Description de Role4All

**Role4All** est un outil de fédération architecturé autour du concept de ***rôle***, un ***rôle*** étant défini ici comme un élément de modèle dont les instances sont associées à des instances de ***players***. Un « ***player*** » étant un méta-modèle représentant un outils ou langage de modélisation [[[1]](#endnote-1)]. Cette définition permet de créer un modèle de rôle séparé des modèles à manipuler puis d’associer ce modèle de rôles aux éléments des modèles de travail.

Depuis sa création Role4All a connu plusieurs versions, au départ pensé comme un outil de création de point de vue son méta-modèle fut repensé afin d’ajouter aux fonctionnalités déjà proposé la notion de fédération. Cela impliqua alors la mise en place d’une communication bidirectionnelle entre les outils utilisé (PimCa, Graffiti, …) et Role4Al. C’est pourquoi des relations ont été créé (entre instances de ***Role***) et des concepts ont été élargies (la notion de contexte). Dans cette partie nous présenteront tout d’abord la dernière version du méta-modèle de Role4All, puis nous détaillerons la notion de fédération dans Role4All et enfin l’utilisation des contextes dans notre outil.

# Le méta-modèle de Role4All

Role4All est un outil au méta-modèle simple reposant sur quatre class principales :

* La classe ***Player*** modélisant les utilisateurs de Role4All. Un ***player*** (une instance de la classe ***Player***) est alors un élément de modèle d’un outil relié à Role4All. Role4All permet de fédérer des player et d’offrir des points de vue sur ces derniers.
* La classe ***Role*** modélisant un type de point de vue. Un ***role*** (une instance de la classe ***Role***) est un point de vue appliqué à un player particulier.

Le méta-modèle de ***Role4All*** présente plusieurs spécificités :

* Il existe deux classes différentiées, l’une adapté au ***rôle*** nommé ***Role*** et l’autre adapté aux éléments de modèle jouant des rôles nommé ***Player.***

Cela permet de considéré le ***rôle*** comme un élément indépendant des modèles de travail mais aussi d’associé un ***rôle*** à un ou plusieurs éléments de modèle. On a alors une indépendance entre ***rôle*** et ***player*** ce qui permet une allocation dynamique des rôles.

* La classe ***Role*** hérite de la de la classe ***Player.***

Cette relation permet à une instance de ***rôle*** de jouer un ***rôle*** c’est-à-dire de considérer une instance de la classe ***Role*** comme une instance de modèle et de lier cette instance à un ***rôle***. Cela permet, entre autre, de créer plusieurs points de vue sur un modèle ou de centraliser dans un même point de vue des informations provenant de divers outils.

* La classe ***Role*** possède deux variables d’instance référençant des éléments d’elle-même : ***containedRoles*** et ***containerRole***.

La variable ***containedRoles*** est une liste d’instance de rôle. La variable ***containerRole*** référence l’instance de ***rôle*** contenant l’instance en question. Ces deux variables permettent de crée des groupes d’instances de ***rôle*** avec un conteneur et des contenus. Cette relation permet de créé une relation bidirectionnel indirecte entre plusieurs instance de ***rôle***. Il est alors possible d’ajouter ou de retirer une des instances de ***rôles*** contenu sans interféré avec les autres, on a donc un système de mise en relation dynamique des instances de ***rôles***.

* Les classes ***Role***, ***Player***, ***PlayRelation*** et ***DynamiqueAdapter*** sont les classes structurantes de ***Role4All***.

Role4All permet à une instance de modèle de jouer un ***rôle***. Cela se traduit par le fait qu’une instance de ***Player*** et une de ***Role*** réfèrent une même instance de ***PlayRelation*.** La relation entre instance de ***player*** et instance de ***rôle*** est alors caractérisée grâce à un adaptateur (une instance de ***DynamiqueAdapter***). La dissociation entre élément de modèle (***players***), les points de vue sur un élément (***rôles***) et caractérisation de ce point de vue (***adapters***) rend les éléments structurants de ***Role4All*** indépendant les uns des autres. C’est pourquoi ***Role4All*** peut être considéré comme un outil de fédération dynamique.

Role4All permet.

De ce fait Role4All est semblable à un outil de fédération car il permet de regrouper des informations provenant de plusieurs modèles en se basant sur un concept (un rôle) commun. Cependant la première version de Role4All (présenté lors de la thèse de Jean-Philippe SCHNEIDER [15]) créait un lien unidirectionnel entre les modèles des outils et Role4All, ce qui est incompatible avec la notion de fédération (nécessitant une relation conceptuelle bidirectionnelle). De plus il n’existait pas de liens entre deux instances d’éléments de modèle jouant le même rôle, ce qui dans l’architecture de Role4All est nécessaire à la fédération. La partie suivante présente l’architecture de Role4All dans sa première version.

(par la suite nous réutilisons un texte en anglais)

Role4All language is mainly based on four main concepts: *Player*, *Role*, *DynamicAdapter* and *PlayRelation*.

The figure 4 illustrates the relation between Role4All’s classes through the example of the role *RoleFPGA* played by the Pimca element *pimcaMachinery0* and the Excel element *excelGroup0*. This example is fully described in the next chapter.

**DynamicAdapter**

RoleFPGA

ExcelGroup

PimcaMachinery

DynamicAdapterFPGA\_Excel

DynamicAdapterFPGA\_Pimca

**0..1 player**

**\* playRelations**

**adapter**

**0..1 role**

**0..1 playRelation**

**PlayRelation**

**Player**

**Role**

Conform to Conform to

**Role4All model**

**A role model**

**Instances**

**0..1**

Figure 1: Méta-Modèle de Role4All

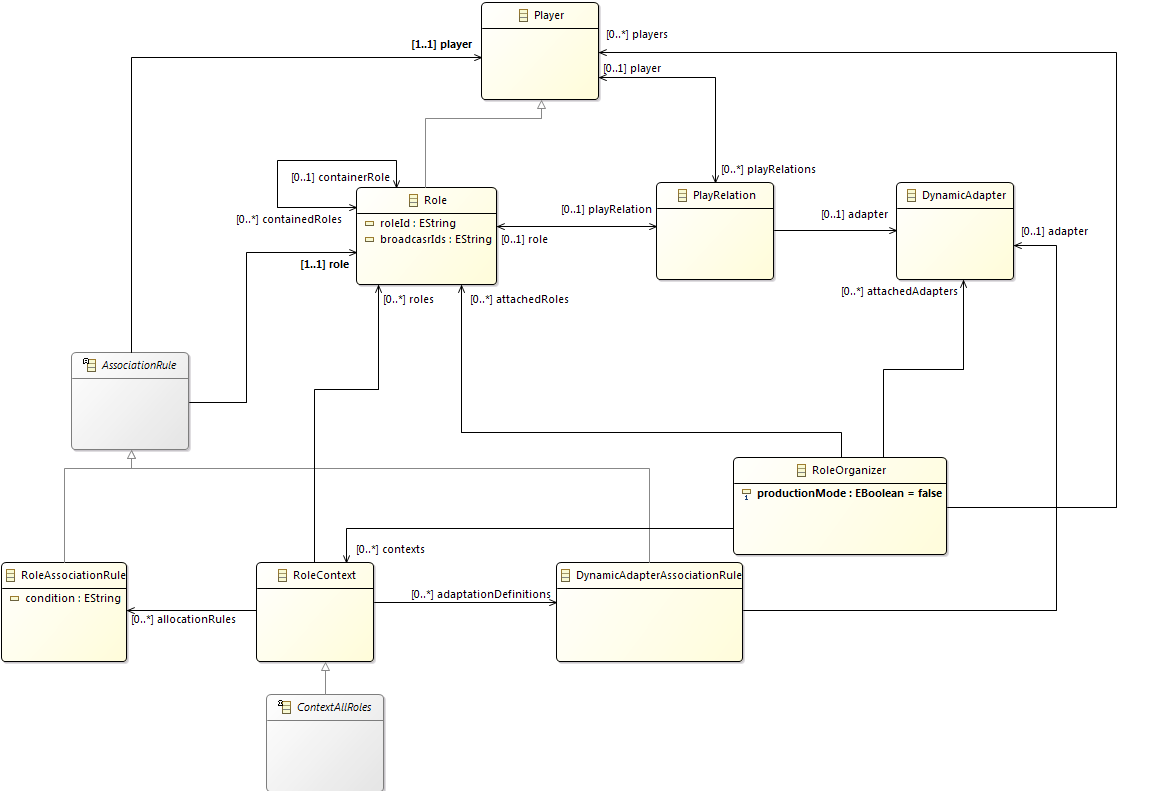
****

Figure 7: Role4All’s meta-model illustrates with the example of the role of FPGA

The elements extending the class *Player* are the model type of the language elements; they are called “players”. For example *PimcaMachinery* (cf figure 7 for all element names) is an element of Pimca’s meta-model and p*imcaMachinery0* is an element of Pimca’s model (detailed after in the figure 10). We want Pimca’s model element *pimcaMachinery0* to play the role of *RoleFPGA,* therefore the model element *PimcaMachinery* is defined as a *Player*. The connection between a role model element *RoleFPGA* and a tool element *pimcaMachinery0* is transformed by an adapter.

The concepts conformed to the Role class, are called “roles”. As an example in the figure 7 *RoleFPGA* is a role.

We can notice that stating “In Role4All a role can play a role” is equivalent to: “a role can be a player” or “the class *Role* extends the class *Player*”.

The elements created through the class *DynamicAdapter* allow transforming a model element for its role, they are called “adapter”. The adapters define the behavior of the relations between players and roles. In the figure 7 *DynamicAdapterFPGA\_Pimca* is the adapter between the instances of *PimcaMachinery* and the role *RoleFPGA* and *DynamicAdapterFPGA\_Excel* is the adapter between the instances of *ExcelGroup* and the role *RoleFPGA*

The elements from the class *PlayRelation* are connectors without behavior between three elements: a role, a player and an adapter.

So according to figure 7, the player *pimcaMachinery0* plays the role *RoleFPGA* and the behavior of this relation is defined in the adapter *DynamicAdapterFPGA\_Pimca*.

Finally a Pimca element and an Excel element play the same role, so we created a unique point of view on two different elements from the two different tools. Now we can use the role *RoleFPGA* to handle the concept of FPGA instead of the tools Excel and Pimca.

1. Jean-Philippe Schneider, Les rôles : médiateurs dynamiques entre modèles système et modèles de simulation. Thèse de doctorat, Université de Bretagne Occidentale, Novembre 2015. [↑](#endnote-ref-1)