Quelques modèles, langages et méthodes pour promouvoir le développement par et pour la réutilisation dans l'ingénierie du logiciel

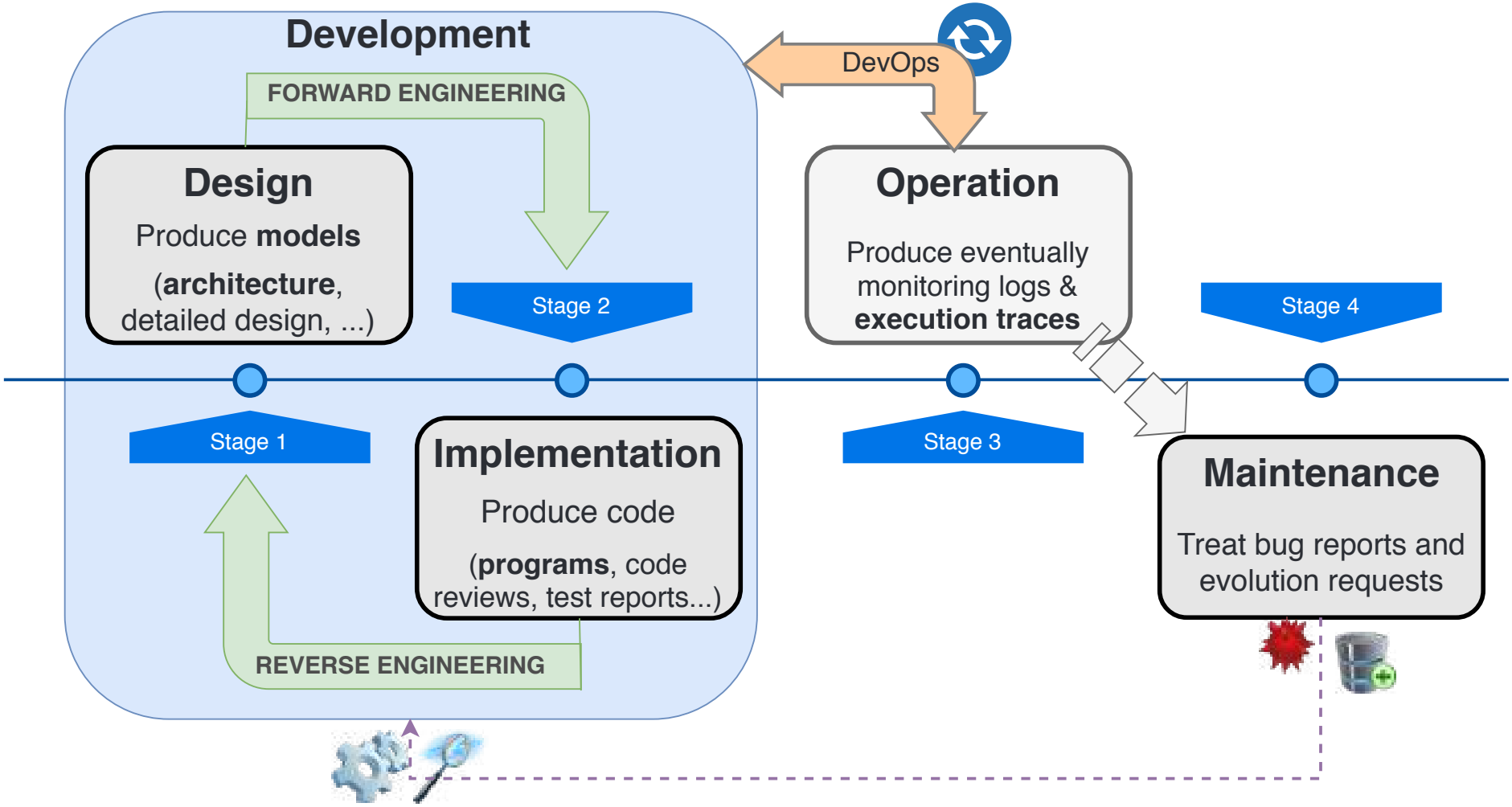
Séminaire LATECE-UQAM – 24 mars 2021

Chouki Tibermacine

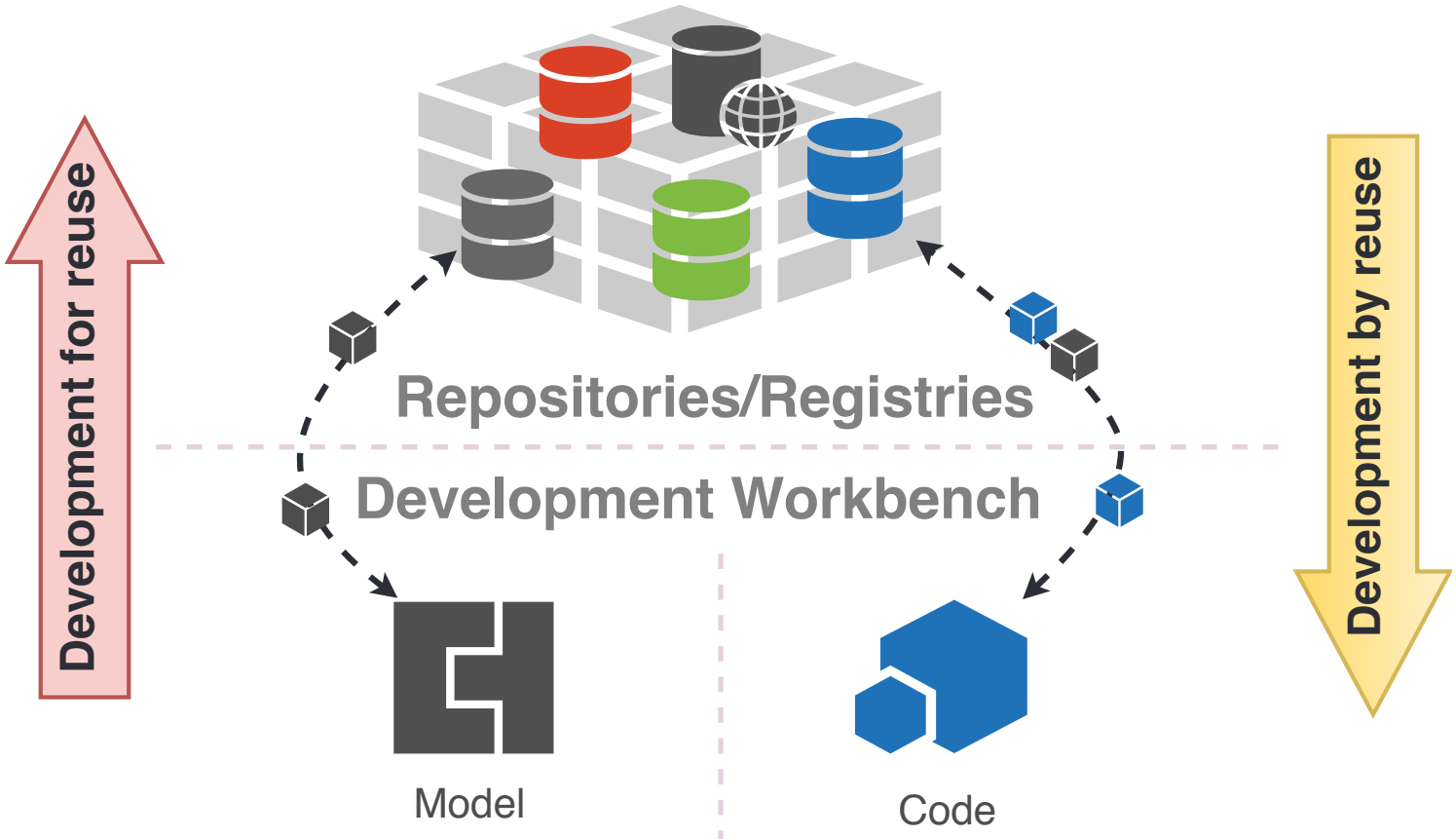
Outline

Outline

A Typical Macro-Process for Software Engineering



(Continuous) Development for- and by- Reuse



Components = first-class entities in dev. for- and by- reuse

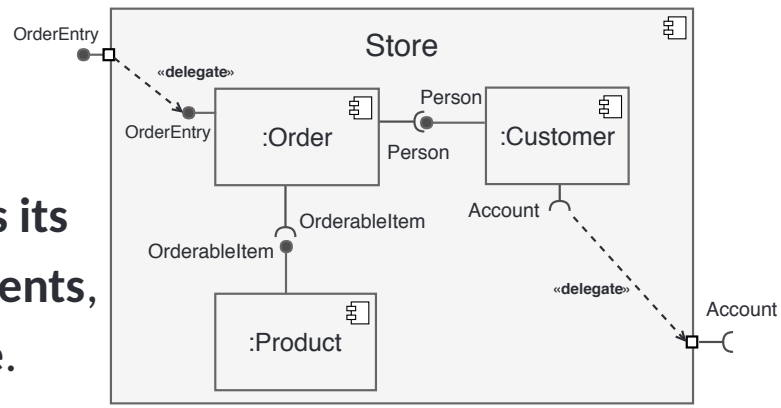
- push & pull software **components** to/from a repository
- register & consume **services** to/from registries

What is a Software Component?

- Term appeared first in McIlroy's paper in 1968[1]
- Concept overloaded with definitions
- My definition[2]:

  A Component is a **part** of a software system (part of its model or code) which **encapsulates its content**, **declares its provisions and requirements**, and may have an **explicit internal architecture**.

  This makes it an easily reusable unit



- A Component conforms to a Component Model

---

[1] Mass Produced Software Components. D. McIlroy. NATO conference on software engineering. Garmisch, Germany, 1968.

[2] Just to not make a YACSD (Yet Another Citation of Szyperski's Definition).

What is a Software Component?

Plethora of component models and their implementations (see survey of component models[3])

- Architecture Modeling Languages (ADLs) since early 90s (UML and its component & composite structure models, for ex.)
- Many Programming languages and frameworks, like:
  - OSGi Java Framework and its Bundle System (May 2000)
  - Spring Java Framework and its Bean System with DI (June 2003)
  - Module System in Java 9+ (Sept. 2017)
- (Web) service = distributed running component (accessible on the Web)

[3] A Classification Framework for Component Models. I. Crnkovic et. al., IEEE Trans. on Software Engineering 37(5), 2010. pp 593–615

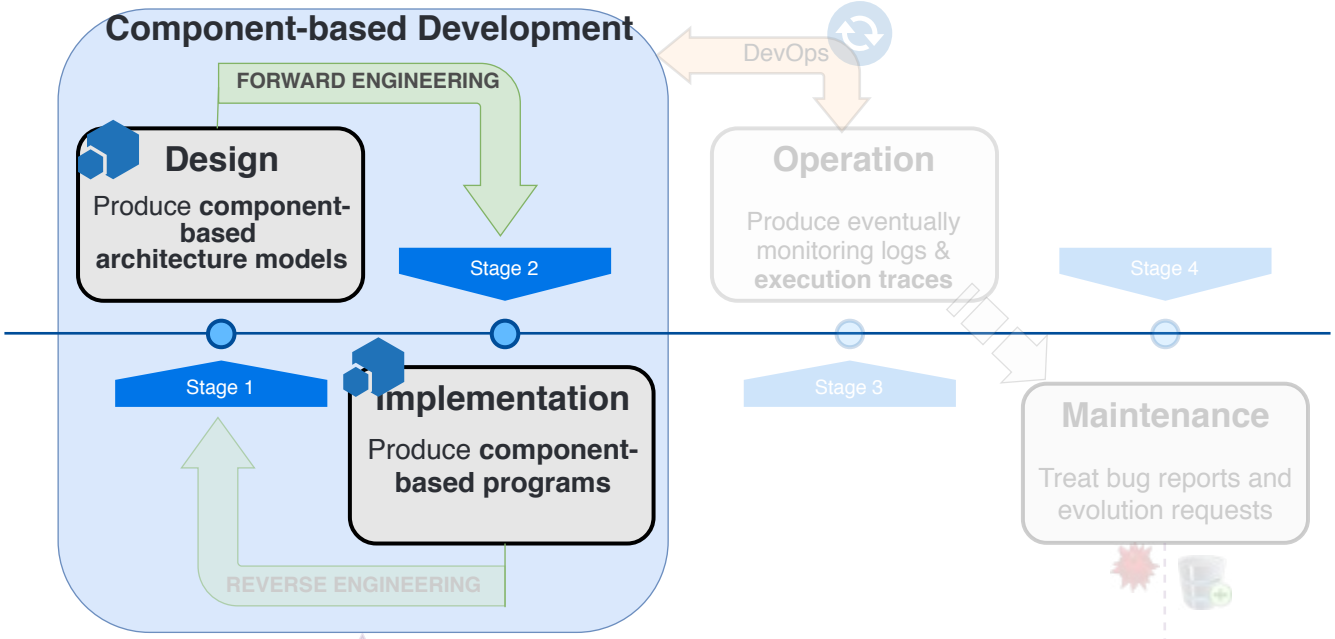
General Research Questions

How to promote the **software component** concept in some software engineering activities?

- How the **architecture model** of a software system can be made **explicit and reusable** in software artifacts of both Design and Implementation activities?
- How to evolve the **code** of existing software systems in a Maintenance activity in order to **make it more reusable**?
- How development-by-reuse in an Implementation activity can be improved through sophisticated processes for the **classification** of artifacts in **repositories**?

Outline of Contributions

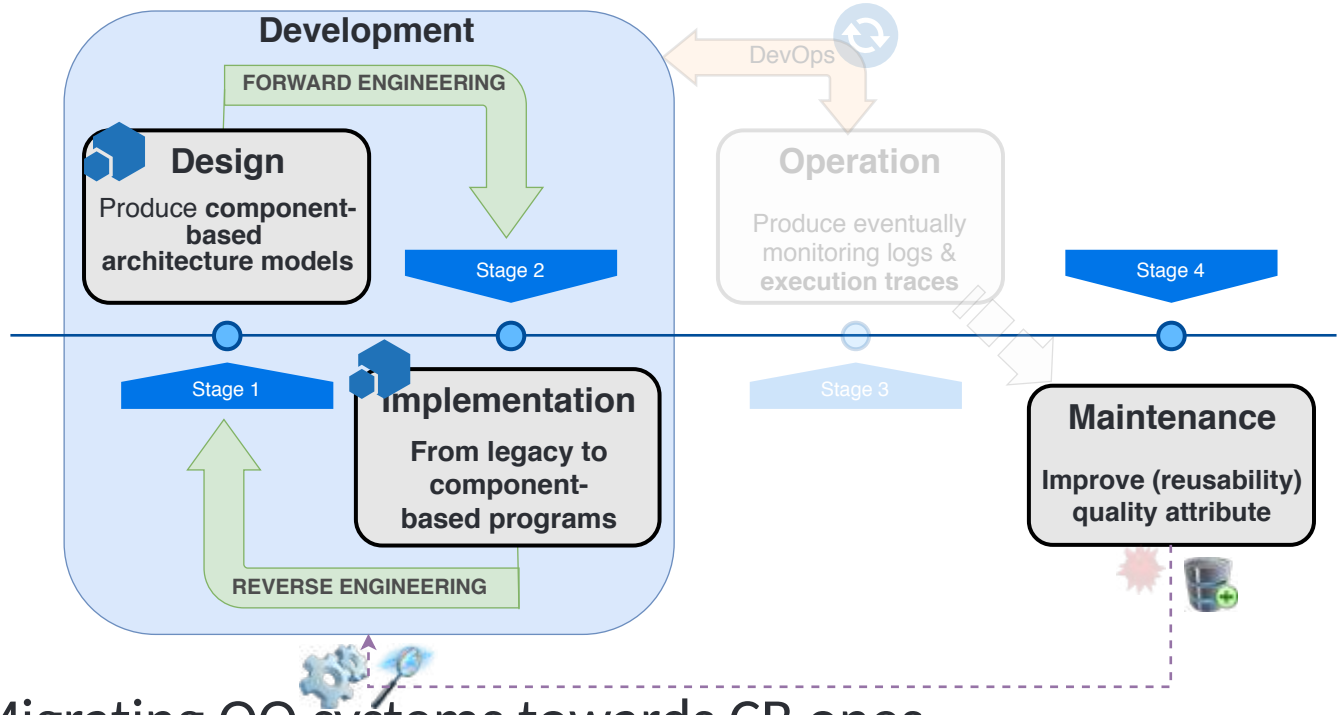
Conducted Research #1: Component-based Architecture Description in Reusable Models and Programs



1. **Design:** CB Specification of Architecture Constraints
2. **Implem.:** Programming Reusable Comp. and Arch. Constraints
3. **Bridging the gap between the two:** Translation of Architecture Constraints into Reusable Meta-Programs

Outline of Contributions -Ctd-

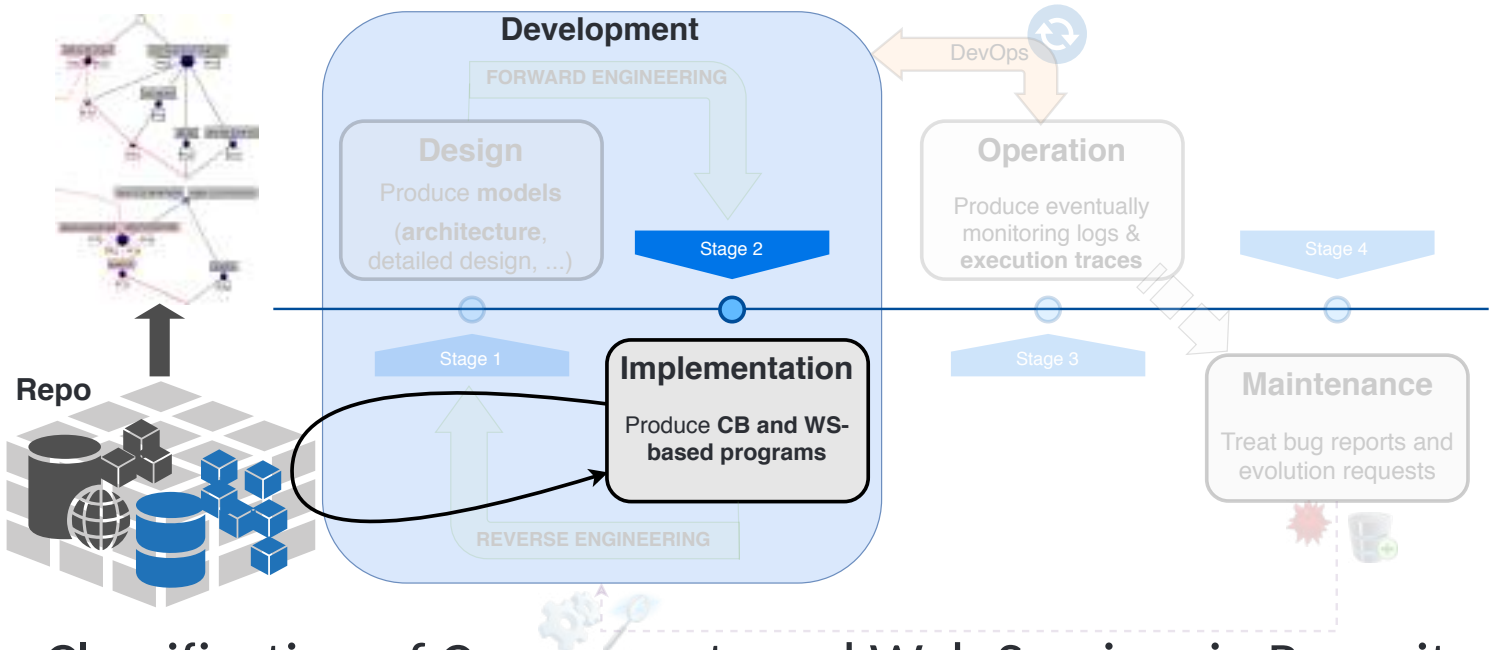
Conducted Research #2: Re-engineering and Evolution of Legacy Applications for improving Reusability



1. Migrating OO systems towards CB ones
2. Migrating Web Apps towards Web Service-oriented ones
3. Managing the Evolution of Web service-oriented Applications using Reusable Specifications of Patterns

Outline of Contributions -Ctd-

Conducted Research #3: Classification and Selection of Components and Web Services for Dev-by-Reuse



1. Classification of Components and Web Services in Repositories
2. Selection of Primitive and Composable Web Services from Classified Repositories
3. Models and Methods for Assessing WS Quality of Experience for enriching the classification

Outline

Conducted Research #1: Architecture Description in Reusable Models and Programs

Research #1.1: Specification of Architecture Constraints and their Reuse-by-Composition

What are Architecture Constraints?

Predicates that accompany an architecture model and which enable to specify in a checkable way the structural conditions induced by design choices

Example

Instantiation of MVC (*Model*, *View & Controller*) pattern in an architecture model implies that the entities (classes, for instance) part of the (business) Model should not have dependencies with entities which belong to the View, ...

Example of an Architecture Constraint in OCL[4]

Layered Architecture Style

```
context ACS:Component inv LayeredArchitecture:
let layers : Set(Component) = ACS.part.type->asSet()
->select(t:Type | t.oclIsKindOf(Component))
->collect(t:Type | t.oclAsType(Component)),
connectors:Set(Connector)=ACS.ownedConnector
->reject(c:Connector | c.kind=ConnectorKind::delegation)
in
-- Each layer should be connected
layers->forAll(c:Component|connectors.end.role->includes(c.ownedPort))
and
-- Each layer should be connected to at most two other layers
layers->forAll(com:Component | connectors->select(con:Connector |
  con.end.role->includes(com.ownedPort))->size() <= 2)
and
-- The number of layers that are connected to only one other
-- layer is equal to two (top and bottom layers)
layers->select(c:Component | connectors->one(con:Connector |
  con.end.role->includes(c.ownedPort)))->size() = 2
```

[4] https://www.omg.org/spec/OCL/

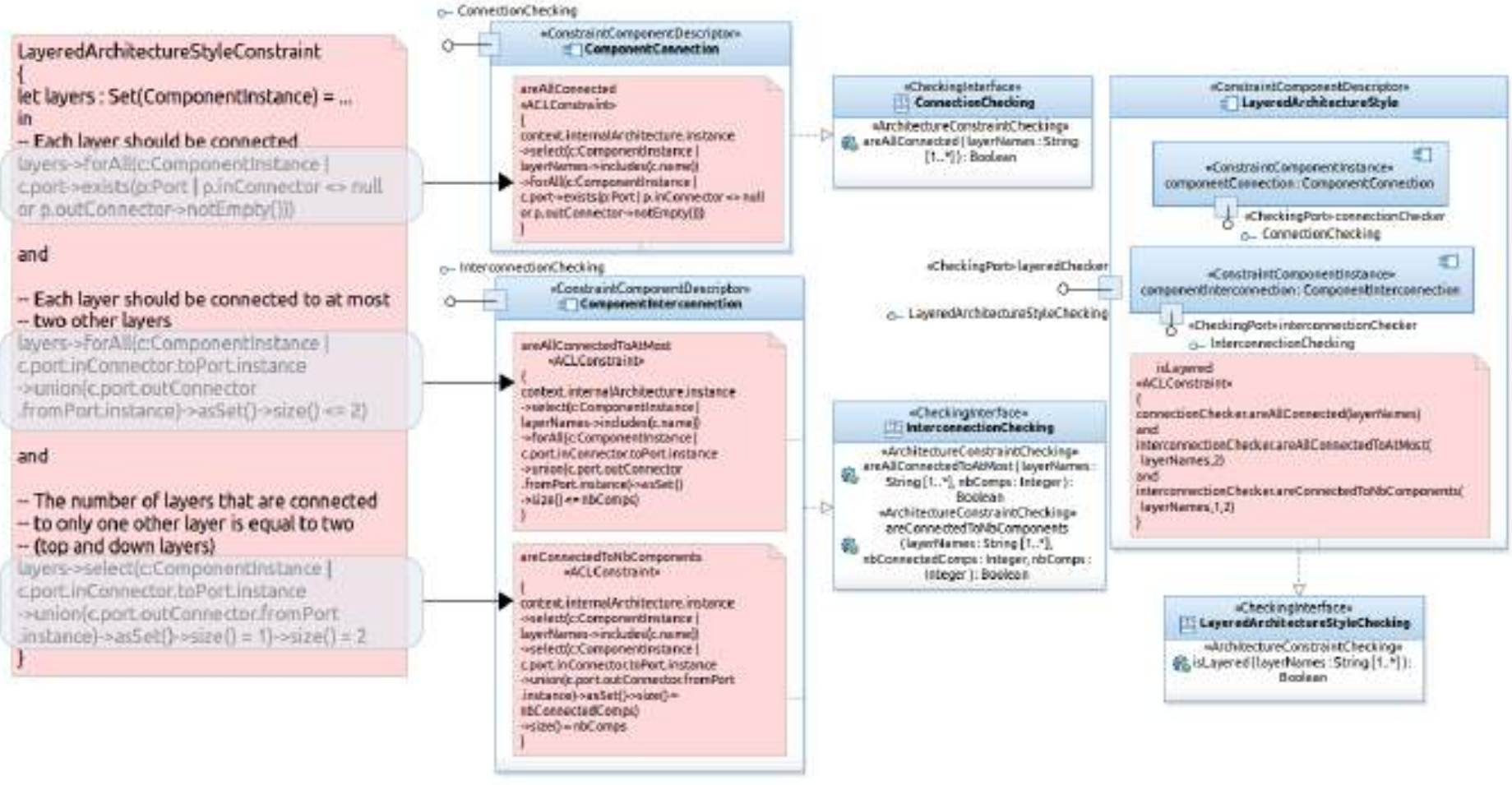
Limits of Existing Architecture Constraint Specification Languages

They provide limited support for the following features:

- Reusability: constraints need to be specified in a way that makes them reusable specifications
- Customizability: to be reused (applicable in different contexts), constraints should be parameterized by architecture elements of the arch. model on which they are checked
  → not known *a priori*
- Composability: complex constraints can be beneficially built as combinations of other (existing) simpler ones

Componentization of Architecture Constraints

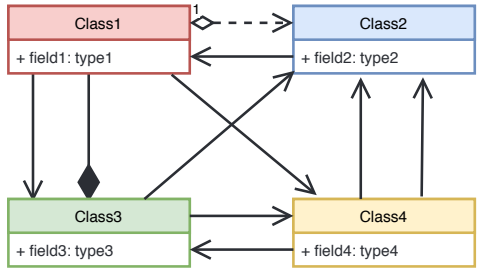
CLACS Architecture Constraint-Component Model[5,6]



[5] C. Tibermacine, S. Sadou, C. Dony and L. Fabresse. In Proc. of CBSE'11.

[6] C. Tibermacine, S. Sadou, M. T. Ton That, and C. Dony. In Journal of Future Generation Computer Systems, 61:37–53, 2016.

Research #1.2: Programming Reusable Components and Architecture Constraints

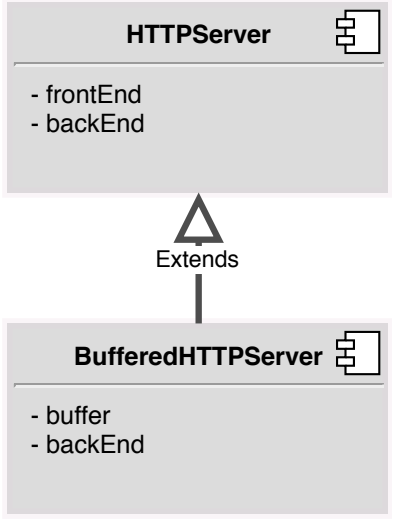


- **Problems:** Coupling and Instantiation Anticipation in existing OO Programs[7]
- **Existing Work, SCL Language[7]:**
  - A pure component-based programming (CBP) language (no mix between objects and components)
  - Component Descriptors vs Component Instances (like classes vs objects)
  - Communication between components via ports only
  - “Connection” instead of “anticipated instantiation+assignment”

[7] PhD Thesis of Luc Fabresse. University of Montpellier, 2008.

COMPO, a CBP Language that Pushes Reuse farther

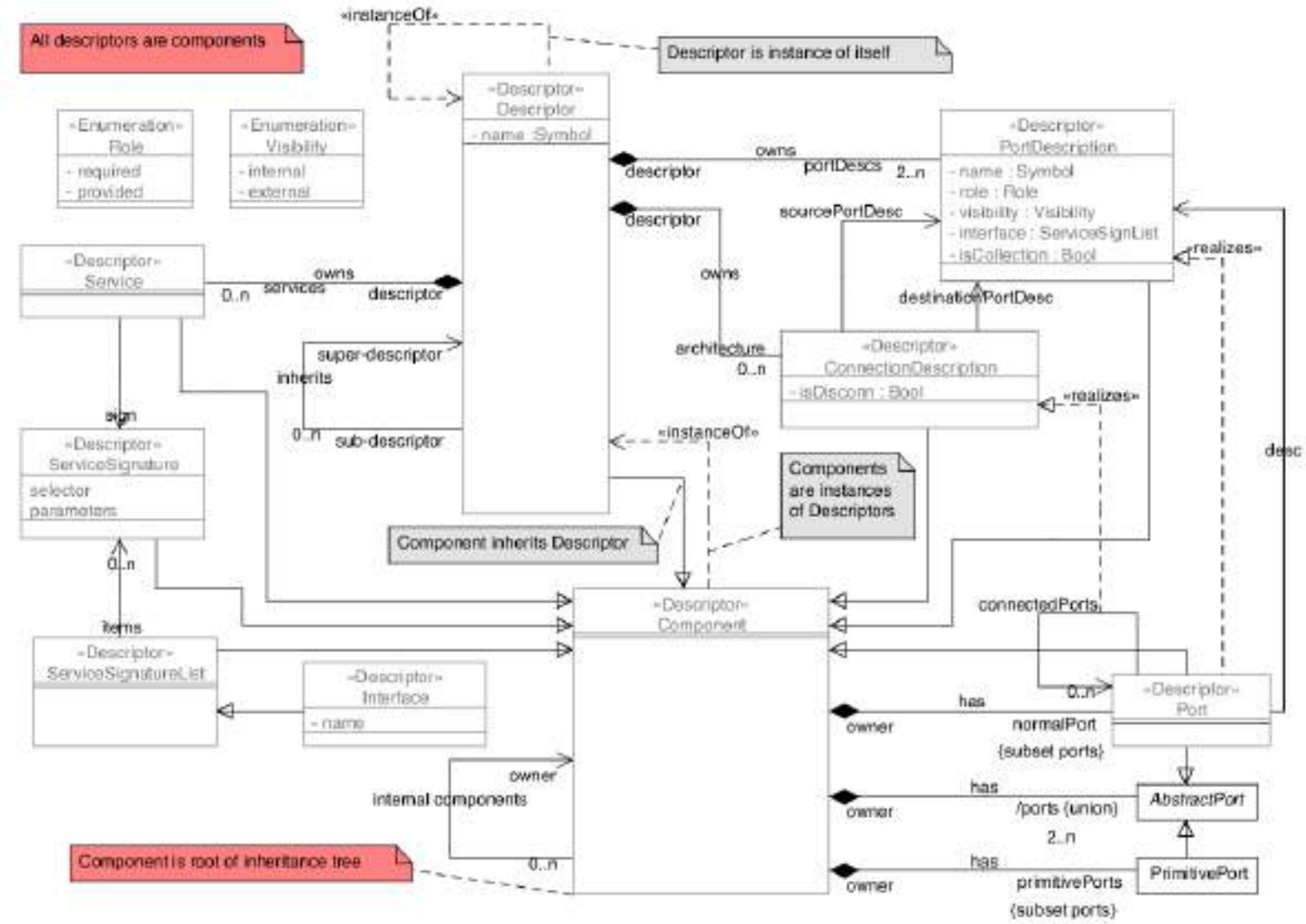
Two New Extra Features (PhD Thesis of Petr Spacek: 2010-2013):



- An inheritance system[8] (new relation between component descriptors):
  an alternative reuse mechanism (+ composition)
- A reflective design and implementation of the language[9]:
  specification of architecture constraints as meta-programs
  → a special kind of component descriptors

[8] P. Spacek, C. Dony, C. Tibermacine and L. Fabresse. In Proc. of GPCE'12.
[9] P. Spacek, C. Dony and C. Tibermacine. In Proc. of CBSE'14.

COMPO's Metamodel showing its Reflective Design

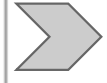


Research #1.3: Translation of Architecture Constraints into Reusable Meta-Programs

- **Problem:** Constraints are specified and checked at design time only (not in the implementation or at runtime)
- Solutions in the PhD Thesis of Sahar Kallel (2014-2018)

1. From OCL OO arch. constraints to Java meta-programs[10]

```
context Package inv:
let Model:
 Set(Class)= self.ownedType
 ->oclAsType(Class)->select(c:Class
 |c.getAppliedStereotypes()
 ->exists(s:Stereotype | s.name='model'))
-- ... the same for view and controller
in
-- No dependencies between Model and View
-- and between Model and Controller
Model->forAll(c:Class|c.supplierDependency
 .client->forAll(c1:Class|View->excludes(c1)
  and Controller->excludes(c1)))
```



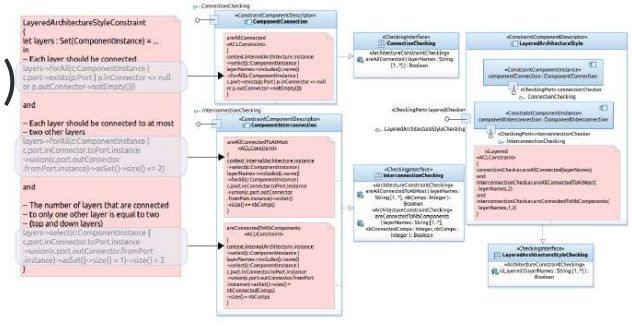
```
public class MVCConstraint {
 //...
 public boolean invariant(Class<?>[] self) {
  ArrayList<Class<?>> klass = new ArrayList<Class<?>>();
  for(Class c : self) {
   boolean bool = c.isAnnotationPresent(Model.class);
   if(bool) {
    klass.add(c);
   }
  }
  Class<?>[] klass1 = new Class<?>[klass.size()];
  //...
 }
}
```

[10] S. Kallel, C. Tibermacine, S. Kallel, C. Dony and A. Hadj Kacem. In Journal of Information and Software Technology (IST). 101:16–31, May 2018. Elsevier.

Research #1.3: Translation of Architecture Constraints into Reusable Meta-Programs -Ctd-

2. From OCL arch. constraints to components and services[11, 12]:
   - CLACS Component Descriptors (Design)
   - COMPO Component Descriptors (Implem.)
   - OSGi Bundles and services (Implem.)

     ⟶ an alternative reuse mechanism through service registries



3. Multi-paradigm architecture constraint specification (OCL+graphs) ⟶ configuration with a feature model ⟶ generation of paradigm-dependent constraints

---

[11] S. Kallel, B. Tramoni, C. Tibermacine, C. Dony & A. H. Kacem. In Proc. of ECSA'15.

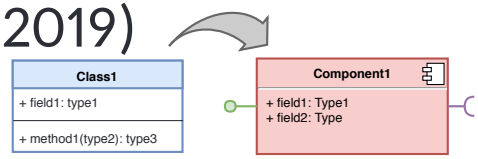
[12] S. Kallel, B. Tramoni, C. Tibermacine, C. Dony and A. H. Kacem. In Journal of Systems and Software, 127:91-108, Feb. 2017. Elsevier.

Conducted Research #2: Re-engineering and Evolution of Legacy Applications for improving Reusability

Research #2.1: Migrating Object-oriented Applications towards Component-based ones

- **Problem:** Existing large OO apps are difficult to reuse
- **Solution:** Transform them to CB apps and investigate two different viewpoints on components (two reuse granularity levels):

1. A Component Descriptor = a cluster of classes (PhD Thesis of Zak Alshara: 2013-2016)
2. A Component Descriptor = a decoupled class (PhD Thesis of Soumia Zellagui: 2015-2019)



Migration through Architecture Recovery

- Refactoring OO code using the result of component-based architecture recovery[13]:
  - Inter-cluster inheritance relations ⟶ composition
  - Inter-cluster anticipated instantiations ⟶ factory methods
- Materializing the recovered components[14]:
  - Generating classes representing each component (cluster of classes)
- Implementation: Java ⟶ OSGi

---

[13] Z. Alshara, A.-D. Seriai, C. Tibermacine, H. L. Bouziane, C. Dony and A. Shatnawi. In Proc. of GPCE'15, Pittsburgh, USA, 2015.

[14] Z. Alshara, A.-D. Seriai, C. Tibermacine, H. L. Bouziane, C. Dony and A. Shatnawi. In Proc. of ECSA'16, Copenhagen, Den., 2016.

Pushing Reuse further by Refactoring every Class[15]

- Each class is refactored and transformed into a component descriptor (*à la* COMPO):
  - make explicit their required interfaces
  - publish their provided interfaces
  - remove anticipated instantiations and replace them by annotations processed automatically by *dependency injection*
- Implementation: Java ⟶ Spring

```java
@Component
public class Customer {
    private Person person;
    @Autowired
    public Customer (Person person) {
        this.person=person;
    }
}
```

[15] S. Zellagui, C. Tibermacine, H.-L. Bouziane, A.-D. Seriai and C. Dony. In Proc. of SEKE'17, Pittsburgh, PA, USA, 2017

Research #2.2: Migrating Web Applications towards Web Service-oriented Ones

- **Problem:** Existing Web applications closed for development-by-reuse by third-parties
- **Solution:** A method for generating Web services from Web apps code
- Improving reuse of Web application functionality[16]: PhD Thesis of Mohamed Lamine Kerdoudi (2009-2016)
- New concepts of "Web interface as a service"[17] and "Web navigation as a service orchestration"

[16] M. L. Kerdoudi, C. Tibermacine and S. Sadou. In Journal of Service-Oriented Computing and Applications, 2:1–27 2016. Springer

[17] C. Tibermacine and M. L. Kerdoudi. In Proc. of ICWS'12, Honolulu, HI, USA, 2012

Research #2.3: Evolution of WS-oriented Apps by the Integration of Reusable Specs of Patterns

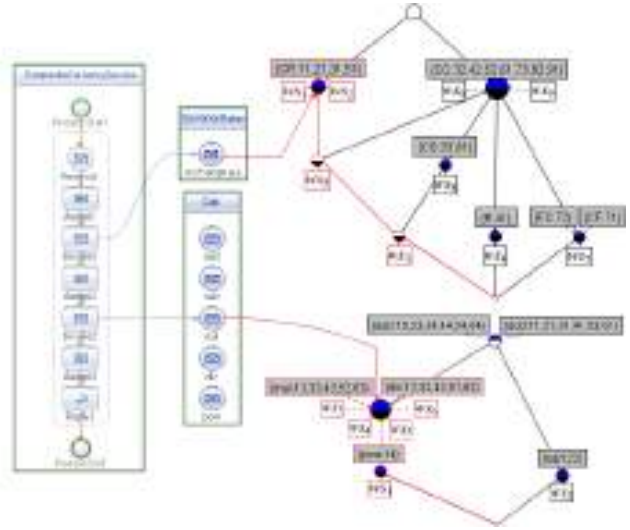
- **Problem:** lack of support to assist developers of WS-oriented apps in handling new quality requirements: PhD Thesis of Tarek Zernadji (2010-2016)
- **Solution:**

1. A Language for the specification of SOA patterns as reusable:
   - parameterized scripts for their instantiation and cancellation
   - checkable OCL architecture constraints
2. A Catalog of SOA patterns (from Thomas Erl's Book) defined with this language, for feeding repositories
3. A Process for selection of patterns from this catalog to satisfy new quality requirements[18]

[18] T. Zernadji, C. Tibermacine, F. Cherif and A. Zouioueche. In Journal of Systems and Software, 122:463–483, 2016. Elsevier.

Conducted Research #3: Classification and Selection of Components and Web Services for the Development by Reuse

Research #3.1: Classification of Repositories of Components and Web Services



- **Problem:** it is difficult to use existing large flat repositories/directories with hundreds to thousands of items
- **Solution:** Using Formal Concept Analysis to build hierarchies of components[19] & Web services[20]

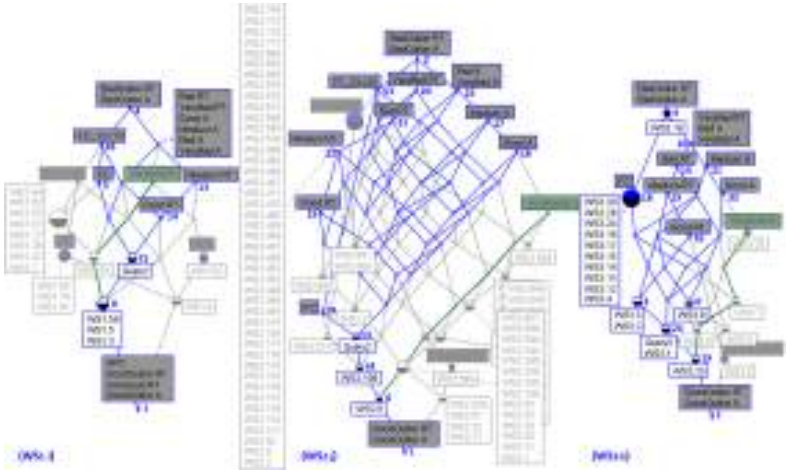
  (PhD Thesis of Zeina Azmeh: 2007-2011)
- **Final Goal:** easily find a substitute to a failing item

---

[19] N. A. Aboud, G. Arévalo, J.-R. Falleri, M. Huchard, C. Tibermacine, C. Urtado and S. Vauttier. In Proc. of WICSA/ECSA'09, Cambridge, UK, 2009.

[20] Z. Azmeh, M. Huchard, C. Tibermacine, C. Urtado, and S. Vauttier. In Proc. of ECOWS'08, Dublin, Ireland, 2008.

Research #3.2: Selection of Primitive and Composable Web Services



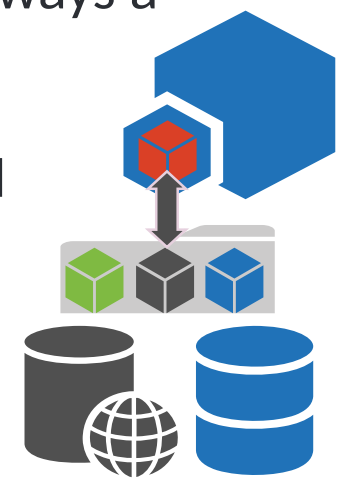
- A method for selecting Web services starting from user requirements: i) functionality (an abstract process of tasks described by keywords) and ii) QoS[21]
- Use Relational Concept Analysis[22] (a variant of FCA) to encode composition relations between WSs
- Goal: Identify a set of Web services that are composed 2-by-2 and which satisfy user requirements

[21] Z. Azmeh, M. Driss, F. Hamoui, M. Huchard, N. Moha and C. Tibermacine. In Proc. of ICWS'11, Washington, D.C., USA

[22] Relational concept discovery in structured datasets. M. Huchard *et. al.* Annals of Mathematics and Artificial Intelligence, 49(1-4):39-–76, 2007.

Selection of Web Service Compositions

- **Problem:** For a failing Web service, we do not find always a simple substitute
- **Solution:** Exploit FCA classifications to identify N-to-1 substitutes[23]: PhD Thesis of Okba Tibermacine (2010-2015)
- A process for:
  1. extracting keywords from WS descriptions in repos
  2. building matrices of similarity scores between WS descriptions (input vs output parameters)
  3. transforming these matrices and building FCA classifications
  4. interpreting them to identify a sequence of N WSs to replace the failing service



[23] O. Tibermacine, C. Tibermacine and F. Cherif. In Journal of Systems and Software 104:1–16, 2015. Elsevier.

Research #3.3: Models and Methods for Assessing Web Service Quality of Experience

- **Problem:** WS QoS scores (used in classification) are most of the time outdated or inflated by providers: do not reflect reality
- **Solution:** A Method for measuring the quality of experience (reputation)
- Aggregates scores given by users (considers presence of malicious users & temporal sensitivity of scores)

$$\Phi(S_k) = \left(\sum_{i=1}^{n} \delta^{+}_{(i,k)} \times \lambda^{d_i} \times H_i\right) + \left(\sum_{j=1}^{m} \delta^{-}_{(j,k)} \times \lambda^{d_j} \times H_j\right)$$

$$H_i = \frac{\sum_{s=1}^{t}\left(\frac{\Psi(s)}{\Psi^{+}(s)+\Psi^{-}(s)}\right)}{t}$$

$$\Omega(S_k) = \begin{cases} \frac{\left(\sum_{i=1}^{n} \delta^{+}_{(i,k)} \times \lambda^{d_i} \times H_i\right) - \left(\sum_{j=1}^{m} \delta^{-}_{(j,k)} \times \lambda^{d_j} \times H_j\right)}{\Phi(S_k)} & if \Phi(S_k) \neq 0 \\ -1 & otherwise \end{cases}$$

$$R(S_k) = \begin{cases} \frac{\left(\sum_{i=1}^{n} \delta^{+}_{(i,k)} \times \lambda^{d_i} \times H_i\right)}{10 \times n} & if\ \Omega(S_k) = 1 \\ \frac{\left(\sum_{j=1}^{m} \delta^{-}_{(j,k)} \times \lambda^{d_j} \times H_j\right)}{10 \times m} & if\ \Omega(S_k) = -1 \\ \frac{\Omega(S_k)+1}{2} & Otherwise \end{cases}$$

- Solution to the "Cold Start" problem: bootstrapping the system using a regression-based estimation method[24]
- **Final Goal:** Feed repos with new information about WSs

[24] O. Tibermacine, C. Tibermacine and F. Cherif. In Proc. of ICWS'15, NY, USA, 2015

Thank You

Questions | Remarks