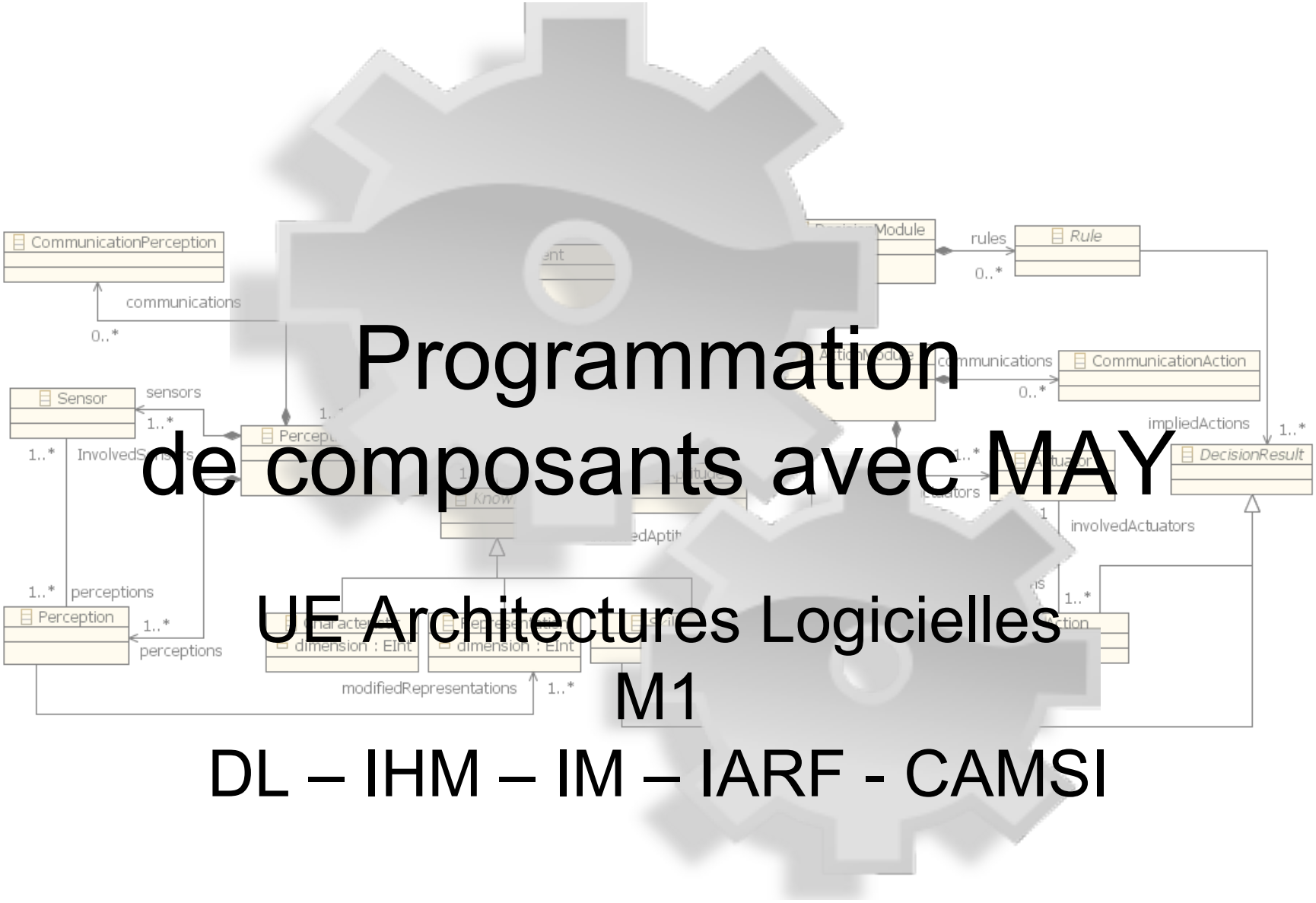


Programmation de composants avec MAY

UE Architectures Logicielles

M1

DL – IHM – IM – IARF - CAMSI

F. Migeon, janvier 2014

Plan

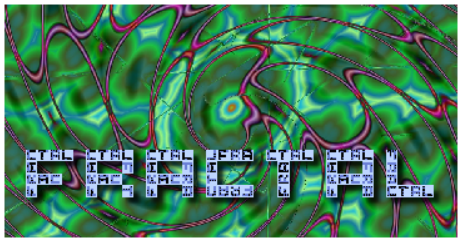
- Concevoir et Programmer des composants
- La plateforme MAY
  - Le langage SpeADL
  - La démarche exigences SpEArAF
  - Un IDE MAY
- Une DEMO !

Concevoir et Programmer des composants

- Concevoir et programmer ?
  Ou
  Concevoir ou programmer ?
- Théorie et pratique
  - Un machin avec des pattes en entrée et des pattes en sortie, ça ne se programme pas bien avec des objets.

La programmation en pratique

- Avec Fractal ou MAY
  - Un modèle propre
  - Un ADL dédié
  - Une implantation en Java
- Avec des Beans, des Composants Corba, etc
  - Une implantation en Java, Corba ou autre…
  - Pas d'ADL
  - Un modèle ? Superflu



MAY : Panorama

- Faciliter le passage de la **conception** d'un SMA, en termes de types d'agents et d'interactions, à son **implémentation**.
- Démarche d'Architecture Logicielle
- Approche « requirement-aware » ;-)
- Basé sur le concept de composant logiciel.
- Equipe SMAC – Victor Noël

http://www.irit.fr/~Victor.Noel/Projects/MAY

MAY : faits d'armes

- 20 ans de recherche sur le dév de SMA Adaptatifs
  - Réflexivité
  - Agent mobile, flexible, adaptatif
  - Architecture à composants
  - Ingénierie Dirigée par les Modèles
  - Architecture des SMA
- Utilisé dans le dev de SMA pour la recherche et l'industrie
  - Upetec avec Airbus, DCNS, Grand Toulouse, Ubisoft
  - IRIT/SMAC avec Artal, Renault, Ragt, Rocwell Collins

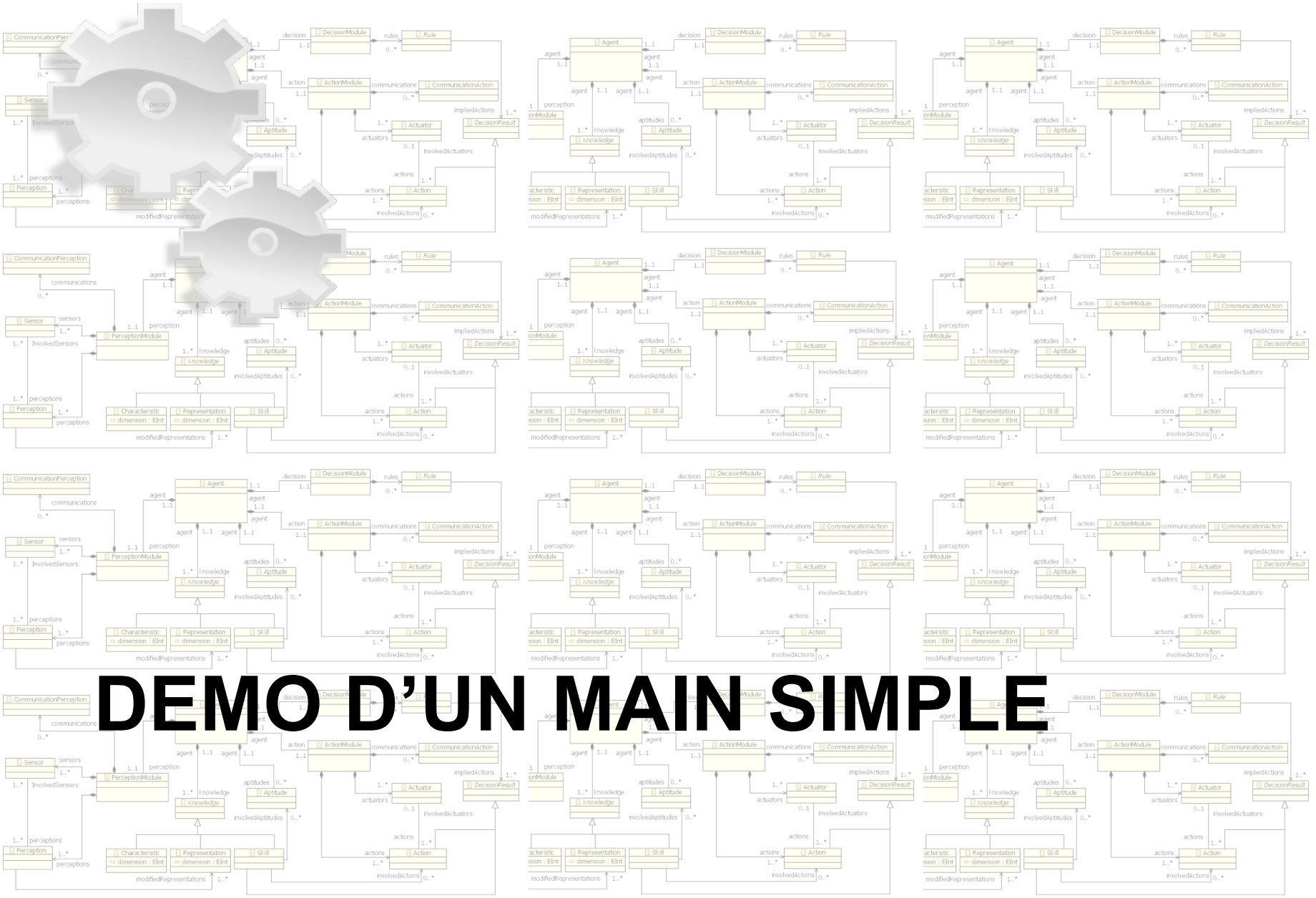
Principes de MAY

- Modèle de composants SpeAD (Species-based Architectural Design)
- Langage de description d'architecture SpeADL (Species-based Architectural Description Langage)
- Abstractions plus riches que les abstractions classiques des modèles de composants.
  - Ecosystème
  - Espèce

A l'exécution

1. On crée une instance
   1. À partir d'une définition de type
   2. Et d'une implantation de composant

   Le démarrage du composant est automatique
2. On invoque son service
   1. À travers un port
   2. Dont on invoque une méthode



DEMO D'UN MAIN SIMPLE

Instanciation, Démarrage et Invocation

```
package impl;
import OCTest.MyFirstComponentType;

public class Run {
  public static void main(String[] args) {
    MyFirstComponentType.Component myComp =
      (new MyFirstCompImpl()).newComponent();
    myComp.sonPortDeService().saMethode();
  }
}
```

Nom du type de composant

Référence de composant

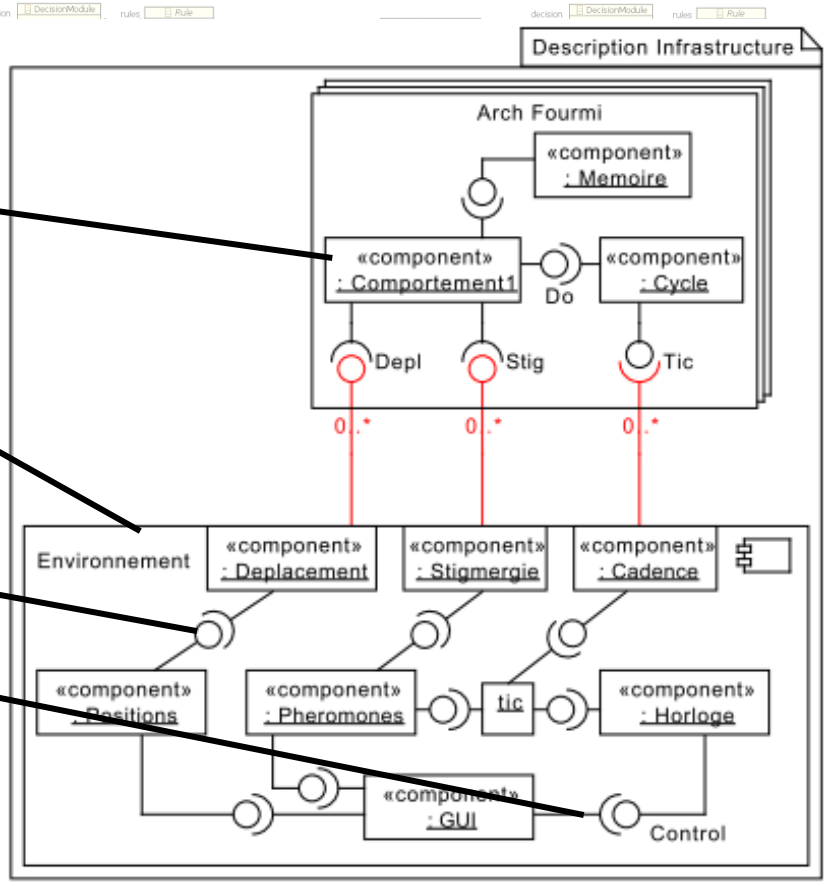
Création de la fabrique

Invocation de la fabrique pour créer un composant

Invocation du service

La vision composant de SpeAD

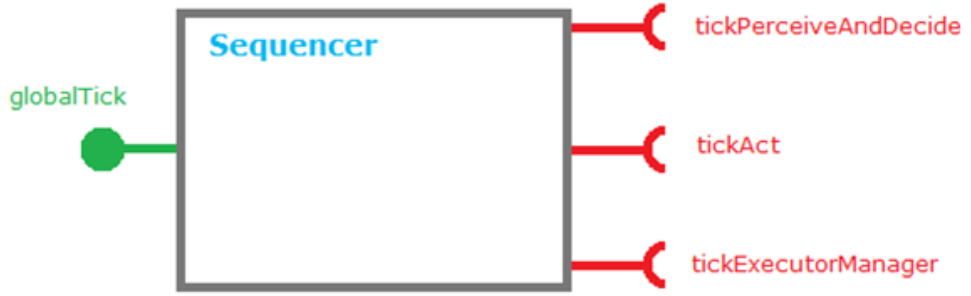
- Composant
  - Simple
  - Composite
- Port ≈ Interface
  - Fourni
  - Requis



(photo non contractuelle)

Le langage SpeADL (1)

- Composant simple
  - un nom
  - des ports :
    - un « rôle »
    - un nom
    - une signature (interface java)




```
component Sequencer {
        provides globalTick : Tick
        requires tickPerceiveAndDecide : Tick
        requires tickAct : Tick
        requires tickExecutorManager : Tick
}
```

A condition que

public interface Tick {…}

soit définie quelque part…

DEMO DE FICHIER SPEADL

Définition d'un Type de Composant Simple

Importation (classique) de définition

Packaging

Nom du type de composant

Rôle du port

Nom du port

Type du port

```
import interfaces.UneInterfaceDeTypage

namespace OCTest {

component MyFirstComponentType {

provides sonPortDeService : UneInterfaceDeTypage

}

}
```

```
package interfaces;
public interface UneInterfaceDeTypage {
    public void saMethode();
}
```

L'implantation des composants

- Génération automatique
  - des classes de composants

- Il reste à :
  - Implanter les composants
  - Soit pour chaque port fourni
    - Donner une implantation de l'interface du port.
    - Inutile pour un composite avec des ports « exportés » ou « importés ».

DEMO DE L'IMPLANTATION

Implantation d'un Composant Simple

Nom de (de la classe pour) l'implantation

Nom du type du composant

Type du port

Nom du port

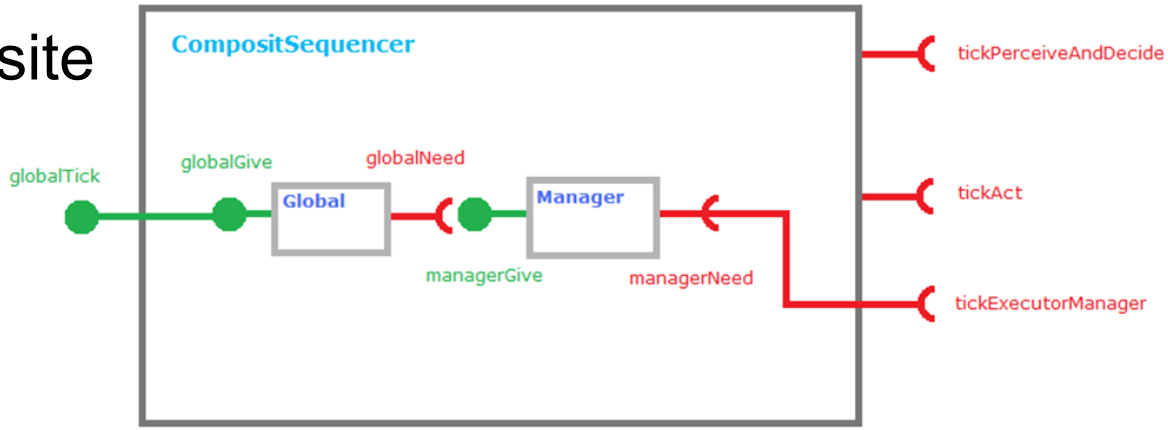
Implantation anonyme de l'interface

Implantation du service

```
package impl;
import interfaces.UneInterfaceDeTypage;
import OCTest.MyFirstComponentType;

public class MyFirstCompImpl extends MyFirstComponentType{

    @Override
    protected UneInterfaceDeTypage make_sonPortDeService() {

        return new UneInterfaceDeTypage(){

            public void saMethode() {
                System.out.println("Ca marche");
            }
        };
    }}
```

Le langage SpeADL (2)

- Composant composite
  - un nom
  - des ports



component CompositSequencer {
provides globalTick : Tick
requires tickPerceiveAndDecide : Tick
requires tickAct : Tick
requires tickExecutorManager : Tick

A condition toujours que

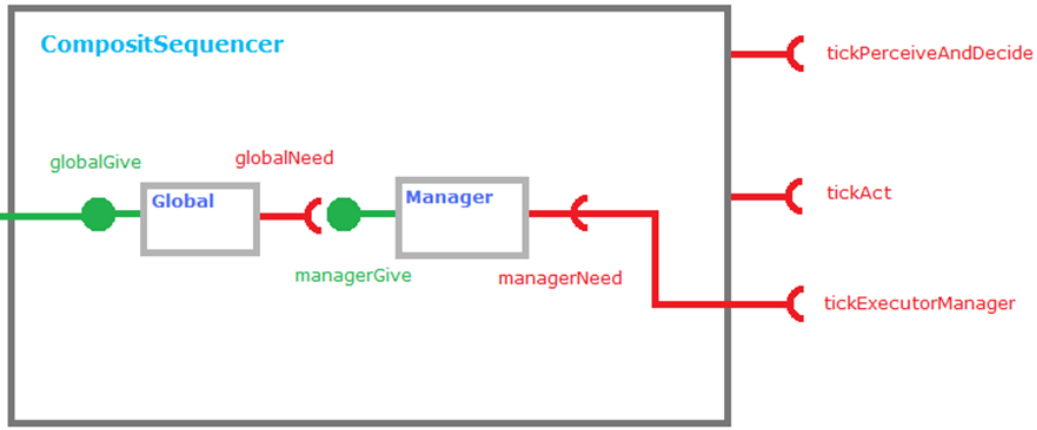
public interface Tick {…}

soit définie quelque part…

}

Le langage SpeADL (2)

- Composant composite
  - un nom
  - des ports
  - un contenu :
    - des sous-composants




```
component CompositSequencer {
    provides globalTick : Tick
    ...
    part glob : Global {
    }

    part manage : Manager {

    }
}
```

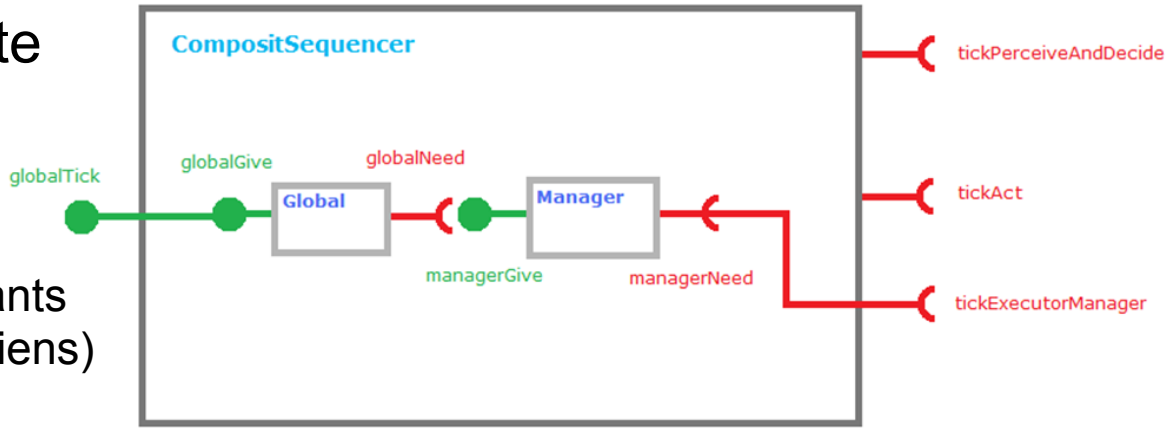
A condition toujours que

public interface Tick {…}

soit définie quelque part…

Le langage SpeADL (2)

- Composant composite
  - un nom
  - des ports
  - un contenu :
    - des sous-composants
    - des connecteurs (liens)




```
component CompositSequencer {
    provides globalTick : Tick  = glob.globalGive
    ...
    part glob : Global {
        bind globalNeed to manage.managerGive
    }
    part manage : Manager {
        bind managerNeed toThis tickExecutorManager
    }
}
```

A condition toujours que public interface Tick {…} soit définie quelque part…

Le langage SpeADL (3)

- Configuration cohérente
  - Tous les ports requis des sous-composants connectés
    - À un port fourni par un autre sous-composant
    - À un port requis du composant composite
  - Tous les ports fournis des sous-composants connectés
    - À un port requis d'un autre composant
    - À un port fourni du composant composite (exposition)

Le langage SpeADL (4)

- Expression des connexions
  - entre deux composants de même niveau
    bind <required> to <provided>
  - entre composite et sous-composant
    bind <required> toThis <required>
  - *entre ports fournis du composant et du composite*
    <provided> = <provided>

DEMO D'UN « GROS » COMPOSANT

Définition d'un Type de Composant Composite

Définition d'un client (simple)

Définition d'un serveur (simple)

Nom du type du composite

Définition du port

Inclusions des parts

```
namespace CS {
import interfaces.TypeDeServiceSuperChouette
import interfaces.TypeDeServicePasTerrible

component ClientComponentType {
  provides unServiceSuperChouette :TypeDeServiceSuperChouette
  requires unServiceBasiqueBofBof : TypeDeServicePasTerrible
}
component ServerComponentType {
  provides unServicePasSiMal : TypeDeServicePasTerrible
}
component BigComponentType {
  provides unServiceQuiDechire : TypeDeServiceSuperChouette =
                                 client.unServiceSuperChouette
  part  client: ClientComponentType {
    bind unServiceBasiqueBofBof to server.unServicePasSiMal
  }
  part server : ServerComponentType
}
}
```

Liaisons internes

Implantation d'un Composant Composite

Nom de (de la classe pour) l'implantation

Nom du type du composant

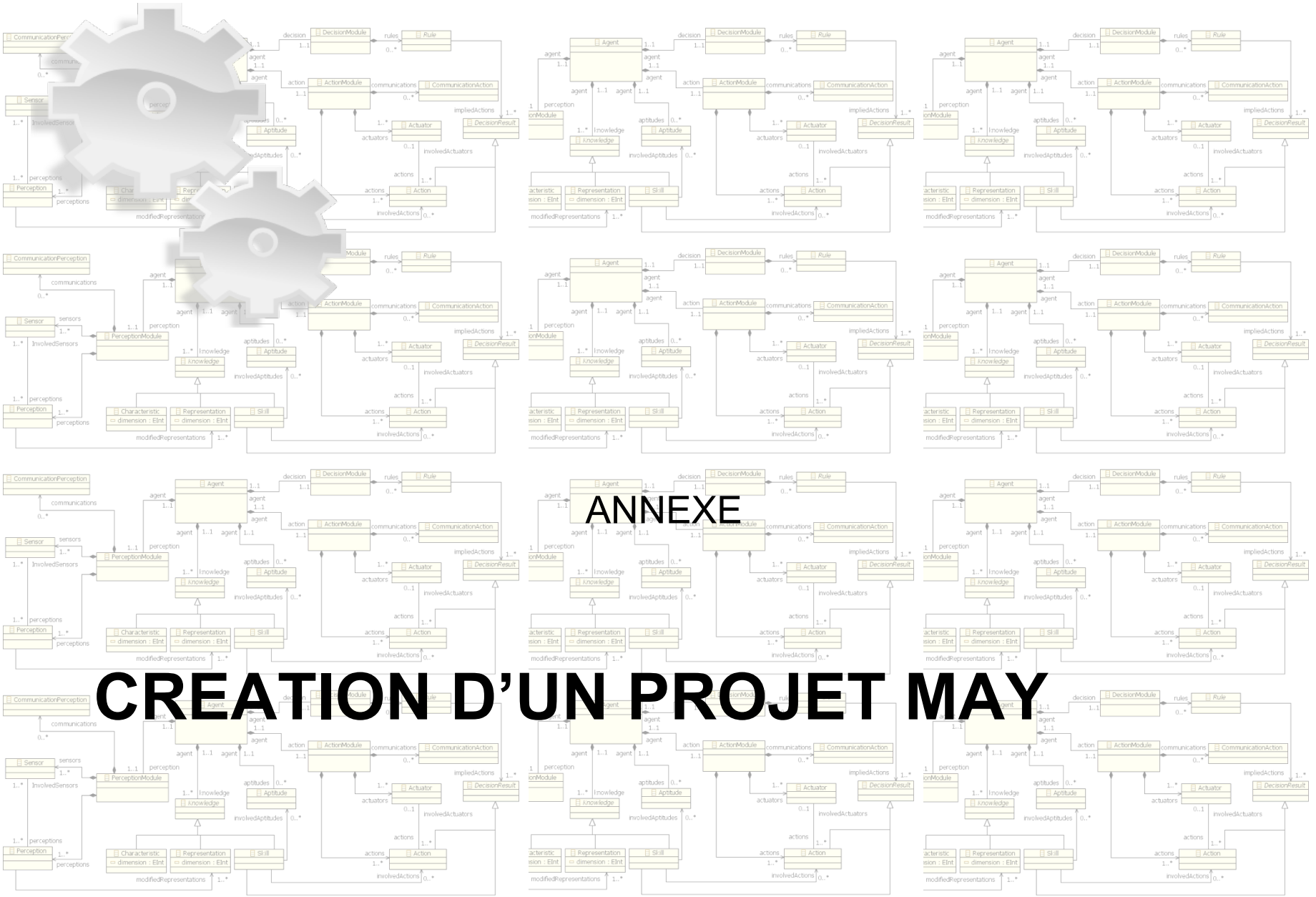
Déclaration des implantations des parts

Type des parts

Implantation des parts

```
package impl;
import CS.BigComponentType;
import CS.ClientComponentType;
import CS.ServerComponentType;

public class BigComponentImpl extends BigComponentType {

    @Override
    protected ClientComponentType make_client() {
        // TODO Auto-generated method stub
        return new ClientImpl();
    }

    @Override
    protected ServerComponentType make_server() {
        // TODO Auto-generated method stub
        return new ServerImpl();
    }

}
```

Nom des parts

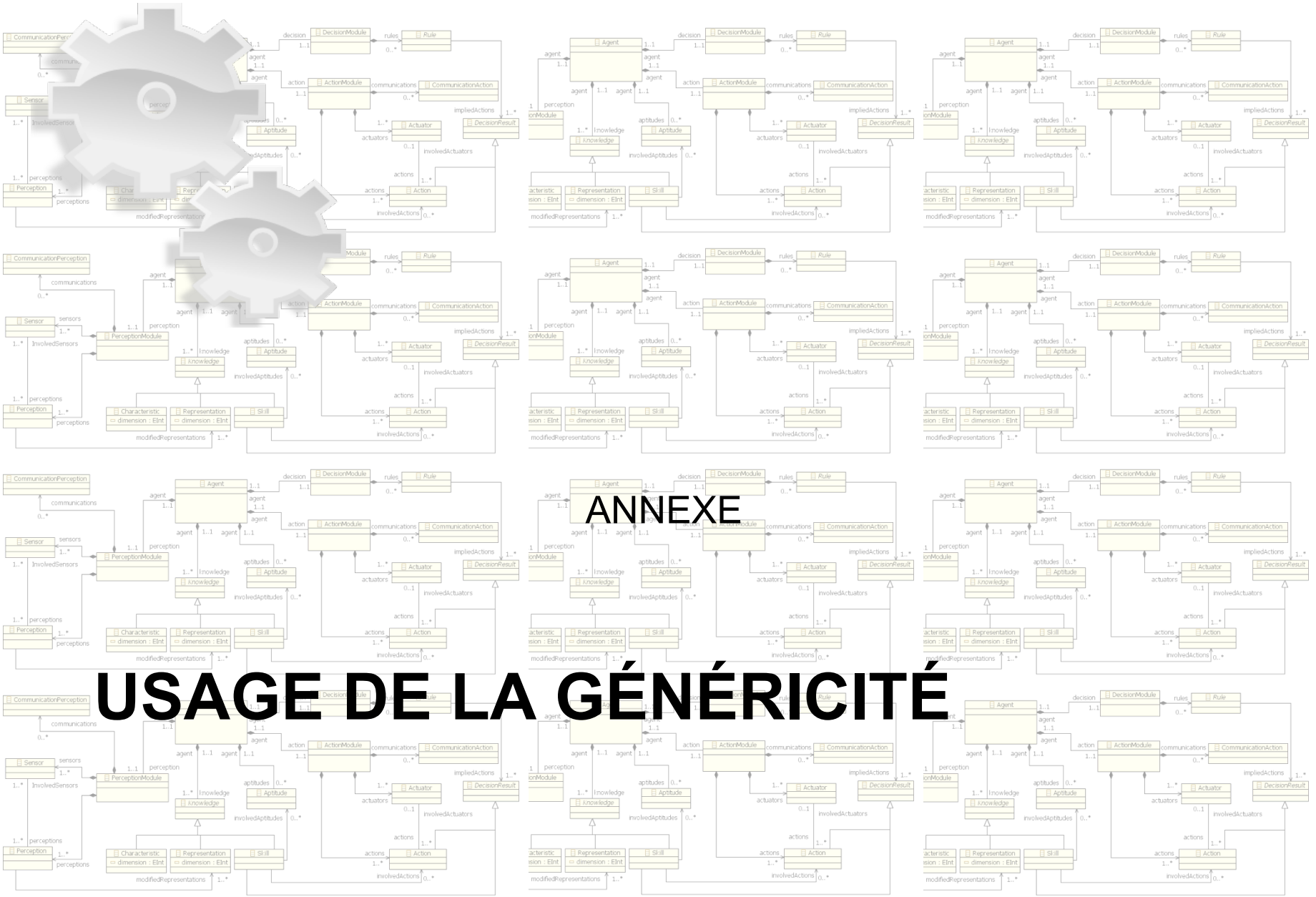


ANNEXE

CREATION D'UN PROJET MAY

Méthodologie

- Créer un projet java
- Créer 2 sous-packages src/impl et src/services pour organiser le code
  - src/impl regroupe toutes les classes java pour implanter les composants
  - src/services contient toutes les interfaces java typant les ports des composants
- Définir les types de composants dans un fichier nommé avec l'extension « .speadl »
- Définir les interfaces typant les ports définis précédemment
- Définir un (ou plusieurs) implantation(s) des types de composants
- Définir le main de votre appli (cf. Instanciation, Démarrage et Invocation)
- Enjoy ! ;-)



ANNEXE

USAGE DE LA GÉNÉRICITÉ

Le langage SpeADL (5)

- La généricité
  - pour paramétrer un type de composant par des types.
  - qui peuvent ensuite être utilisés dans
    - les descriptions
    - les implantations

Définition d'un Service Générique et d'un Type de Composant Générique

```
namespace CS {
import services.GenericService
import services.Start
import java.lang.Boolean

component BasicServer1Type {
        provides someBasicService : GenericService[Boolean]
}
component GenericClientType[Type] {
        provides go:services.Start
        requires aGenericService:GenericService[Type]
}
component GenericCompositeType{
        part cg : GenericClientType[Boolean] {
                bind aGenericService to s.someBasicService
        }
        part s : BasicServer1Type
}
}
```

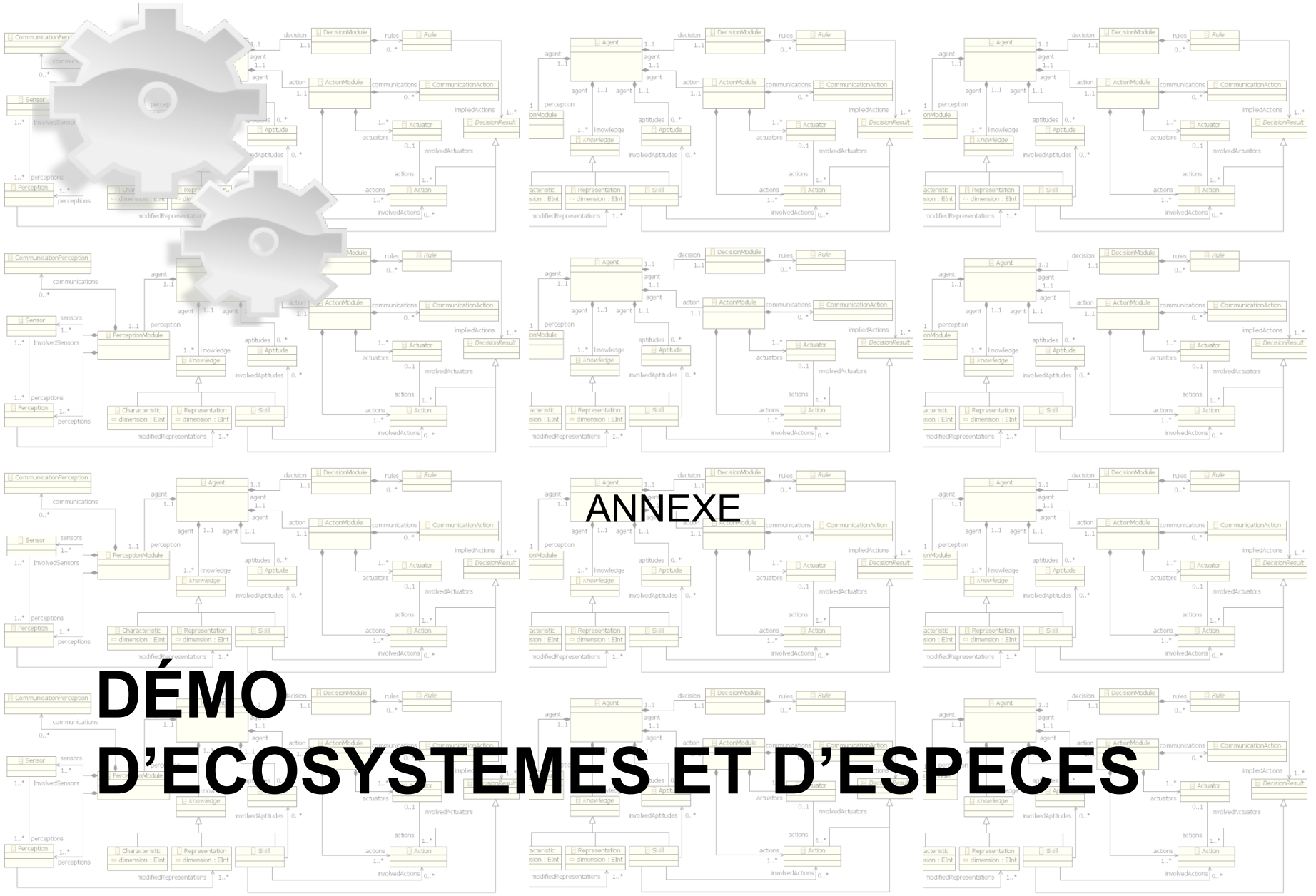
```
package services;
public interface GenericService<T> {
        public T genericMethod(T t);
}
```

Définition des implantations associées

```
public class GenericClientImpl extends GenericClientType<Boolean> {
@Override
protected Start make_go() {
        return new Start() {
                public boolean go(boolean b) {
                        return aGenericService().genericMethod(b);
                }
        };
}}
```

```
public class GenericCompositeImpl extends GenericCompositeType {
protected GenericClientType<Boolean> make_cg() {
        return new GenericClientImpl();
}

protected BasicServer1Type make_s() {
        return new BasicServerImpl();
}

public void start () {
        super.start();
        System.out.println("Démarrage du composite générique");
        System.out.println(cg().go().go(false));

}
}
```

Possibilité de redéfinir la méthode start() d'un composant



ANNEXE

DÉMO D'ECOSYSTEMES ET D'ESPECES

Le langage SpeADL (6)

- Un écosystème est un composant qui peut
  - fournir des services
  - requérir des services
  - être composé de parts

Définition d'un écosystème simple

```
namespace basicEcosystem {
ecosystem BasicServerEco {
        provides basicService : services.DelegateService
}
ecosystem BasicClientEco {
        provides go : services.Start
        requires bs : services.DelegateService
}
ecosystem CompositeEco {
        provides gogo : services.Start = c.go
        part c : BasicClientEco {
                bind bs to s.basicService
        }
        part s : BasicServerEco
}}
```

Le langage SpeADL (7)

- Une espèce se définit
  - relativement à un écosysteme
  - comme un composant

Définitions d'un espèce simple, de son implantation

```
namespace basicEcosystem {
        ecosystem NothingButSpecies {
                species SimpleServerSpecies {
                        provides service : services.DelegateService
                }
        }
}
```

```
public class SimpleServerSpeciesImpl extends SimpleServerSpecies {
protected DelegateService make_service() {
        return new DelegateService() {
                public void launch() {
                        System.out.println("...");
                }
        };
}}
```