapping) that are detected. Besides standard completion, the editor supports smart completion in the case of assembly links.

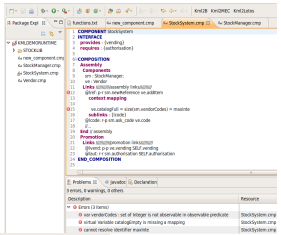
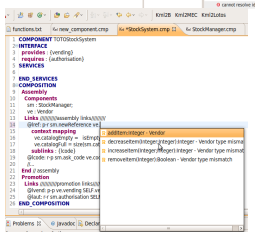


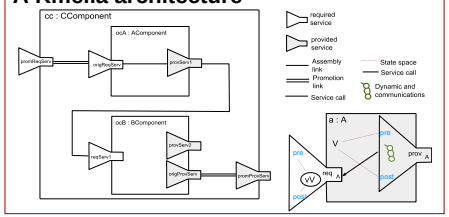
Figure 1. Kmelia editor in Eclipse

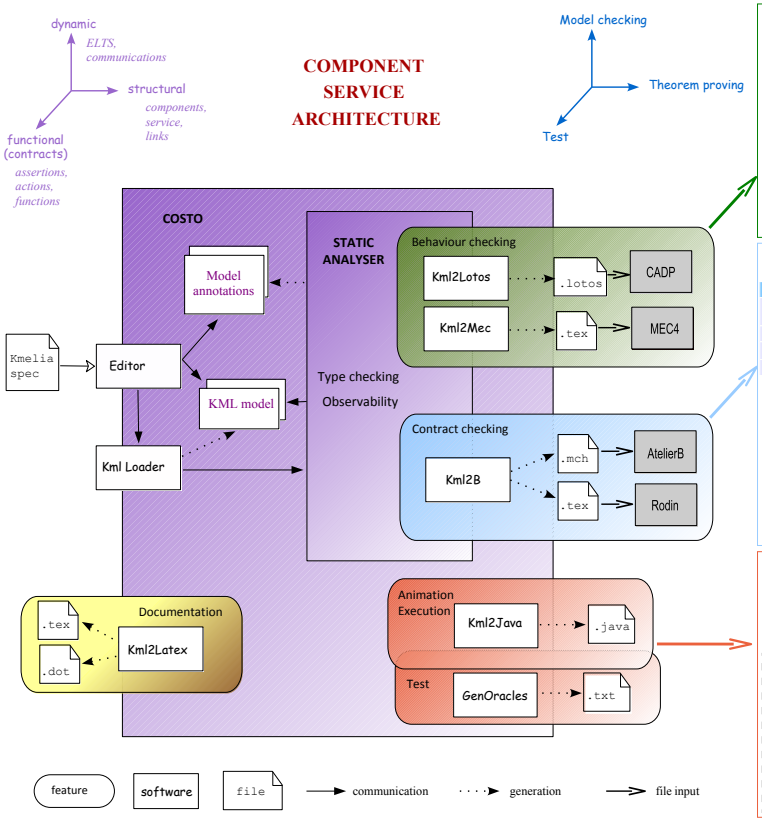


In Figure 2, only required services defined in the `Vendor` component type are proposed and the user is warned that some of them do not match the exact signature of the provided service `new_reference` which is defined in the `StockManager` component type.

Figure 2. Smart completion

A Kmelia architecture





Behaviour checking

Using the exportation buttons on the top of the editor while selecting an assembly link generate models in MEC, LOTOS to be verified in external tools.

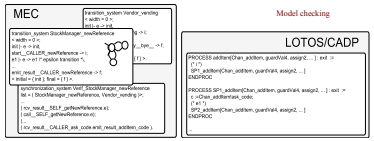
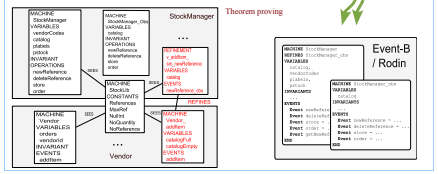


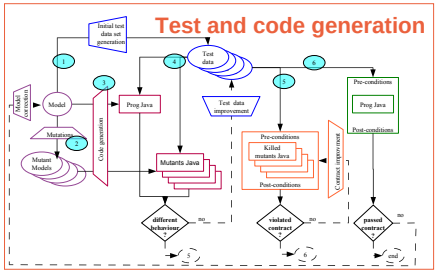
Figure 3. Exportations to MEC/LOTOS

Contract checking

service	component	assembly	composite
behavioural consistency	component consistency	service signature compatibility (ssic)	ssic
functional correctness	protocol correctness	service structure consistency	sstc
	service accessibility	service compliance	sco
		behavioural compatibility	bhc



Test and code generation



Some Publications

- Christian Attiogbé, Pascal André, and Gilles Ardourel. Checking Component Composability. In 5th International Symposium on Software Composition, SC'06, volume 4089 of LNCS. Springer, 2006.
- Pascal André, Gilles Ardourel, and Christian Attiogbé. Dening Component Protocols with Service Composition: Illustration with the Kmelia Model. In 6th International Symposium on Software Composition, SC'07, volume 4829 of LNCS. Springer, 2007.
- Pascal André, Gilles Ardourel, Christian Attiogbé, and Arnaud Lanoix. Using assertions to enhance the correctness of kmelia components and their assemblies. Electronic Notes in Theoretical Computer Science, 263:5 – 30, 2010. Proceedings of the 6th International Workshop on Formal Aspects of Component Software (FACS 2009).
- P. André, G. Ardourel, C. Attiogbé, and A. Lanoix. Contract-based Verification of KmeliaComponent Assemblies using Event-B. In Proceedings of the Formal Foundations of Embedded Software and Component-Based Software Architectures (FESCA 2010), 2010.
- M. Messabihi, P. André, C. Attiogbé. Multi-levels Use of Contracts for Trusted Components. In Workshop on Component and Service Interoperability, 2010. (WCSI 2010) – EPTCS http://arxiv.org/pdf/1010.2827.
- P. André, G. Ardourel, M. Messabihi. Component Service Promotion: Contracts, Mechanisms and Safety. Proceedings of the 7th International Workshop on Formal Aspects of Component Software (FACS 2010) – LNCS Springer.

Key features

- Component and Service
- Required service context
- Contract and Protocols
- Assembly and mappings
- Composite and promotion
- Syntax and type checking
- Model-checking
- Theorem proving
- Test and mutation

For further information

Please contact coloss@univ-nantes.fr

More information on this and related projects can be obtained at http://www.lina.sciences.univ-nantes.fr/coloss/index_en.php.

A PDF-version of the poster is available at http://www.lina.sciences.univ-nantes.fr/coloss/download/posterCosto.pdf

Outline

Conclusion

- Making explicit contract at different levels 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 levels 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 levels 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)

- Fill the missing (action behaviour) (Key, Test...)
- Enable the feedback to the specification step from the results of external tools
- Using contracts for testing component models and codes

Medium term

- Apply to real-size systems.
- Extend to multi-services

Long term

- Implement the whole process
- Apply these ideas and techniques to heterogeneous component and service models

Perspectives

Short term (actually ongoing work)

- Fill the missing (action behaviour) (Key, Test...)
- Enable the feedback to the specification step from the results of external tools
- Using contracts for testing component models and codes

Medium term

- Apply to real-size systems.
- Extend to multi-services

Long term

- Implement the whole process
- Apply these ideas and techniques to heterogeneous component and service models

Perspectives

Short term (actually ongoing work)

- Fill the missing (action behaviour) (Key, Test...)
- Enable the feedback to the specification step from the results of external tools
- Using contracts for testing component models and codes

Medium term

- Apply to real-size systems.
- Extend to multi-services

Long term

- Implement the whole process
- Apply these ideas and techniques to heterogeneous component and service models

Focus sur le test de composants!

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.