Des modèles aux programmes à base de composants

Besoin de langages à composants

Luc Fabresse

École des mines de Douai

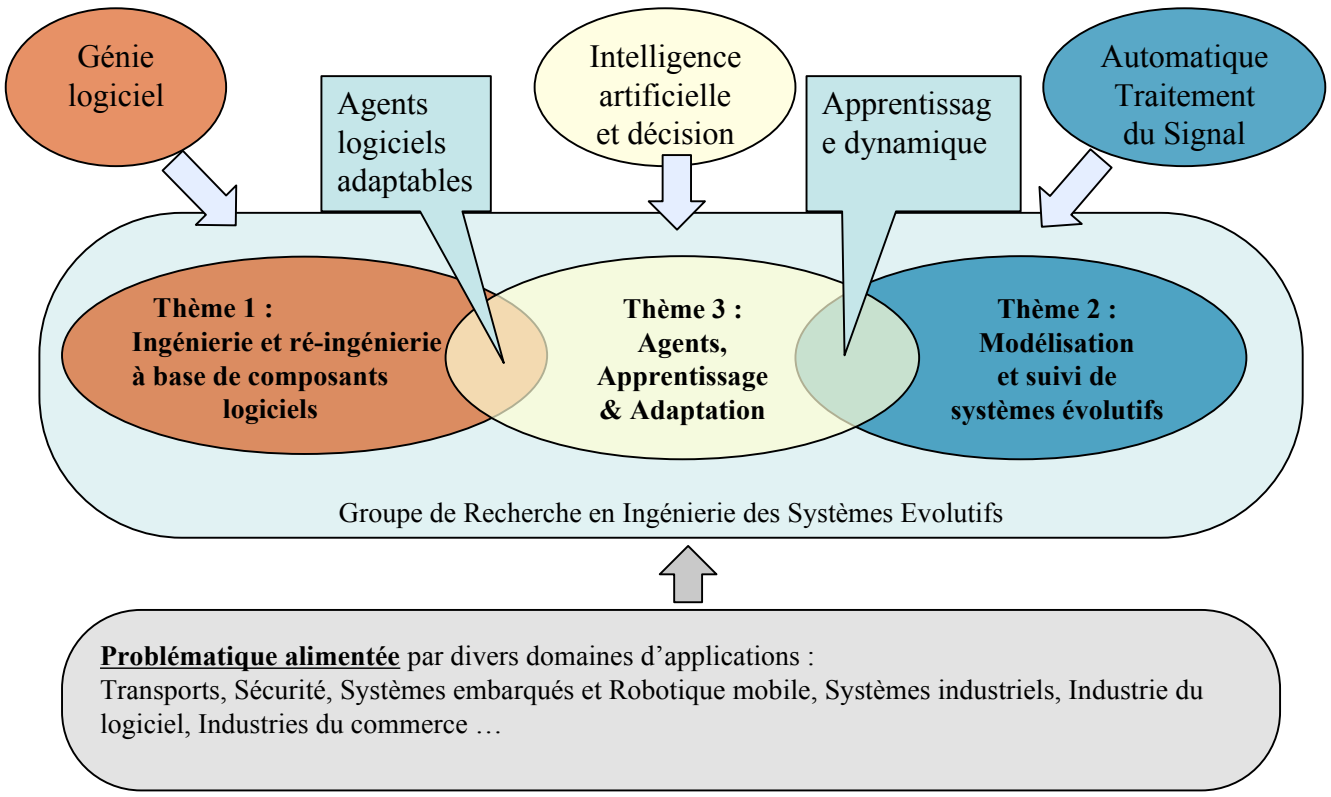
Unité de Recherche Informatique et Automatique

http://vst.ensm-douai.fr/research/

Groupe de travail COSMAL

27 janvier 2009

URIA : Thèmes de recherche - Projet



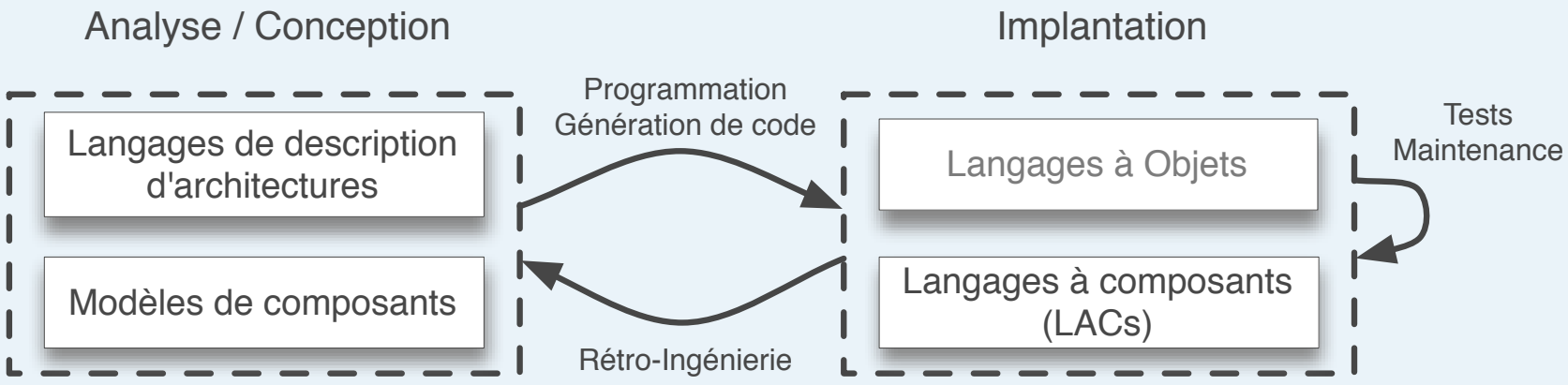
Plan

Plan

De l'utilité des langages à composants (LACs)

Programmation par composants (PPC) : 4 techniques

- Langage de programmation + conventions (Javabeans)
- Extension d'un *framework* Objet (EJB)
- Transformation de modèles (Fractal → Julia)
- Langages à composants (ArchJava)



Constats et objectifs

Constats

- Mise en pratique difficile de la PPC (peu de LACs)
- Prise en compte partielle du découplage et de la non-anticipation
- Diversité des concepts
- Les LACs sont généralement sous-spécifiés et non-uniformes

Notre objectif : réaliser un langage de programmation

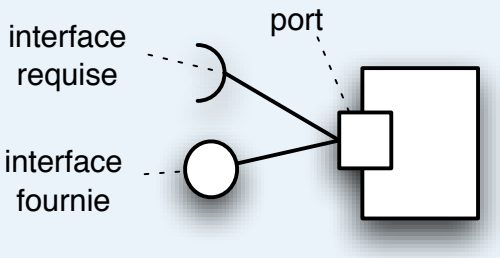
- Intégrant les concepts et mécanismes nécessaires et suffisants pour faire de la PPC (Minimal)
- Prenant en compte le découplage et la non-anticipation
- N'entretenant pas la confusion objets/composants (Uniforme)
- Offrant des mécanismes de haut niveau (Simple)

Plan

Plan

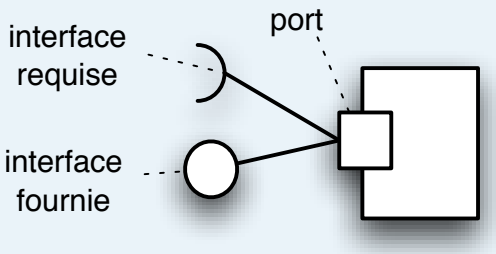
Les concepts de « port » et d'« interface »

UML

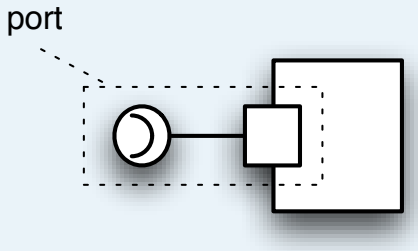


Les concepts de « port » et d'« interface »

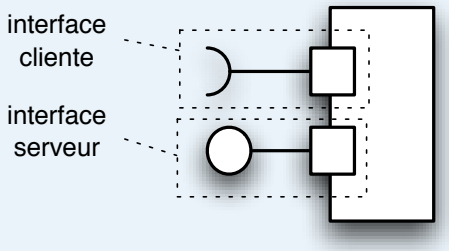
UML



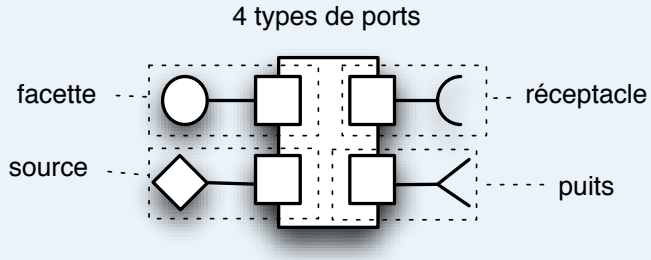
ArchJava



Fractal



CCM

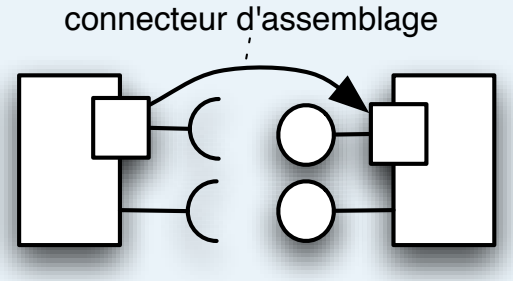


Diversité des concepts

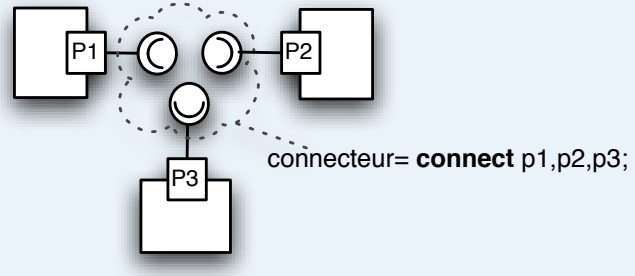
- Communications unidirectionnelles / bidirectionnelles
- Nombre d'interfaces par port

Les concepts de « connexion » et de « connecteurs »

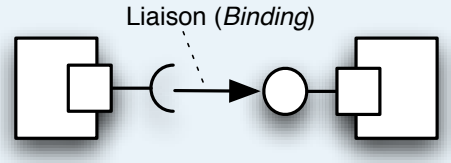
UML



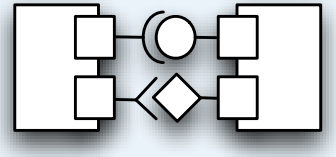
ArchJava



Fractal



CCM



Diversité des concepts

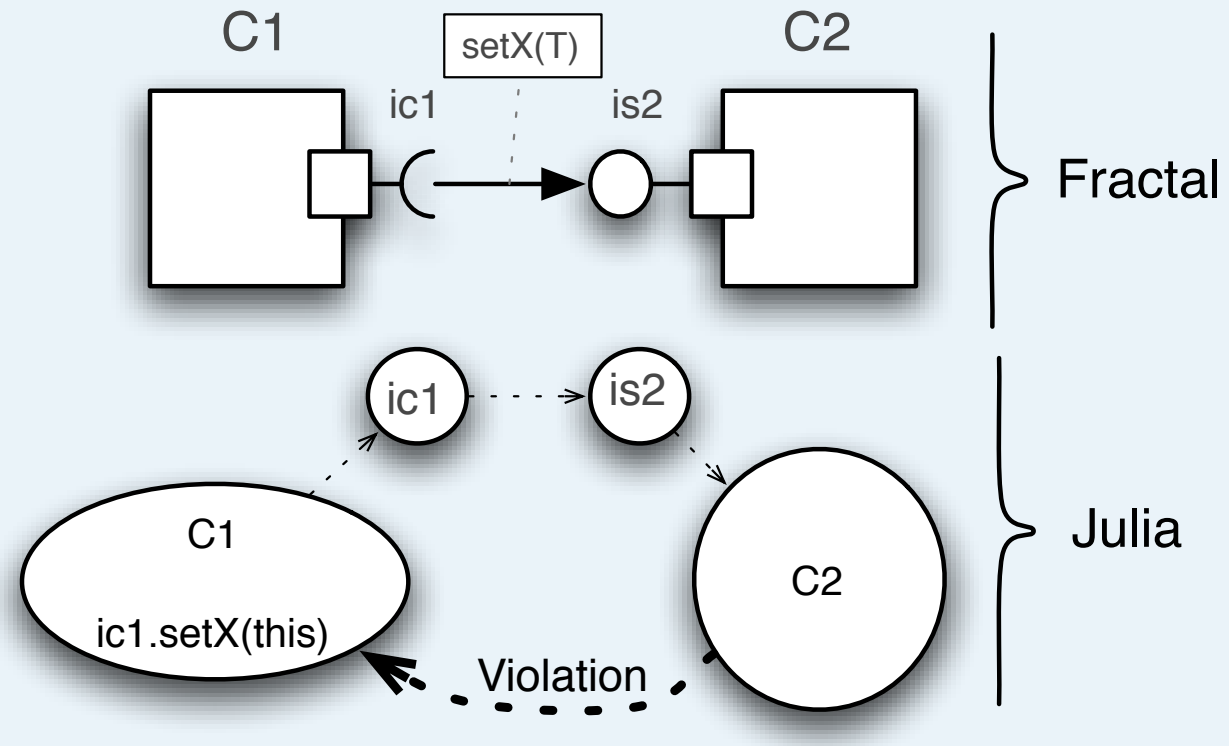
- Que connecte-t-on : les ports et/ou les interfaces ?
- Les connexions sont-elles binaires et/ou n-aires ?
- Les connexions sont-elles réifiées (connecteurs) ?

Plan

Problème de l'échange de références entre objets

Illustration en Julia

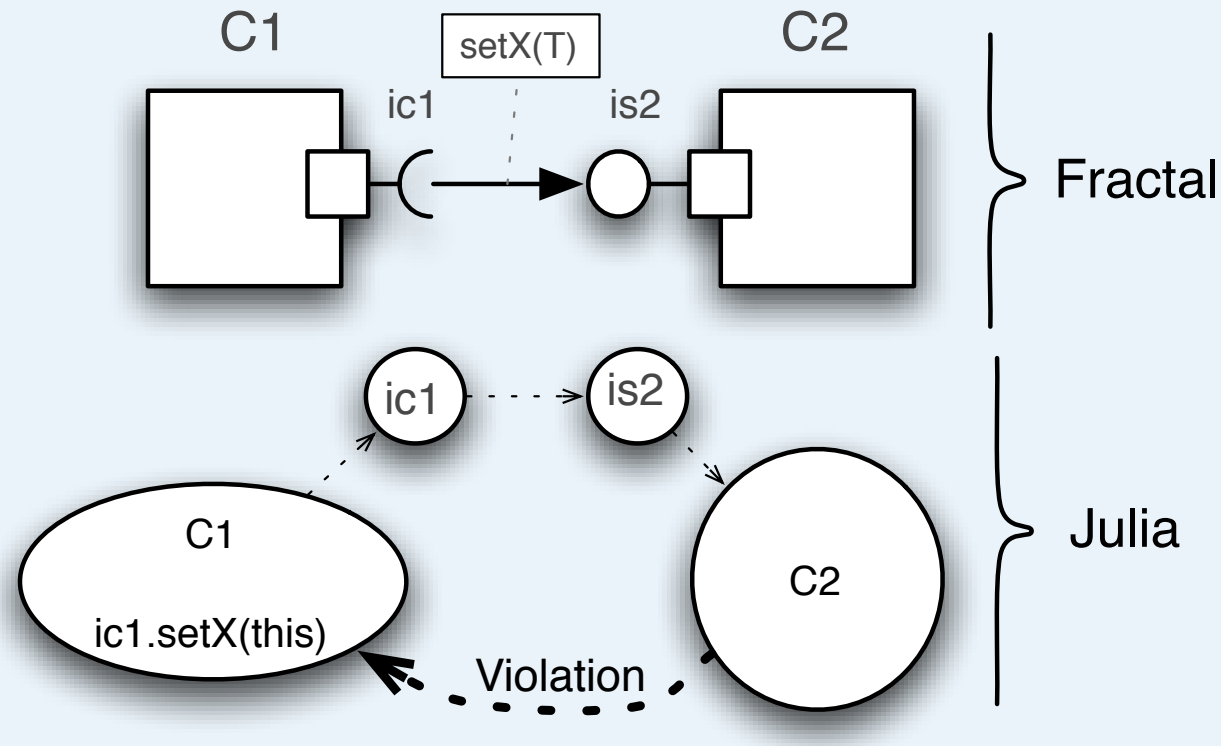
Violation de l'intégrité des communications en Julia [LCL06]



Problème de l'échange de références entre objets

Illustration en Julia

Violation de l'intégrité des communications en Julia [LCL06]



Sous-spécification

Comment s'effectue le passage d'arguments ?

Synthèse comparative

	Julia	ArchJava
Découplage	☺	☹
Non-anticipation	☹	☹
Passage d'arguments	☹	☹
Auto-référence	☹	☹
Uniformité	☹	☹

Plan

Synthèse des problématiques traitées

- Faut-il des descripteurs de composants ? oui (Choix 1)
- Que mettre sur étagère : les composants ou les descripteurs ? les descripteurs (Choix 2)
- Assemble t-on les composants et/ou les descripteurs ? les composants (Choix 3)
- Faut-il des ports unidirectionnels ou bidirectionnels ? unidirectionnels (Choix 4)
- À quoi associe t-on une interface ? un port (Choix 5)
- Que décrit une interface ? un ensemble de signatures de services (Choix 6)
- Comment représenter les types de base ? par des composants (Choix 7)
- Comment uniformiser le traitement des types de bases ? introduction d'un port par défaut (Choix 8)
- Comment connecter des composants ? liaison binaire, port requis $\longrightarrow$ port fourni (Choix 9)
- Comment gérer les collections ? introduction des « multi-ports » (Choix 10)
- Comment invoquer un service ? toujours à travers un port (Choix 11)
- Comment traiter les auto-références ? introduction du port `Self` (Choix 12)
- Que peuvent-être les arguments ? des références sur des ports (Choix 13)
- Comment traiter le passage d'arguments ? par des (dé)connexions automatiques (Choix 14)
- Faut-il réifier les connexions ? oui (Choix 15)
- Qu'est-ce qu'un connecteur ? un composant (Choix 16)
- Comment faciliter l'écriture des connecteurs ? introduction d'un service par défaut (« code glue ») (Choix 17)
- Comment réutiliser des connecteurs ? standardisation des connecteurs et de leur paramétrage (Choix 18)
- Faut-il des composites ? oui (Choix 19)
- Peut-on imbriquer les descriptions de composants ? non (Choix 20)
- Comment référencer les sous-composants ? par des ports internes requis
- Comment traiter les invocations de services en tenant compte de tous les choix ? (Algorithme 1-2)

Synthèse des problématiques traitées

- Faut-il des descripteurs de composants ? oui (Choix 1)
- Que mettre sur étagère : les composants ou les descripteurs ? les descripteurs (Choix 2)
- Assemble t-on les composants et/ou les descripteurs ? les composants (Choix 3)
- Faut-il des ports unidirectionnels ou bidirectionnels ? unidirectionnels (Choix 4)
- À quoi associe t-on une interface ? un port (Choix 5)
- Que décrit une interface ? un ensemble de signatures de services (Choix 6)
- Comment représenter les types de base ? par des composants (Choix 7)
- Comment uniformiser le traitement des types de bases ? introduction d'un port par défaut (Choix 8)
- Comment connecter des composants ? liaison binaire, port requis $\longrightarrow$ port fourni (Choix 9)
- Comment gérer les collections ? introduction des « multi-ports » (Choix 10)
- Comment invoquer un service ? toujours à travers un port (Choix 11)
- **Comment traiter les auto-références ?** introduction du port `Self` (Choix 12)
- Que peuvent-être les arguments ? des références sur des ports (Choix 13)
- Comment traiter le passage d'arguments ? par des (dé)connexions automatiques (Choix 14)
- Faut-il réifier les connexions ? oui (Choix 15)
- Qu'est-ce qu'un connecteur ? un composant (Choix 16)
- Comment faciliter l'écriture des connecteurs ? introduction d'un service par défaut (« code glue ») (Choix 17)
- Comment réutiliser des connecteurs ? standardisation des connecteurs et de leur paramétrage (Choix 18)
- Faut-il des composites ? oui (Choix 19)
- Peut-on imbriquer les descriptions de composants ? non (Choix 20)
- Comment référencer les sous-composants ? par des ports internes requis
- Comment traiter les invocations de services en tenant compte de tous les choix ? (Algorithme 1-2)

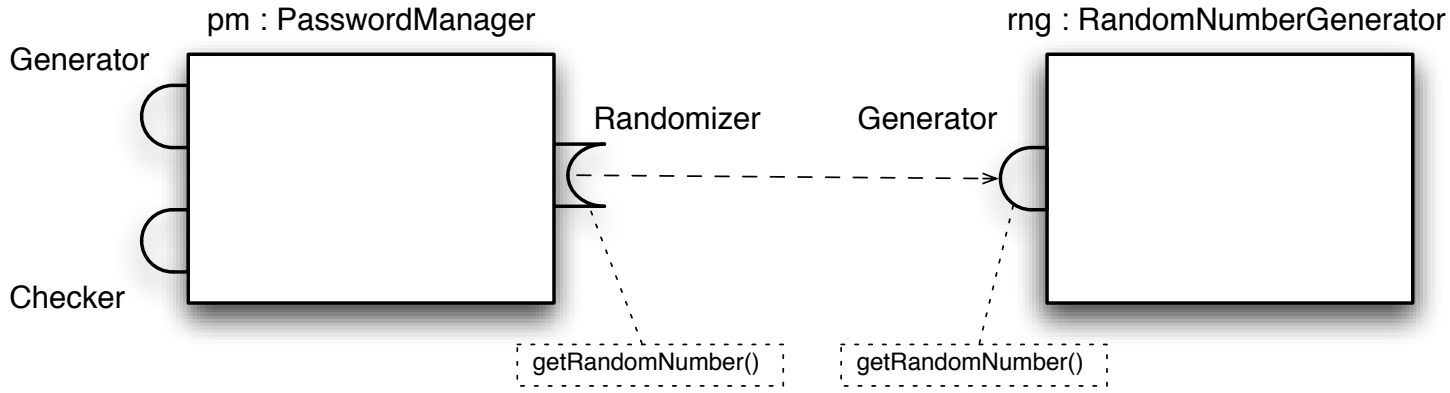
Synthèse des problématiques traitées

- Faut-il des descripteurs de composants ? oui (Choix 1)
- Que mettre sur étagère : les composants ou les descripteurs ? les descripteurs (Choix 2)
- Assemble t-on les composants et/ou les descripteurs ? les composants (Choix 3)
- Faut-il des ports unidirectionnels ou bidirectionnels ? unidirectionnels (Choix 4)
- À quoi associe t-on une interface ? un port (Choix 5)
- Que décrit une interface ? un ensemble de signatures de services (Choix 6)
- Comment représenter les types de base ? par des composants (Choix 7)
- Comment uniformiser le traitement des types de bases ? introduction d'un port par défaut (Choix 8)
- Comment connecter des composants ? liaison binaire, port requis $\longrightarrow$ port fourni (Choix 9)
- Comment gérer les collections ? introduction des « multi-ports » (Choix 10)
- Comment invoquer un service ? toujours à travers un port (Choix 11)
- Comment traiter les auto-références ? introduction du port `Self` (Choix 12)
- Que peuvent-être les arguments ? des références sur des ports (Choix 13)
- **Comment traiter le passage d'arguments ?** par des (dé)connexions automatiques (Choix 14)
- Faut-il réifier les connexions ? oui (Choix 15)
- Qu'est-ce qu'un connecteur ? un composant (Choix 16)
- Comment faciliter l'écriture des connecteurs ? introduction d'un service par défaut (« code glue ») (Choix 17)
- Comment réutiliser des connecteurs ? standardisation des connecteurs et de leur paramétrage (Choix 18)
- Faut-il des composites ? oui (Choix 19)
- Peut-on imbriquer les descriptions de composants ? non (Choix 20)
- Comment référencer les sous-composants ? par des ports internes requis
- Comment traiter les invocations de services en tenant compte de tous les choix ? (Algorithme 1-2)

Connexion de composants

Liaison binaire : un port requis → un port fourni (Choix 9)

- Suffisamment expressif
- Moins d'ambiguïtés (ex : ArchJava)



La primitive : **bind** <port requis> **to** <port fourni>

```
pm := new PasswordManager
rng := new RandomNumberGenerator
bind pm.Randomizer to rng.Generator
```

Connexion et connecteur

Nécessité des connecteurs (Choix 15)

- Séparer le code métier et le code de l'interaction
- Traiter les adaptations dans les connecteurs

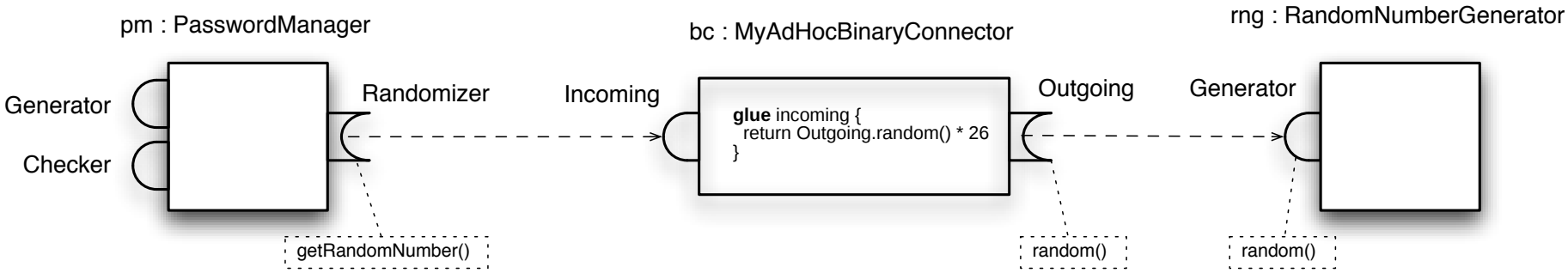
Un connecteur est un composant (Choix 16)

- Uniformité
- Produire des connecteurs réutilisables et paramétrables

Associer du « code glue » à un port fourni (Choix 17)

- Service particulier associé à un port fourni
- Exécuté en cas d'absence du service demandé
- Accès à l'invocation de service courante (pseudo-variable **si**)

Connexion via un connecteur

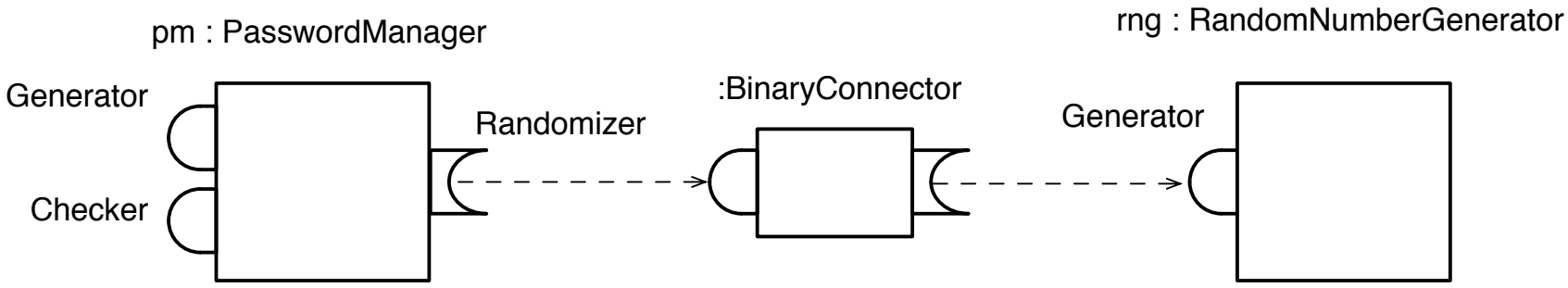


La primitive : **bind** <pr> **to** <pf> **glue** {...}

```
bind pm.Randomizer to rng.Generator glue {
     return Outgoing.random() * 26
}
```

Séparation des préoccupations transversales

Exemple d'une fonctionnalité de trace

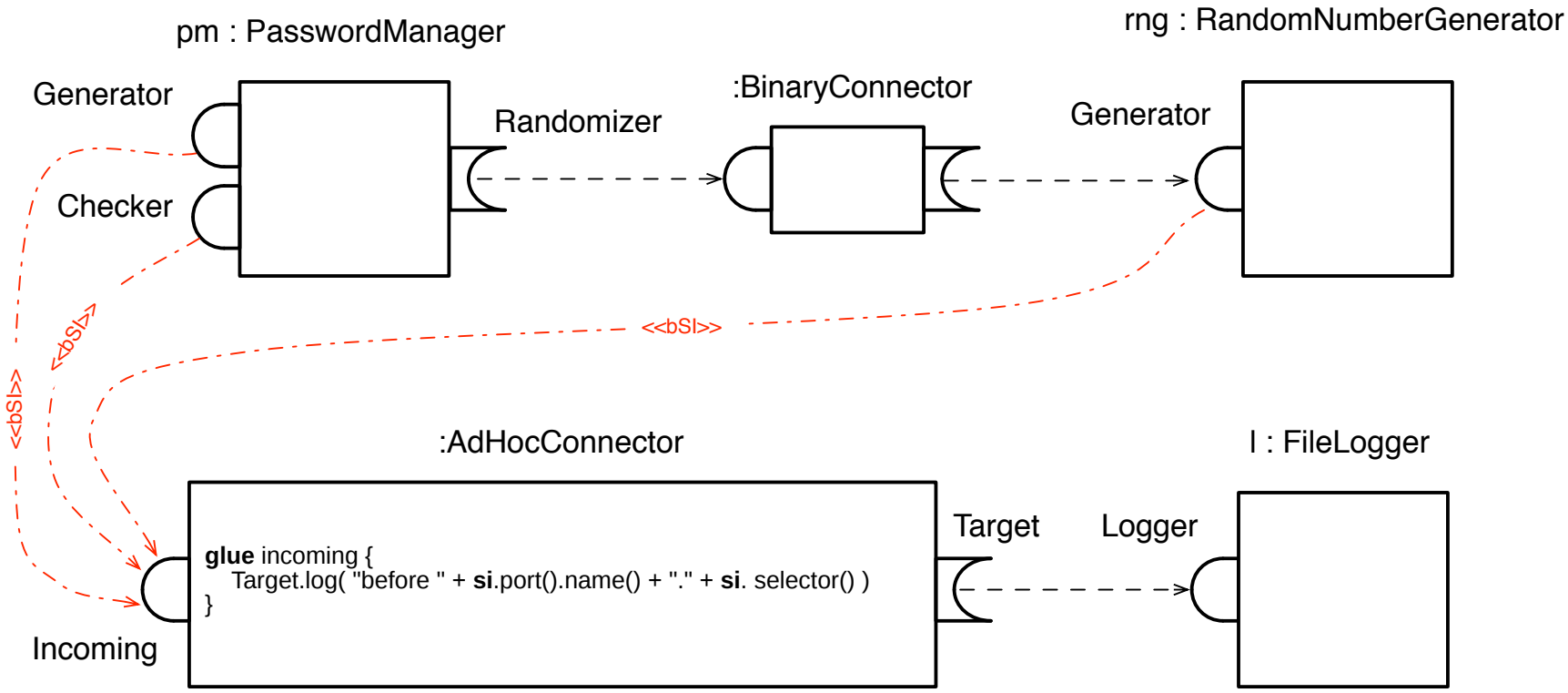


La primitive : **bindbsi** <ports> **to** <port> **glue** {...}

```
bindbsi pm.Generator, pm.Checker, rng.Generator to l.Log glue {
     return Target.log( " before " + si.port().name() + "." + si.selector())
}
```

Séparation des préoccupations transversales

Exemple d'une fonctionnalité de trace



La primitive : **bindbsi** <ports> **to** <port> **glue** {...}

```
bindbsi pm.Generator, pm.Checker, rng.Generator to l.Log glue {
     return Target.log( " before " + si.port().name() + "." + si.selector())
}
```

Code Scl écrit avec le prototype en Smalltalk

```
(SclBuilder newDescriptor: #PasswordManager)
    requiredPort: #Randomizer -> {#getRandomNumber};
    providedPort: #Generator -> {#generatePwd:. #generateADigitsOnlyPwd:};
    providedPort: #Checker -> {#isValid:}.

PasswordManager>>generatePwd: size
    "..."
    i:= Randomizer getRandomNumber
    "..."

"connexion"
pm := PasswordManager newC.
rng := RandomNumberGenerator newC.

(pm port: #Randomizer) bindTo: (rng port: #Generator)
glue: [ :source :target :si |
    ^target random * 26
]
```

Quelques détails d'implantation

- Utilisation de l'interception des envois de messages
- Même technique que les *Domain Specific Languages*
- Primitives **newC**, **bindTo:**, **bindTo:glue:**

Plan

Conclusion

Analyse comparative de l'existant

- Diversité des concepts
- Mise en pratique difficile de la PPC
- Sous-spécifications et non-uniformité des LACs

SCL : synthèse comparative

	Julia	ArchJava	SCL
Passage d'arguments	☹	😟	☺
Uniformité	😟	☹	☺
Auto-référence	☹	☹	☺
Découplage	☺	😟	☺
Séparation des préoccupations	😟	😟	☺
Non-anticipation	😟	😟	☺

Perspectives

- Relations avec les Services Web
- Proposer une version réflexive de SCL
- Un LAC pour les systèmes embarqués

Nao

WifiBot

Merci pour votre attention !

Publications

Luc Fabresse

Du découplage à l'assemblage non-anticipé de composants : Conception et mise en œuvre du langage à composants SCL.
Thèse de doctorat de l'université Montpellier 2, 2007. http ://www.lirmm.fr/~fabresse/.

Luc Fabresse, Christophe Dony, et Marianne Huchard.

Foundations of a Simple and Unified Component-Oriented Language.
Journal of Computer Languages, Systems & Structures, 2007. doi :10.1016/j.cl.2007.05.002.

Luc Fabresse, Christophe Dony, et Marianne Huchard.

SCL : a Simple, Uniform and Operational Language for Component-Oriented Programming in Smalltalk.
In *Advances in Smaltalk, Proceedings of 14th International Smalltalk Conference (ISC), September 4–8, 2006*, éditeur De Meuter, Wolfgang, volume 4406, pages 91–110. LNCS, Springer-Verlag, April 2007.

Luc Fabresse, Christophe Dony, et Marianne Huchard.

Unanticipated Connection of Components based on their State Changes Notifications.
In *International Workshop on Evaluation and Evolution of Component Composition (EECC'06). In proceedings of 18th International Conference on Software Engineering and Knowledge Engineering (SEKE 2006)*, pages 702–708. Knowledge System Institute (KSI), 5–7 July 2006.

Luc Fabresse, Christophe Dony, et Marianne Huchard.

Connexion non-anticipée de composants en SCL : Une voie pour l'évolution des logiciels.
In *Atelier sur l'Evolution du Logiciel*, pages 1–7, 21 Mars 2006.

Luc Fabresse.

Programmation de composants interfaçables.
In *Actes de JOCM, Journées du groupe Objets, Composants et Modèles*, pages 19–24, 16 Mars 2004.

Luc Fabresse, Christophe Dony, Marianne Huchard, et Olivier Pichon.

Vers des composants logiciels interfaçables.
In *8ème Colloque Agents Logiciels, Coopération, Apprentissage et Activité humaine (ALCAA'04)*, pages 33–48, 17–18 Juin 2004.