Version Propagation in Three-Level Component-Based Architectures

1. Context and objectives

- Versioning models / architectures
  - Representing the whole life-cycle of an application and version its representations → Co-evolution
  - Versioning models / architectures
- Dedal
  - 3 abstraction levels (Figure 1): **Specification** (Roles) / **Configuration** (Component classes) / **Assembly** (Component instances) → Keeping track of the whole life-cycle
  - Changes may occur at any of the 3 architecture levels
- Problematics: Management of co-evolution and versioning of architecture models

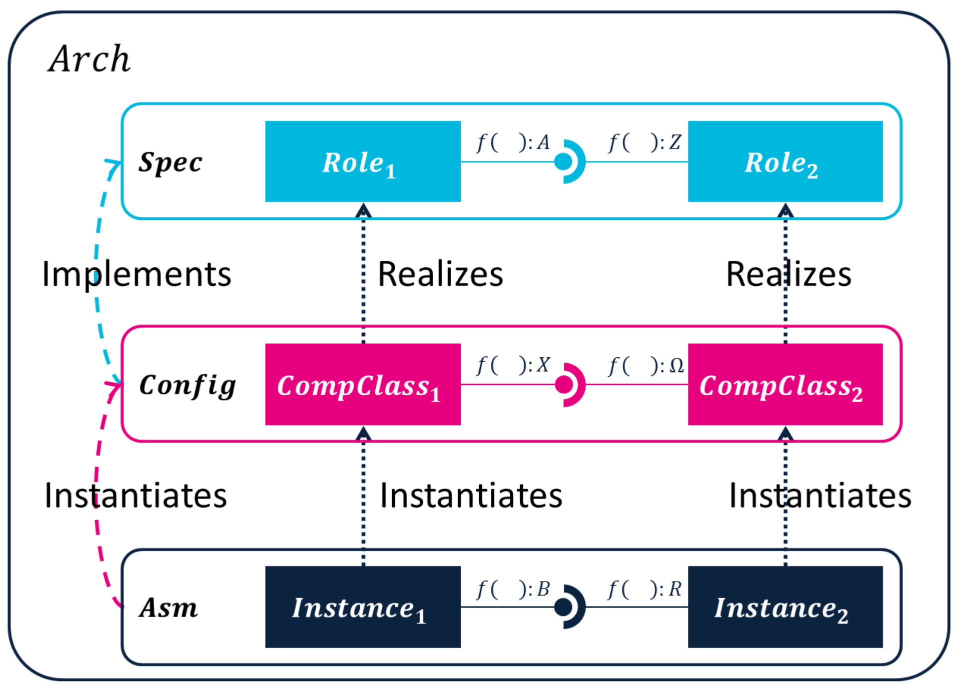


Figure 1. Base case: Dedal three-level architecture

Architectural rules

I. Intra-level consistency

1. Unique name
2. Connected interfaces are compatible
3. The architecture realizes its functional objectives and the architecture definition is composed of a connected graph

II. Inter-level coherence

1. All component roles from the specification are realized by component classes in the configuration
2. Each connected provided interface in the configuration is included in the specification
3. Every component class from the configuration is instantiated at least once by a component instance in the assembly
4. Each connected provider in the assembly is an instance of a provided interface from the configuration

2. Rules for predicting version propagation

Hypothesis on types (Figure 1): $B \preccurlyeq X \preccurlyeq A \preccurlyeq Z \preccurlyeq \Omega \preccurlyeq R$

Provided functionality

Specification	Configuration	Assembly
$Y \rightsquigarrow A$	$Y \rightsquigarrow X$	$Y \rightsquigarrow B$
Non-propagation		
$X \preccurlyeq Y \preccurlyeq Z$	$B \preccurlyeq Y \preccurlyeq A$	$Y \preccurlyeq X$
Propagation		
Inter-level: $(Y \parallel X) \vee (Y \prec X)$ Intra-level: $(Y \parallel Z) \vee (Y \succ Z)$	Inter-level: $(\neg(Y \preccurlyeq A \Rightarrow \uparrow)) \vee (\neg(Y \succcurlyeq B \Rightarrow \downarrow))$ Intra-level: $\neg(Y \preccurlyeq \Omega)$	Inter-level: $\neg(Y \preccurlyeq X)$ Intra-level: $\neg(Y \preccurlyeq R)$
$(Y \parallel X) \wedge (Y \parallel Z)$	$[(\neg(Y \preccurlyeq A)) \vee (\neg(Y \succcurlyeq B))] \wedge [\neg(Y \preccurlyeq \Omega)]$	$\neg(Y \preccurlyeq X)$

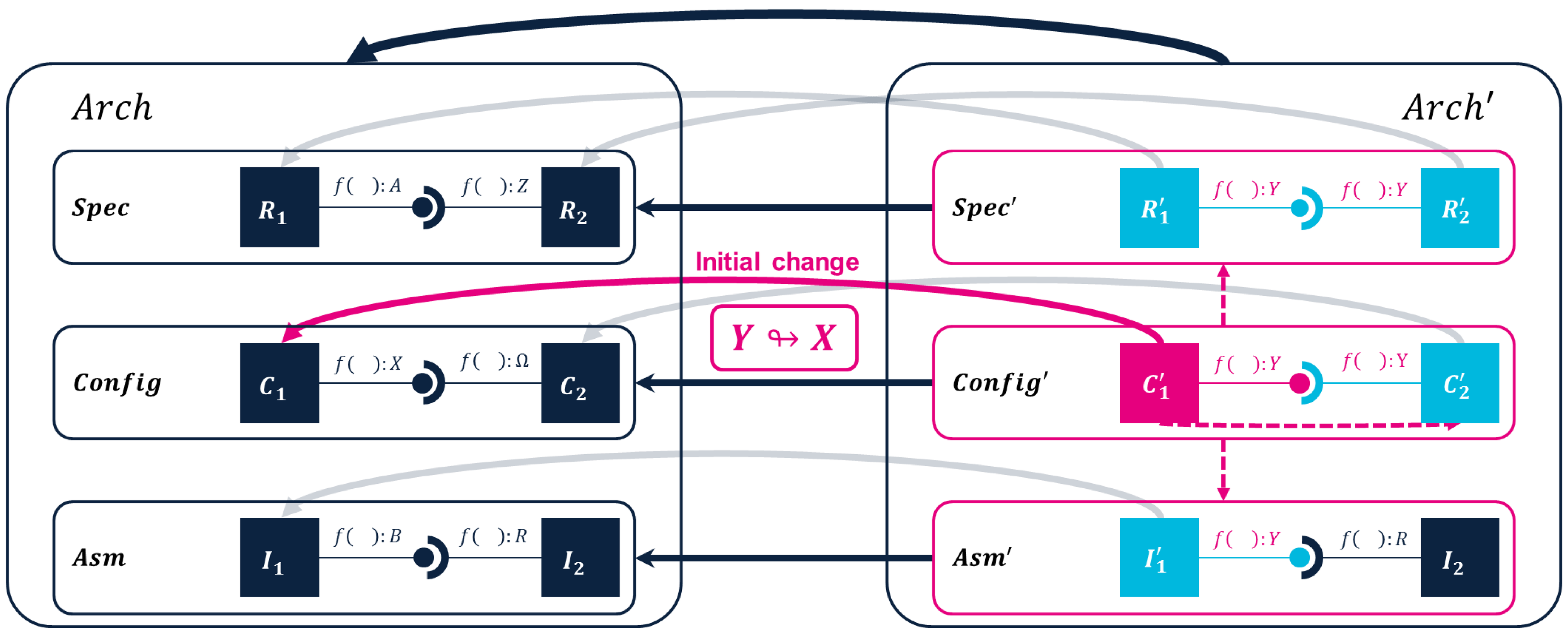
Required functionality

Specification	Configuration	Assembly
$Y \rightsquigarrow Z$	$Y \rightsquigarrow \Omega$	$Y \rightsquigarrow R$
Non-propagation		
$A \preccurlyeq Y \preccurlyeq \Omega$	$Z \preccurlyeq Y \preccurlyeq R$	$Y \succcurlyeq \Omega$
Propagation		
Inter-level: $\neg(Y \preccurlyeq \Omega)$ Intra-level: $\neg(Y \succcurlyeq A)$	Inter-level: $(\neg(Y \succcurlyeq Z \Rightarrow \uparrow)) \vee (\neg(Y \preccurlyeq R \Rightarrow \downarrow))$ Intra-level: $\neg(Y \succcurlyeq X)$	Inter-level: $\neg(Y \succcurlyeq \Omega)$ Intra-level: $\neg(Y \succcurlyeq B)$
$(Y \parallel \Omega) \wedge (Y \parallel A)$	$[(\neg(Y \succcurlyeq Z)) \vee (\neg(Y \preccurlyeq R))] \wedge [\neg(Y \succcurlyeq X)]$	$(\neg(Y \succcurlyeq \Omega)) \wedge (\neg(Y \succcurlyeq B))$

Notations

- $T_1 \prec T_2$: T_1 is a **subtype** of T_2
- $T_1 \preccurlyeq T_2$: T_1 is a **subtype** of T_2 or equal to T_2.
- $T_1 \succ T_2$: T_1 is a **supertype** of T_2.
- $T_1 \succcurlyeq T_2$: T_1 is a **supertype** of T_2 or equal to T_2.
- $T_1 \parallel T_2$: T_1 is not comparable to T_2.
- $\neg(T_1 \preccurlyeq T_2) \Leftrightarrow ((T_1 \succ T_2) \vee (T_1 \parallel T_2))$: T_1 is either a supertype of T_2 or not comparable to T_2.
- $\neg(T_1 \succcurlyeq T_2) \Leftrightarrow ((T_1 \prec T_2) \vee (T_1 \parallel T_2))$: T_1 is either a subtype of T_2 or not comparable to T_2.
- $T_2 \rightsquigarrow T_1$: T_2 replaces T_1.

3. Example of version propagation



$B \preccurlyeq X \preccurlyeq A \preccurlyeq Z \preccurlyeq \Omega \preccurlyeq R$
$(Y \preccurlyeq R) \wedge (Y \parallel \Omega)$
$[(\neg(Y \preccurlyeq A)) \vee (\neg(Y \succcurlyeq B))] \wedge [\neg(Y \preccurlyeq \Omega)]$

← Is a version of

⇠ Version/Change propagation

Figure 5. Propagating version at three architecture levels

4. Generalization

1 to n replacement

- Cases of 1 to n replacement:
  - A role may be realized by n component classes
  - Many roles may be realized by one component class
  - A component class may be instantiated by n component instances
- multiple connected components can be considered as a single composite component

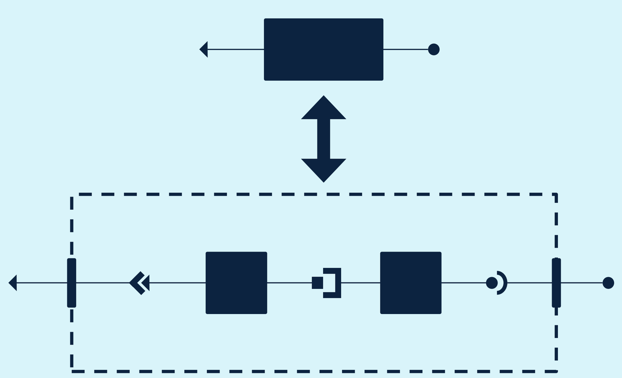


Figure 3. Connected components seen as a single composite component

Multiple connections

- separately study each connection

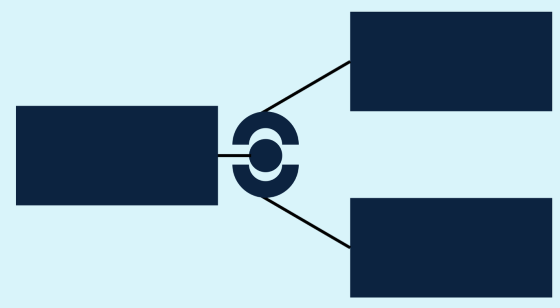


Figure 4. Multiple connections with an interface

Authors

Alexandre Le Borgne *
David Delahaye+
Marianne Huchard +
Christelle Urtado *
Sylvain Vauttier *

5. Conclusion and future work

- Substitutability-based principles for predicting version propagation in three-level component-based architectures
  - Identification of component substitution scenarios
  - component **substitution is not a fine-grained enough criterion** → parameter types into signatures
- Future work
  - Formalization and automation of version propagation

Publications

[SATToSE 2017, Madrid, Spain] A. Le Borgne, D. Delahaye, M. Huchard, C. Urtado, and S. Vauttier, "*Preliminary study on predicting version propagation in three-level component-based architectures*", to appear in *Proceedings of the 7th Seminar on Advanced Techniques & Tools for Software Evolution*, 2017.

[SEKE 2017, Pittsburgh, USA] A. Le Borgne, D. Delahaye, M. Huchard, C. Urtado, and S. Vauttier, "*Substitutability-Based Version Propagation to Manage the Evolution of Three-Level Component-Based Architectures*", to appear in *Proceedings of the 29th International Conference on Software Engineering & Knowledge Engineering*, 2017.

* IMT – Mines Alès
{ Alexandre.Le-Borgne, Christelle.Urtado, Sylvain.Vauttier } @mines-ales.fr

+ LIRMM / CNRS & Montpellier University, France
{ David.Delahaye, Marianne.Huchard } @lirmm.fr

Juin 2017 – GDR GPL, Montpellier