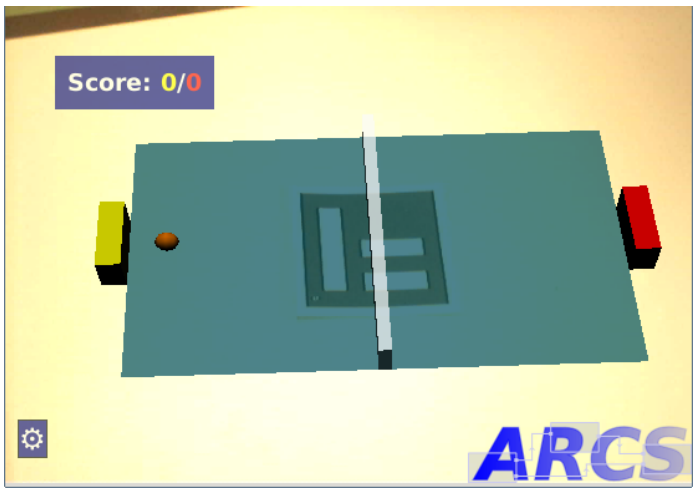
ARCS.js – Un Framework Web pour développer des applications de réalité augmentée

Jean-Yves Didier

didier@ufrst.univ-evry.fr



1 Concepts

2 Architecture du framework

3 Développer les composants

4 Décrire une application

ARCS.js en bref

ARCS : *Augmented Reality Component System*

- *Framework* de programmation orienté composants ;
- dédié aux applications de réalité augmentée.

Infrastructure technique

- Langage de programmation : Javascript ;
- Plate-formes ciblées / environnements de développement :
  - Côté client : navigateur récent (compatible HTML5),
  - Côté serveur : node.js.

Définitions

Framework

Cadre normatif et collection **d'outils** pour développer des applications.

Programmation orientée composants

- Composant : **élément de logiciel** (code compilé, scripts. . .) non auto-suffisant, sujet à **composition** ;
- Insiste sur la **réutilisation** du code.

Réalité Augmentée

Ensemble de techniques permettant de fusionner des informations **réelles** avec des entités **virtuelles** en **temps interactif**.

Modèle de composant

Paradigme signal/slot

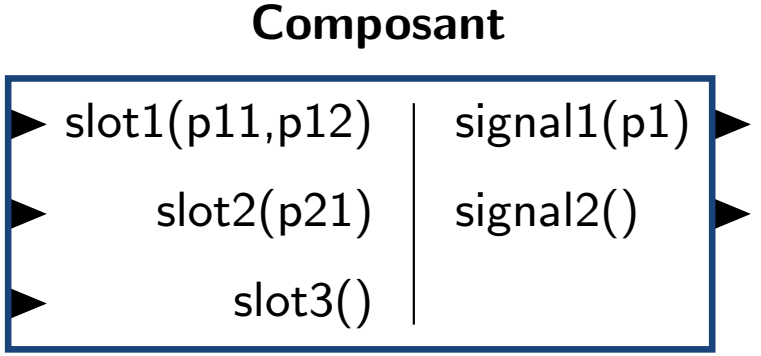
- Entrées : slots (méthode/fonction) ;
- Sorties : signaux (pas d'implémentation).

Modèle de communication

- Communication synchrone ;
- Passage de paramètres valués.

Initialisation/configuration

1. A l'instanciation ;
2. Par appel de slots.



Modèle de communication

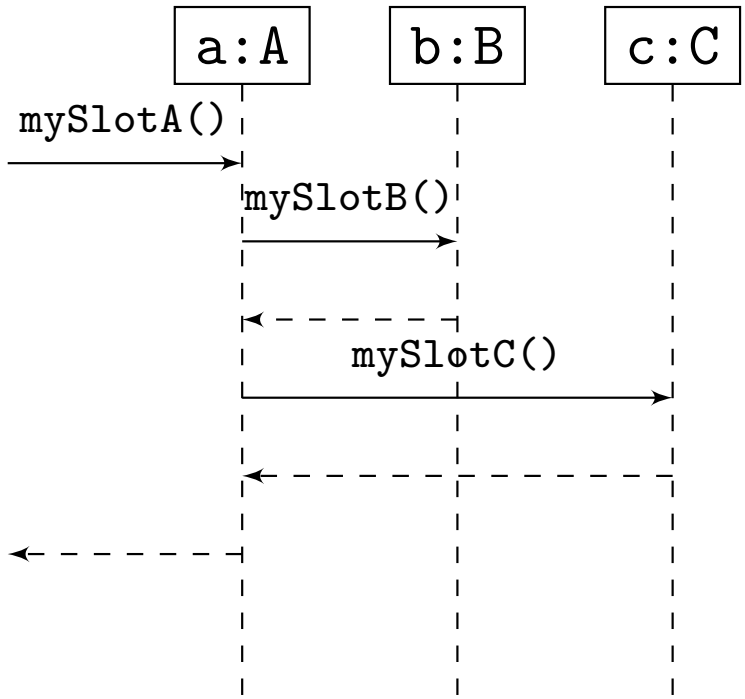
Exemple

Slot du composant a:A :

```
function mySlotA()
{
emit mySignal1();
emit mySignal2();
}
```

Liste de connections :

```
A.mySignal1() --> B.mySlotB()
A.mySignal2() --> C.mySlotC()
```



Modèle d'application

Cycle de vie d'une application

Plusieurs états (initialisation, fonctionnement en mode nominal, passage en mode dégradé...) !

État d'une application

- Configuration et agencement (connections) des composants ;
- Un état de l'application est appelé **feuille** (*sheet*).

Contrôle de l'application

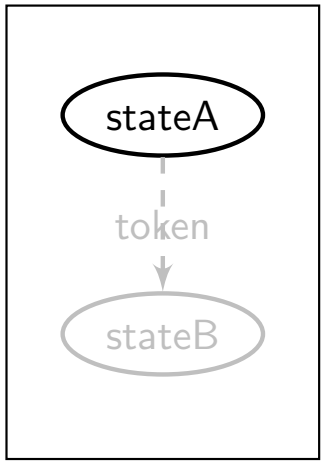
- Par un automate (machine à état fini) ;
- État de l'automate = configuration opérationnelle (feuille) ;
- Déclenchement d'une transition = passage d'une feuille à une autre.

Modèle de feuille

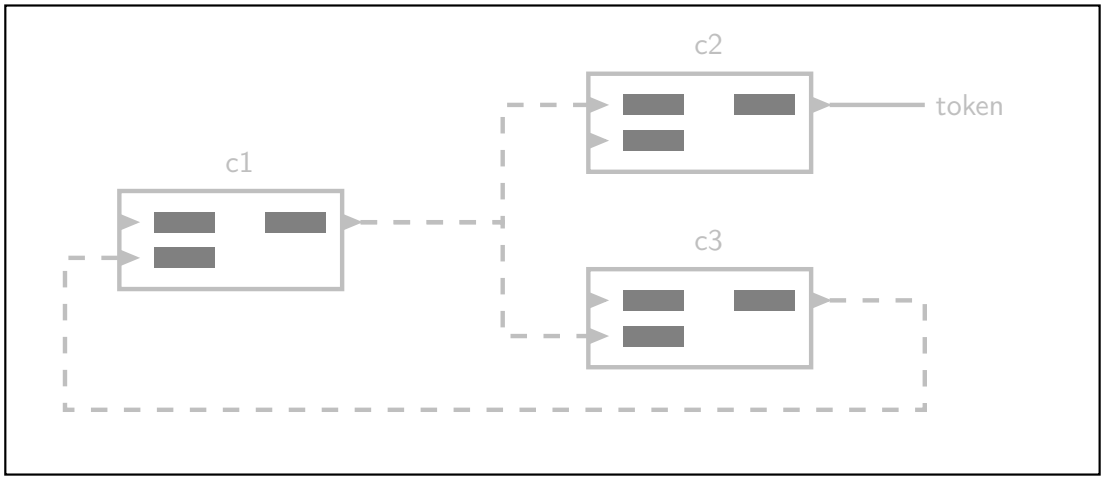
Une structure composite

- Une liste d'*invocations* de *pré-connexion* :
  - En lien avec l'initialisation des composants ;
- Une liste de *connexions* ;
  - Pour établir des chaînes de traitement.
- Une liste d'invocations de *post-connexion* ;
  - Pour démarrer les traitements associés à la chaîne.
- Une liste d'invocations de nettoyage (*cleanup*) :
  - Pour s'assurer de l'état des composants à la fin du traitement.

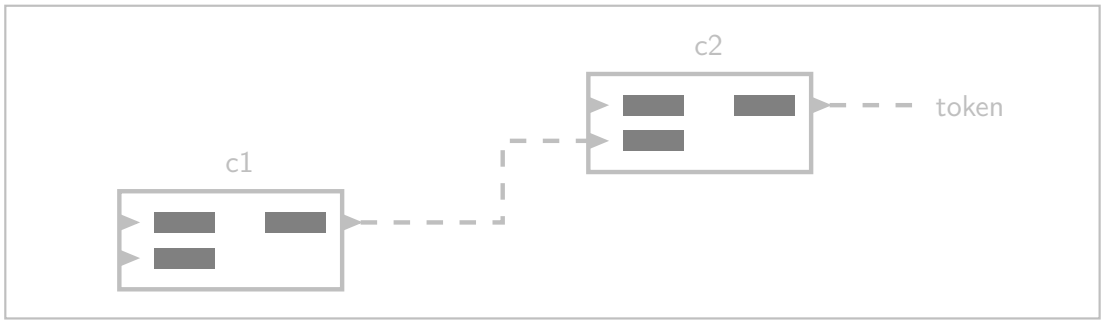
Basculer d'une configuration opérationnelle à une autre



Automate
(contrôleur)



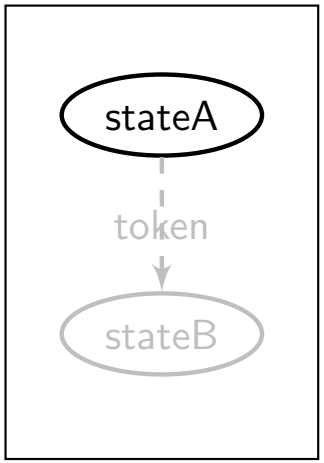
Feuille *stateA*



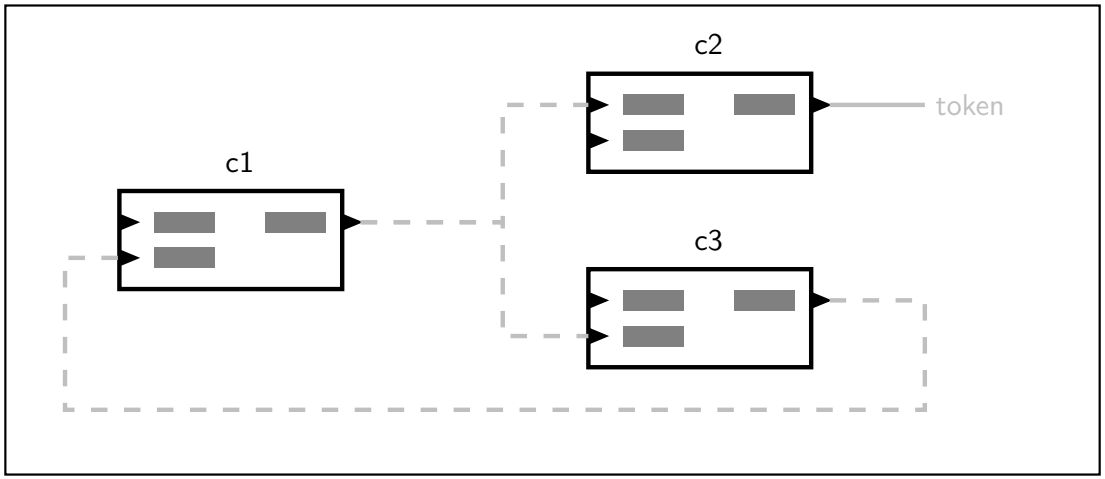
Feuille *stateB*

1. état initial

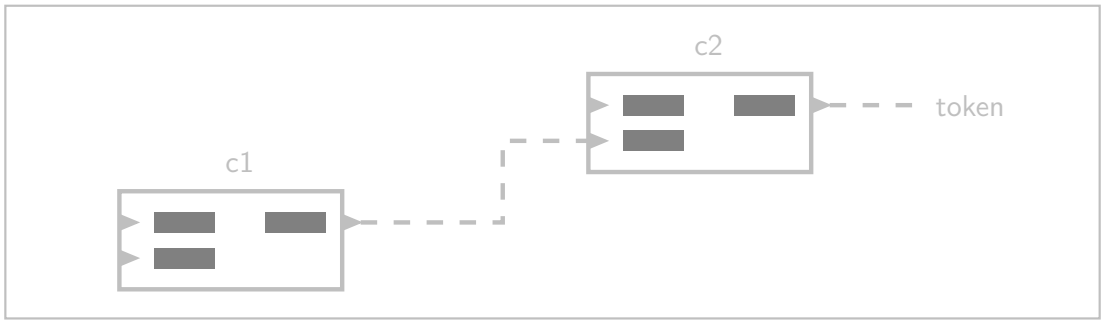
Basculer d'une configuration opérationnelle à une autre



Automate
(contrôleur)



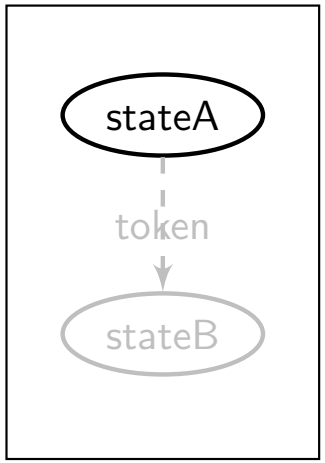
Feuille *stateA*



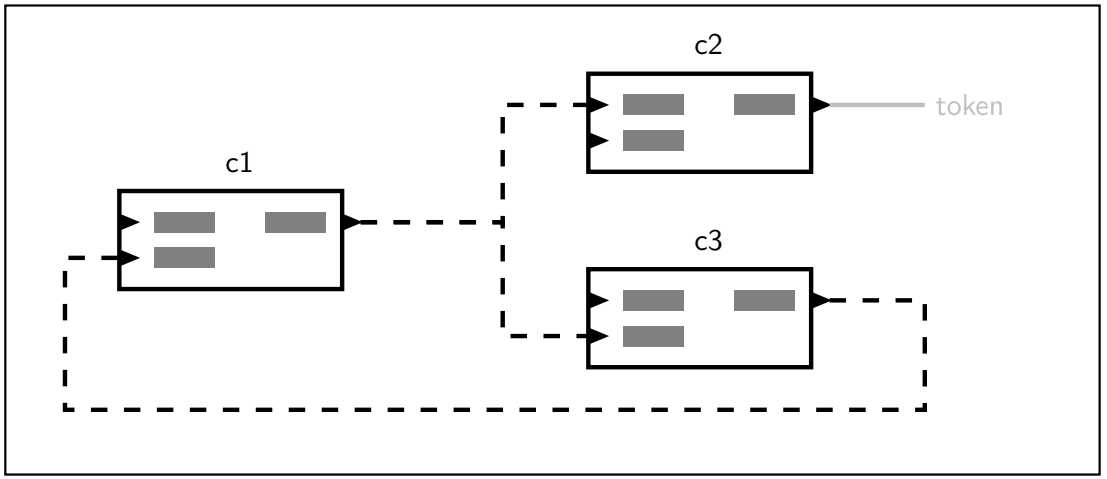
Feuille *stateB*

2. mise en place de la feuille *stateA* (invocation de preconnexion)

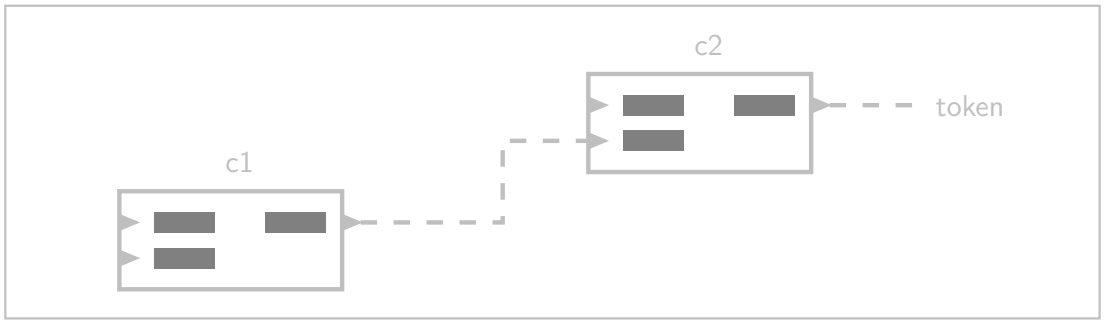
Basculer d'une configuration opérationnelle à une autre



Automate
(contrôleur)



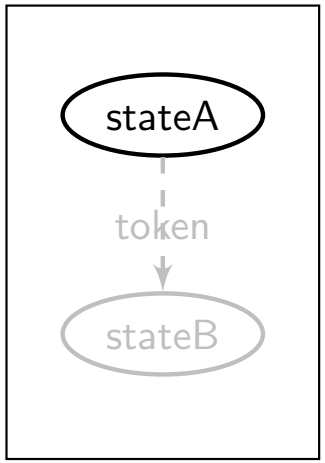
Feuille *stateA*



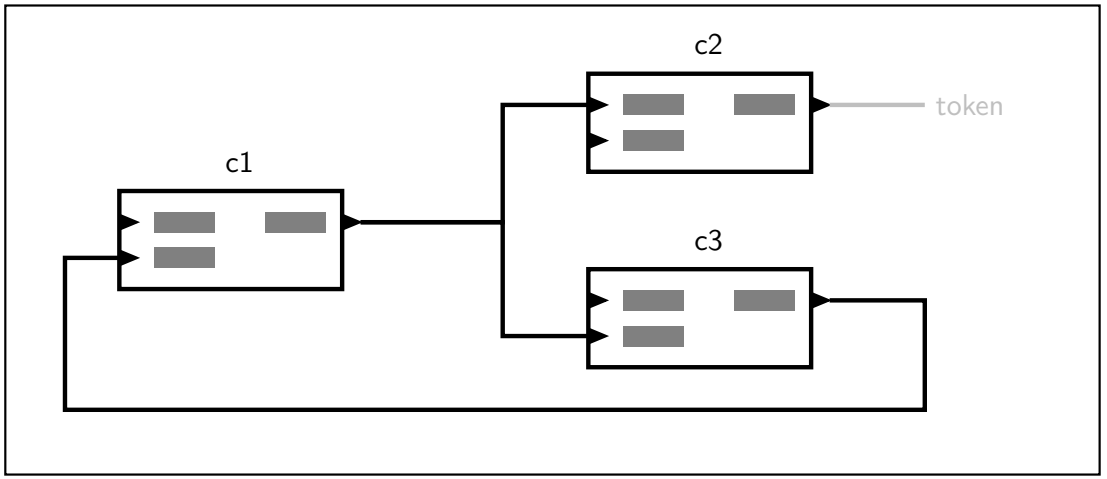
Feuille *stateB*

3. connexion de la feuille *stateA*

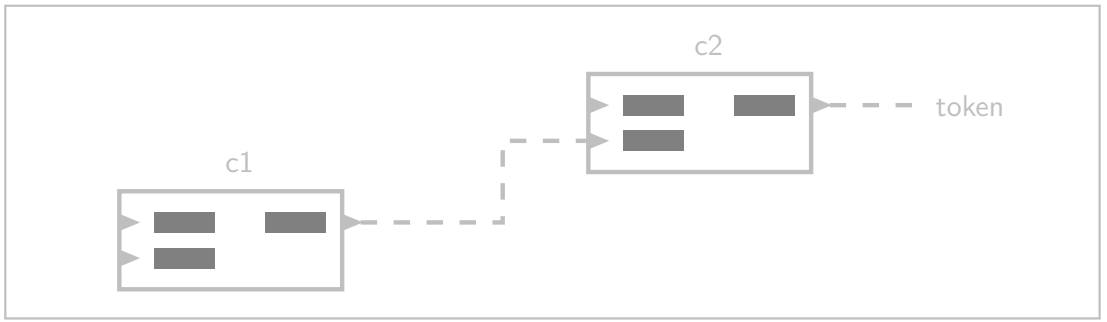
Basculer d'une configuration opérationnelle à une autre



Automate
(contrôleur)



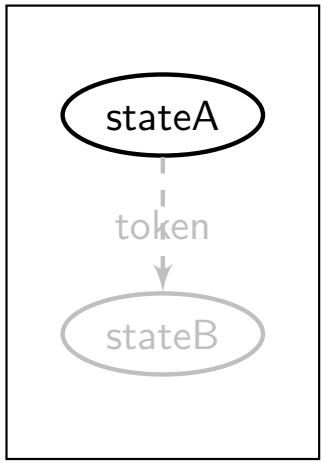
Feuille *stateA*



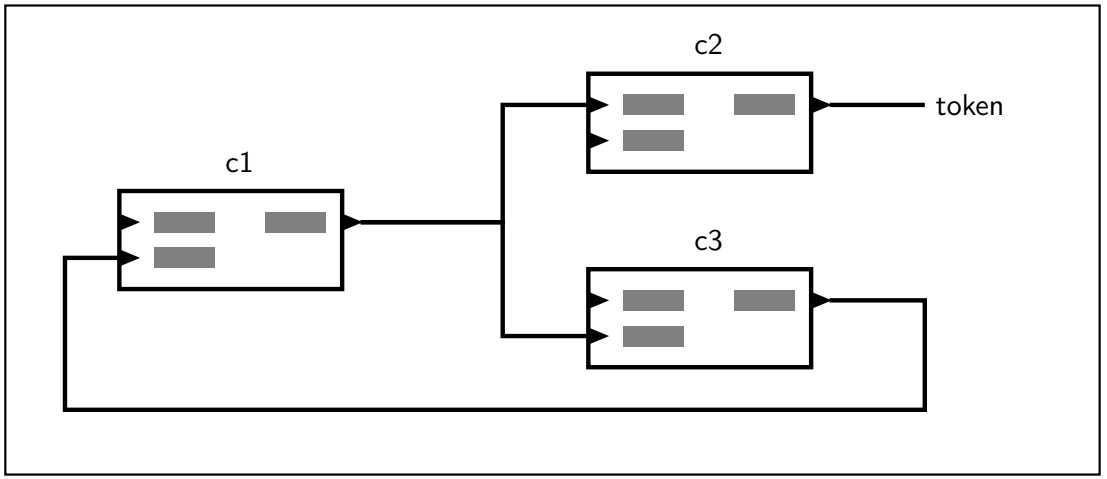
Feuille *stateB*

4. invocation des post-connexions pour la feuille *stateA*

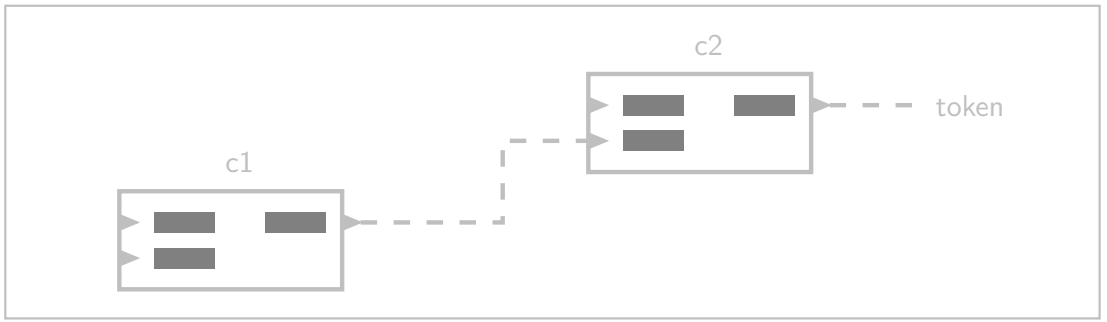
Basculer d'une configuration opérationnelle à une autre



Automate
(contrôleur)



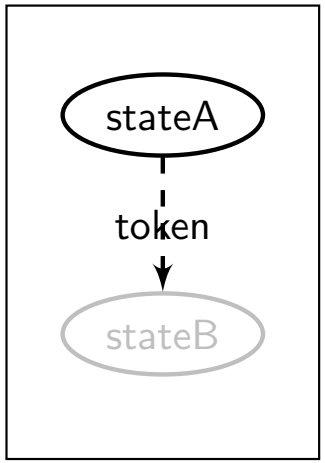
Feuille *stateA*



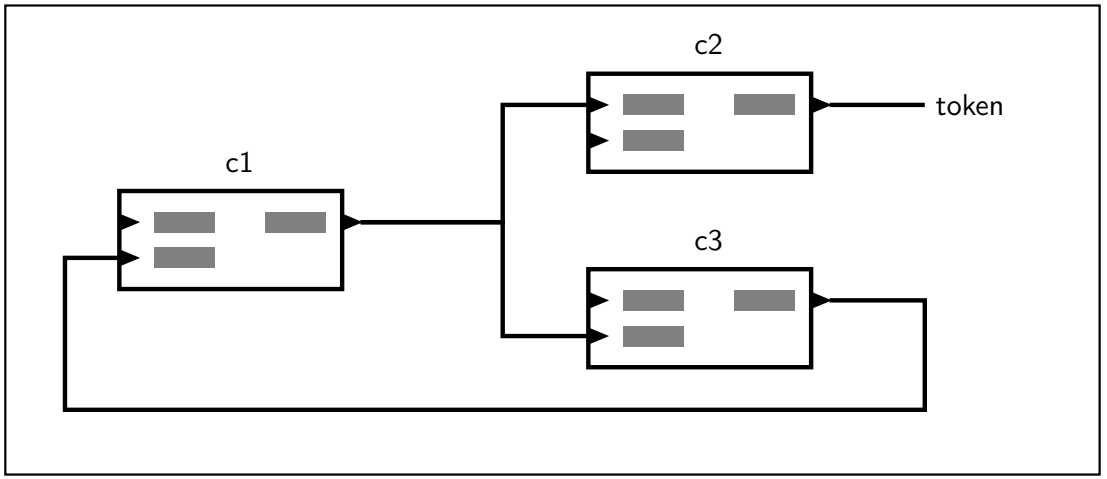
Feuille *stateB*

5. émission de jeton (*token*)

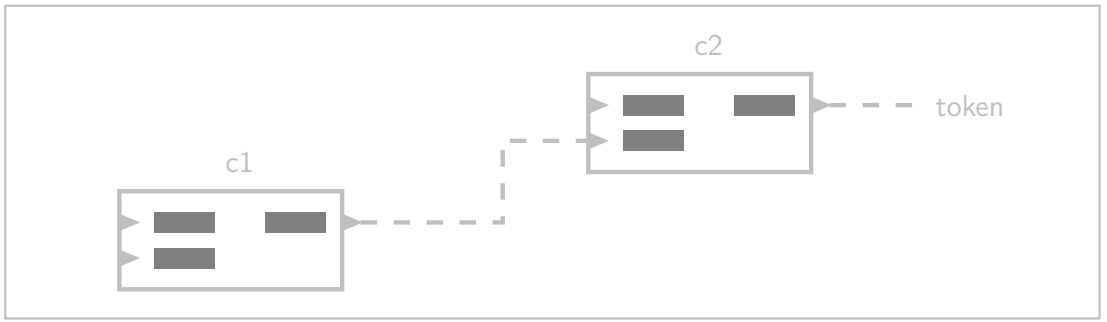
Basculer d'une configuration opérationnelle à une autre



Automate
(contrôleur)



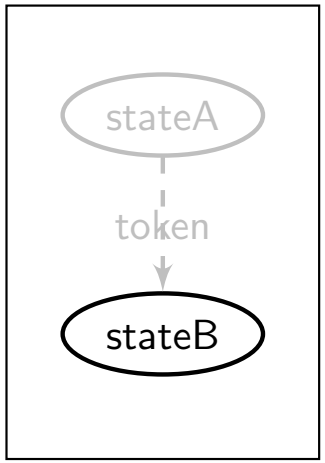
Feuille *stateA*



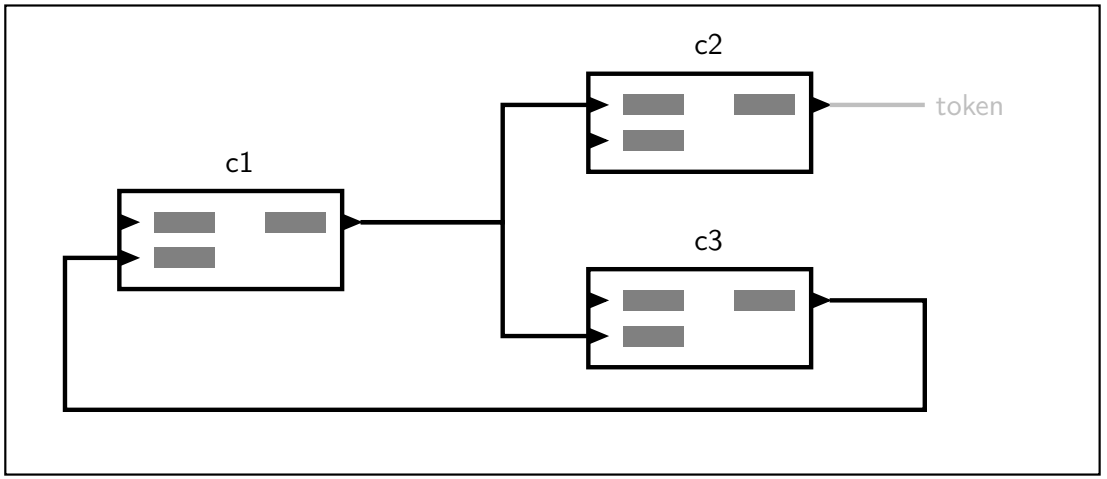
Feuille *stateB*

6. activation de la transition

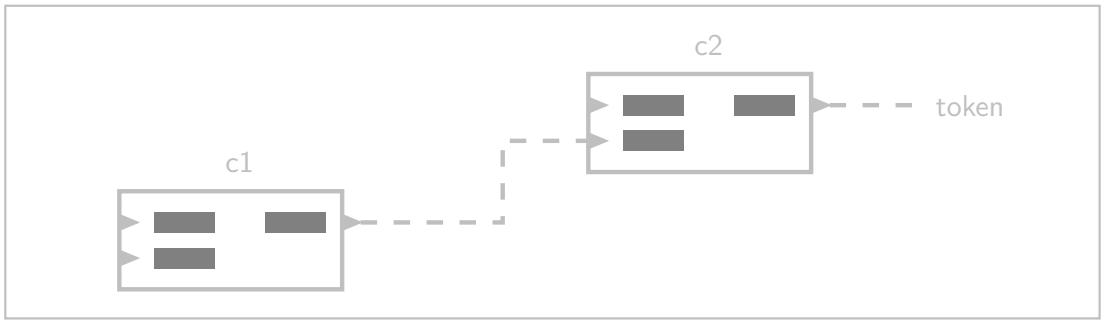
Basculer d'une configuration opérationnelle à une autre



Automate
(contrôleur)



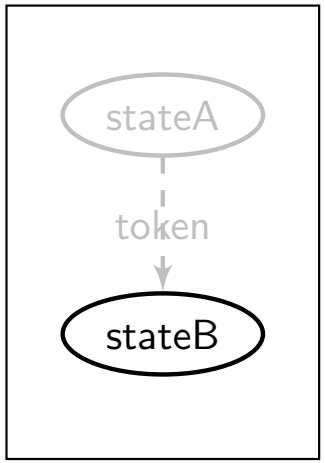
Feuille *stateA*



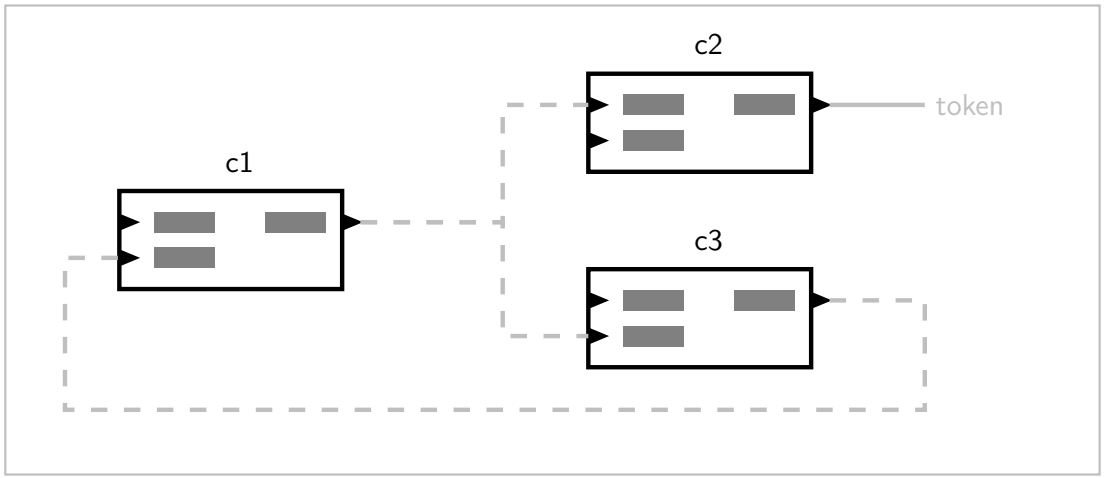
Feuille *stateB*

7. bascule de l'état de l'automate

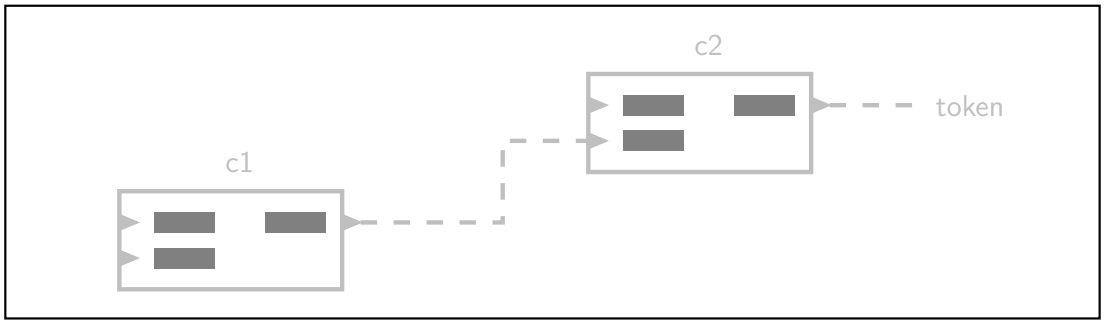
Basculer d'une configuration opérationnelle à une autre



Automate
(contrôleur)



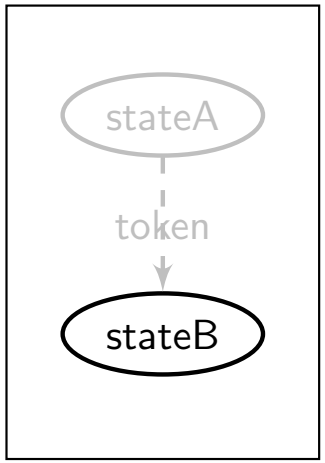
Feuille *stateA*



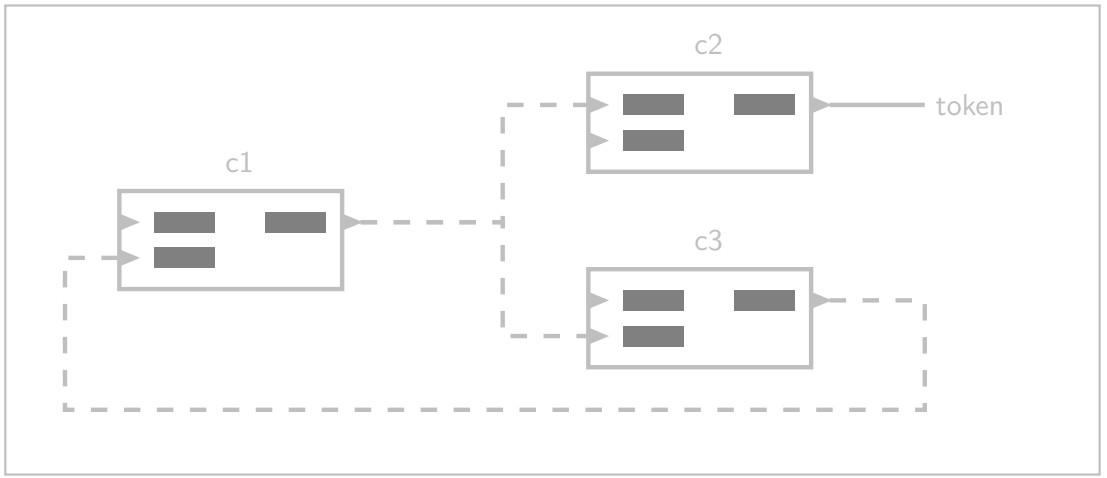
Feuille *stateB*

8. déconnexion de la feuille *stateA*

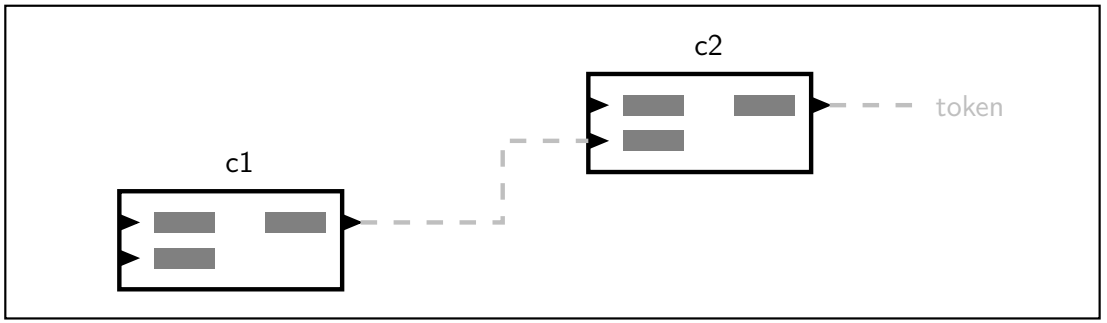
Basculer d'une configuration opérationnelle à une autre



Automate
(contrôleur)



Feuille *stateA*



Feuille *stateB*

9. mise en place de la feuille *stateB*

1 Concepts

2 Architecture du framework

3 Développer les composants

4 Décrire une application

Arborescence

Nom	**Contenu**
`build`	Scripts nécessaires au moteur d'ARCS.js
`components`	Répertoire des composants de l'application
`deps`	Dépendances (scripts) requis par les composants
`docs`	Documentation (API moteur et composants)
`tests`	Répertoires des applications

Développer avec ARCS.js

Séparation des phases de développement

1. Développement des composants :
   - Ils peuvent être réutilisés pour d'autres applications.
2. Description de l'application :
   - Elle sera chargée et interprétée par un moteur d'exécution.

Fonctionnement du moteur

Côté navigateur

Étapes pour exécuter une application

1. Chargement d'un fichier HTML ;
2. Exécution de `require.js` ;
3. Exécution de `arcs_browser.js`, moteur d'ARCS ;
4. Chargement des bibliothèques de composants ;
5. Instanciation des composants ;
6. Mise en place de la première feuille ;
7. ...

1 Concepts

2 Architecture du framework

3 Développer les composants

4 Décrire une application

Les bibliothèques de composants (1/3)

Principe

- contient la définition de plusieurs composants ;
- se traduit sous la forme d'un script (fichier) javascript ;
- respecte une structure particulière.

Structure d'une bibliothèque

```
arcs_module( fonction , [dependances] );
```

- `arcs_module` définit une bibliothèque ;
- `fonction` est la fonction exportant les composants du module ;
- `dependances` est la liste des scripts (tableau) à gérer avant de charger la bibliothèque.

Les bibliothèques de composants (2/3)

Dépendances

- Contient des chaînes de caractères ou des objets :
- Décrit où sont les dépendances par rapport à la racine du framework.

Format des dépendances

- Chaîne de caractères – dans le cas où la dépendance est un module au format AMD (format de module javascript).
  - Voir bibliothèque `arviewer.js`
- Objet – dans le cas où un objet global est défini dans le script et est importable. Champs :
  - **name** : chemin de la dépendance (sans l'extension `.js`) ;
  - **exports** : nom de l'objet global exporté par la dépendance.
  - Voir bibliothèque `arucodetector.js`

Les bibliothèques de composants (3/3)

Fonction d'export

- Prend en entrée un nombre variables de paramètres ;
- Le premier est toujours le module contenant le moteur d'ARCS ;
- Les suivants sont les objets générés par les dépendances (un objet par dépendance) ;
- Doit retourner un objet dont les propriétés (champs) sont les constructeurs des composants.

Définir un composant (1/3)

Définir un composant

- Un composant = un prototype (classe) javascript ;
- Ajout de traits particuliers aux composants ;
- Déclaration explicite des signaux et des slots.

Créer un composant

```
ARCS.Component.create( constructeur, [slots, [
    signaux]])
```

- `constructeur` : constructeur du prototype
  - Peut accepter un paramètre pour initialisation.
- `slots` : tableaux des noms de slots ;
- `signaux` : tableaux des noms de signaux.

Définir un composant (2/3)

Émettre un signal

```
this.emit(nomSignal, [parametres ...])
```

- `nomSignal` : nom du signal à émettre ;
- `parametres` : valeurs à passer au signal.

Exemple de bibliothèque avec composant

```
arcs_module(function (ARCS) {
    var Loop = ARCS.Component.create(
        ...
    );
    return {Loop : Loop};
});
```

Définir un composant (3/3)

Exemple de composant

```
var Loop = ARCS.Component.create(
    function () { // constructeur
        this.setIterations = function (n) {
            var i;
            for (i = 0; i < n; i++) {
                this.emit("newIteration", i);
            }
            this.emit("sendToken", "end");
        };
    },
    ["setIterations"], // liste des slots
    ["sendToken", "newIteration"] // liste des signaux
);
```

1 Concepts

2 Architecture du framework

3 Développer les composants

4 Décrire une application

Fichiers à mettre en place

Fichiers requis

- Fichier HTML : permet le chargement et l'exécution dans le navigateur ;
- Fichier JSON : description d'une application.

Autre fichiers

- Feuilles de style, etc ...
- Fondamentalement une application Web s'exécutant dans le navigateur !

Fichier HTML (1/2)

Structure du fichier HTML

```
<html>
    <head>
        <title>...</title>
        <script data-main="../../build/arcs_browser"
                data-base-url="../.."
                data-arcsapp="arcsapp.json"
                src="../../deps/requirejs/require.js">
        </script>
    </head>
    <body><!-- reste du fichier HTML --></body>
</html>
```

Fichier HTML (2/2)

La balise script

- Format imposé ;
- Charge `require.js`, bibliothèque de chargement de modules ;
- Autres attributs de la balise :
  - `data-main` : chemin relatif vers le moteur d'ARCS.js ;
  - `data-base-url` : chemin relatif vers le répertoire d'installation d'ARCS ;
  - `data-arcsapp` : chemin relatif vers la description de l'application ;

Structure de la description d'application (1/6)

Description d'application

```
{   // description application
    "context" : {
        // liste de bibliothèques, chemin depuis la
            base
        "libraries" : [ ... ],
        // tableau associatif de composants pour
            instanciation
        "components" : { ... }
    },
    // identification de la machine à état
    "controller" : ... ,
    // tableau associatif de feuilles
    "sheets" : { ... }
}
```

Structure de la description d'application (2/6)

Description des feuilles

```
{ // description d'une feuille
    // liste d'invocations de preconnexion
    "preconnections" : [ ... ],
    // liste de connexions
    "connections" : [ ... ],
    // liste d'invocations de post-connexion
    "postconnections" : [ ... ],
    // liste d'invocations de nettoyage
    "cleanups" : [ ... ]
}
```

Instanciation d'un composant

```
// première façon
{ "type" : "..." }
// deuxième façon, en passant un objet au constructeur
{ "type" : "...", value : ... }
```

Structure de la description d'application (3/6)

Exemples de description de composant

```
"components" : {
    "loop": { "type": "Loop" },
    "console": { "type": "Console", "value": "output"},
    "statemachine" : {
        "type": "StateMachine",
        "value" : {
            "initial": "start",
            "final": "end",
            "transitions" : {
                "start" : { "end" : "end"}
            }
        }
    }
}
```

Structure de la description d'application (4/6)

Spécifier une invocation

```
{ // spécification d'une invocation
    // composant sur lequel effectuer l'invocation
    "destination" : "...",
    // slot à invoquer
    "slot" : "...",
    // valeurs à passer (tableau)
    "value": [...]
    // chaque élément du tableau correspond à un
        argument du slot
}
```

Structure de la description d'application (5/6)

Exemples d'invocations

```
"postconnections" : [
    { "destination":"viewer", "slot":"setWidgets",
      "value":["container", "video"]},
    { "destination":"viewer", "slot":"setFocal",
      "value":[600]},
    { "destination":"pong", "slot":"createScene",
      "value":[] }
]
```

Structure de la description d'application (6/6)

Spécifer une connexion

```
{ // spécification d'une connexion
    // composant à l'émission
    "source" : "...",
    // signal émis
    "signal" : "...",
    // composant à la réception
    "destination" : "...",
    // slot à invoquer
    "slot" : "..."
}
```

Exemple de connection

```
"connections" : [
    { "source": "video", "signal": "onImage",
      "destination": "detector", "slot": "detect"}
]
```