Mini-Projet : Ingénierie dirigée par les modèles

Sujet : Chaîne de vérification de modèles de processus

I) Création des métamodèles

Dans un premier temps je me suis concentré sur la création des métamodèles.

On dispose de deux métamodèles :

- Le métamodèle SimplePDL qui permet de décrire des processus de développement logiciel.
- Le métamodèle PetriNet qui permet de décrire des réseaux de Petri.

J'ai choisi de définir ces métamodèles dans des projets Ecore Modeling et de configurer la génération de code Java pour ces métamodèles (`.genmodel`) afin que le code généré soit disponible dans les sources d'un même projet.

1) Métamodèle SimplePDL

Pour définir le métamodèle SimplePDL, j'ai utilisé l'outil EMF (Eclipse Modeling Framework) qui permet de définir des métamodèles sous forme de diagrammes de classes UML. Les quelques choix de conception que j'ai fait sont les suivants :

- J'ai choisi de définir les relations bi-directionnelles via les objets `eOpposite` pour simplifier la navigation dans le modèle.
- J'ai choisi d'interfacer la connection entre les `WorkDefinition` et les `Resource` à travers une AssociationClass `ACResource` pour ajouter des informations supplémentaires à la relation.
- J'ai choisi de définir des valeurs différentes pour les littéraux de l'énumération `WorkSequenceType` afin de pouvoir les distinguer plus facilement dans du code Java.

2) Métamodèle PetriNet

Pour définir le métamodèle PetriNet, j'ai utilisé l'outil EMF (Eclipse Modeling Framework) qui permet de définir des métamodèles sous forme de diagrammes de classes UML. Les quelques choix de conception que j'ai fait sont les suivants :

- J'ai choisi de définir les relations bi-directionnelles via les objets `eOpposite` pour simplifier la navigation dans le modèle.
- J'ai ajouté des noms aux transitions pour pouvoir les identifier plus facilement.
- J'ai ajouté l'opposite de la relation arc -> transition pour pouvoir récupérer les arcs d'une transition

II) Transformation SimplePDL vers PetriNet

Dans un second temps, j'ai implémenté une transformation de modèle SimplePDL vers PetriNet.

Pour cela j'ai décidé de définir une première classe `WorkDefinitionToPetriNet` qui permet de transformer une `WorkDefinition` en un `PetriNet` équivalent. Cela me permet de transformer un processus de développement logiciel en un réseau de Petri selon un schéma bien défini puis de connecter ces réseaux de Petri entre eux afin de définir des `WorkSequence`.

J'ai fait le même choix pour représenter les `Resource` dans les `PetriNet` pour les mêmes raisons que pour les `WorkDefinition`.

Dans ces deux classes `WorkDefinitionToPetriNet` et `ResourceToPetriNet`, j'ai exposé publiquement les attributs, à l'aide de getters et setters, qui permettent de connecter ces éléments entre eux pour réaliser la transformation.

Cette transformation est réalisée en deux étapes :

1. Transformation des `WorkDefinition` en `PetriNet` : Pour chaque `WorkDefinition` on crée un `PetriNet` équivalent.
2. Connexion des `PetriNet` entre eux : Pour chaque `WorkSequence` on connecte les `PetriNet` entre eux en choisissant l'`Arc` approprié au `WorkSequenceType`.
3. On ajoute les `Resource` aux `PetriNet` en fonction des `ACResource` associés aux `WorkDefinition`.

J'ai réalisé la même transformation avec ATL.

III) Tranformations modèles à textes avec Acceleo

J'ai décidé d'utiliser seul plugin pour Acceleo avec deux sous-packages pour les transformations model-to-text des deux métamodèles.

IV) Syntaxe concrète graphique avec Sirius

J'ai décidé de créer une syntaxe concrète graphique pour les deux métamodèles SimplePDL et PetriNet. J'ai défini toutes les requêtes et contraintes avec la syntaxe `AQL` (Acceleo Query Language).

1) SimplePDL

J'ai défini les `Resource`s comme des nodes avec des bordered nodes pour les quantités. Les `ACResource`s sont des edges entre les `WorkDefinition`s et les `Resource`s.

2) PetriNet

Pour les arcs j'ai dû spécifier le sens de l'arc à l'aide d'une expression `AQL` car ces derniers peuvent aller de `Place` à `Transition` mais aussi dans le sens inverse.

V) Syntaxe concrète textuelle avec Xtext

J'ai décidé de créer une syntaxe concrète textuelle pour le métamodèle SimplePDL.
Cette syntaxte est définie de la manière suivante :

```
proc test {
        res Redacteur   (1),
        res Concepteur  (3),
    res Developpeur (2),
    res Testeur     (2),
    res Machine     (4),
    wd Conception {
        need Concepteur : 2,
        need Machine    : 2
    },
    wd Programmation {
        need Developpeur : 2,
        need Machine     : 3
    },
    wd RedactionDoc {
        need Redacteur  : 1,
        need Machine    : 1
    },
    wd RedactionTests {
        need Testeur : 1,
        need Machine    : 2
    },
    ws {
        type f2f,
        from Programmation,
        to RedactionTests
    },
    ws {
        type s2s,
        from Conception,
        to RedactionTests
    },
    ws {
        type f2s,
        from Conception,
        to Programmation
    },
    ws {
        type f2f,
        from Conception,
        to RedactionDoc
    },
    ws {
        type s2s,
        from Conception,
        to RedactionDoc
    }
}
```