MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE ABBES LAGHROURDE KHENCHELA
FACULTE DES SCIENCES ET DE LA TECHNOLOGIE
DEPARTEMENT MATHEMATIQUE ET INFORMATIQUE

Mémoire de fin d'études

Pour l'obtention du diplôme de Master (L.M.D)

Spécialité : Sécurité et technologie web

Génération des cas de test à partir d'une spécification UML utilisant les Diagramme d'Activité et basée sur L'outil de transformation de modèle TGG

Présenté par :

MIRA Haroun

Dirigé par :

Dr. MESSAOUDI .Nabil

Année universitaire : 2018/2019

Dédicaces

Je dédie ce modeste travail

Je dédie ce travail à mes très chers parents qui m'ont

Affectueusement soutenus tout le long de mes études sans jamais faillir

A ma chère et tendre mère, source d'affectation de courage et d'inspiration qui a autant sacrifié pour me voir atteindre ce jour. À l'homme qui m'a toujours guidé vers le droit chemin avec son amour ses sacrifices et encouragements son soutien moral,À mon cher Père « Oh Allah, aie pitié de mon père ».

A tous mes frères " hamid , abdesadek ".

A toute la famille mira.

A toute la promo 2019, et à tous mes amis sans exception.

A tous ceux que je porte dans mon cœur.

MIRA HAROUN .

Remerciements

Au premier lieu, nous tenons à remercier Dieu qui nous a donné le courage et la volonté pour terminer ce travail.

Un très grand merci à :

Notre promoteur dr . nabil messaoudi pour nous avoir encadrés et orientés tout au long de cette recherche.

Nousremercie chaleureusement nos parents et nos familles.

Nousremercions aussi l'ensemble des enseignants du département Informatique

Que tous ceux ou celles qui nous ont apportés leur soutien et qui nous ont aidé de loin ou de prés pour l'achèvement de ce projet trouvent ici l'expression de notre vive et sincère reconnaissance, en particulier nos parents, nos familles et nos amis.

- *MIRA HAROUN .*

Résumé

UML est actuellement largement accepté dans le secteur du développement de logiciels. Quelque diagrammes d'UML sont utilisés pour modéliser les aspects la structure d'un système et d'autres pour modéliser les aspects comportementaux. les diagrammes d'interaction sont largement utilisés pour modéliser le comportement dynamique en UML par exemple comme diagramme d'activité, et aussi pour modéliser l'interaction entre un ensemble d'objets à travers les messages qui peuvent être envoyés entre eux. Toutefois, l'UML est un langage semi formel qui n'a pas des constructions définies exactement. L'objectif visé dans ce mémoire et de proposer une approche automatique de transformation des diagrammes d'activité d'UML vers cas de teste, Les diagrammes d'activité sont utilisés pour modéliser les systèmes workflow

Les tests unitaires, est réalisés a visé la phase de développement, sont Détachable, les tests utilisés il ya deux stratégies pour test : les tests fonctionnels et les tests structurels

Le but dans ce mémoire et de proposer une approche automatique de transformation des diagrammes d'activité d'UML vers cas de test , basée sur la transformation de graphes, et réalisée à l'aide de l'outil TGG , en appliquant une sémantique formelle qui est bien précise dans notre mémoire. Notre approche consiste à proposer un méta modèle des diagrammes d'activité, un méta modèle de cas de teste et une grammaire de graphes.

Mot clés : Diagrammes d'activité d'UML, cas de test, Transformation de graphes, TGG

Abstract

Nowadays ,UML is widely accepted in the software development sector. Some UML diagrams are used to model the structure aspects of one system and others to model the behavioral aspects. Interaction diagrams are widely used to model dynamic behavior in UML for example as an activity diagram, and also to model the interaction between a set of objects through the messages that can be sent between them. However, UML is a semi-formal language that does not have exactly defined constructs. The objective of this thesis is to propose an automatic approach to transform UML activity diagrams into test cases. Activity diagrams are used to model workflow systems

Unit tests, is performed aimed at the development phase, are Detachable, the tests used there are two strategies for testing: functional tests and structural tests

The purpose in this thesis is to propose an automatic approach of transforming UML activity diagrams to test cases, based on the transformation of graphs, and realized using the TGG tool, by applying formal semantics which is very precise in our memory. Our approach consists in proposing a meta model of activity diagrams, a meta model of test cases and a grammar of graphs.

Key words: UML activity diagrams, test case, Graph transformation, TGG

TABLE DES MATIERES

TABLE DES FIGURES

LISTE DE TABLEAUX

Introduction Générale

UML (**Unified Modeling Language**) est un langage de modélisation graphique à base de pictogrammes. Il est apparu dans le monde du génie logiciel, est considéré comme le langage standard de modélisation visuelle utilisé pour spécifier, visualiser, construire et documenter les artefacts d'un système logiciel, Certains diagrammes d'UML sont utilisés pour modéliser la structure d'un système, d'autres sont utilisés pour modéliser son comportement.

Actuellement, le langage de modélisation objet unifié UML est normalisé par l'OMG comme langage de modélisation des systèmes d'information à objets. les diagrammes d'interaction sont largement utilisés pour modéliser le comportement dynamique en UML

Les diagrammes d'activité sont utilisés pour modéliser les systèmes workflow, les systèmes orientés services et les processus métiers. Le flux de contrôle inclut un support pour le séquencement, le choix, le parallélisme et les événements. Les activités peuvent être regroupées en sous activités et peuvent être imbriquées à différents niveaux.

L'UML est un langage semi formel qui n'a pas des constructions définies rigoureusement.

L'ingénierie Dirigée par les Modèles (IDM) est une discipline récente du génie logiciel qui met l'accent sur les modèles au sein du processus de développement logiciel. La transformation des modèles est une tâche complexe qui nécessite la disposition d'outils flexibles permettant la gestion des modèles et des langages durant la transformation et la manipulation. Triple Graph Grammars (TGGs) définissent la relation entre deux modèles, où les modèles sont représentés comme graphes. Leur avantage principal est que TGGs permettent de définir une transformation bidirectionnelle.

Le but de ce travail est de proposer une approche automatique de transformation

des diagrammes d'activité d'UML vers cas de teste, et réalisée à l'aide de l'outil TGG.

Ce notre travaille est organisé comme suit :

Dans le première chapitre, on va présenter le langage de modélisation unifié UML2.0, nous avons donné une description détaillée des diagrammes d' activité d'UML 2.0 qui sera modélisé dans le modèle source de la transformation , et le deuxième partie , On va présenter l'ingénierie dirigée par les Modèles et langages pour créer et transformer des modèles, et comme outil de transformation de graphe le TGG utilisé dans l'implémentation de notre travail.

Dans le deuxième chapitre on va présenter les testes unitaire et explique les testes structurels (boite blanche) et les testes fonctionnelle (boite noire)

Et enfin on va proposer une approche automatique pour transformer les diagrammes d'activité d'UML2.0 vers cas de teste. La méthode proposée se base sur la transformation de graphe, et utilise l'outil TGG.

Chapitre I : Langage de Modélisation Unifié (UML) L'ingénierie Dirigée par les Modèles

1- Introduction

Le langage de modélisation objet unifié UML (Unified Modeling Language) , est né de la fusion des trois méthodes objet : OMT (Object Modeling Technique) de James Rumbaugh, OOSE (Object Oriented Software Engineering) de Ivar Jacobson, et la méthode de Grady Booch. Puis il est normalisé par l'OMG en 1997 dans sa version 1.1 comme langage de modélisation des systèmes d'information à objets. UML est rapidement devenu un standard incontournable. [1]

Le langage de modélisation unifié (UML) est un langage de modélisation visuelle, utilisé pour spécifier, visualiser, construire et documenter les artefacts d'un système logiciel [2].

Il est utilisé pour comprendre, concevoir, naviguer, configurer, maintenir et contrôler les informations sur les systèmes modélisés [3].

L'ingénierie Dirigée par les Modèles (IDM), ou Model Driven Engineering (MDE) est une discipline récente du génie logiciel qui met l'accent sur les modèles au sein du processus de développement logiciel. L'IDM est donc le domaine de l'informatique mettant à disposition des outils, concepts et langages pour créer et transformer des modèles. La transformation des modèles est une tâche complexe qui nécessite la disposition d'outils flexibles permettant la gestion des modèles et des langages durant la transformation et la manipulation des modèles. La sémantique de ces derniers doit être spécifiée, et les langages utilisés doivent être décrit de manière précise. [4]

L'approche des grammaires de graphes triples (Triple Graph Grammar : TGG), introduite par Andy Schürr [5] . TGG est spécifié pour les transformations bidirectionnelles (transformation à l'avant et en arrière).

Le but de ce chapitre est de donner une description détaillée de l'un des diagrammes fondamentaux d'UML, qui est le diagramme d'activité et donnons un rappel sur la transformation de modèle et la méta-modélisation. , et les fondements Théoriques de TGG. La transformation de graphe est présentée comme outil de transformation de modèles et TGG comme outil de transformation de graphe.

2- Les Diagrammes UML

UML 2.0 propose treize types de diagrammes (9 en UML 1.3) pour représenter les différents points de vue de modélisation. Ils se répartissent en deux grands groupes :

1- Diagrammes structurels (*Structure Diagram*).

2- Diagrammes comportementaux (Behavior Diagram).

2.1 Diagrammes structurels ou diagrammes statiques

Ces diagrammes permettent de visualiser, spécifier, construire et documenter

l'aspect statique ou structurel du système informatisé [6].

– Diagramme de classes (Class diagram)

– Diagramme d'objets (Object diagram)

– Diagramme de composants (Component diagram)

– Diagramme de déploiement (Deployment diagram)

– Diagramme des paquetages (Package Diagram)

– Diagramme de structure composite (Composite Structure Diagram)

2.2 Diagrammes comportementaux ou diagrammes dynamiques

Les diagrammes comportementaux modélisent les aspects dynamiques du système. Ces aspects incluent les interactions entre le système et ses différents acteurs, ainsi qu la façon dont les différents objets contenus dans le système communiquent entre eux.

– Diagramme des cas d'utilisation (Use Case Diagram)

– Diagramme d'activité (Activity Diagram) : Un diagramme d'activité est une variante des diagrammes d'états-transitions. Il permet de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation. dans un diagramme d'activité les états correspondent à l'exécution d'actions ou d'activités et les transitions sont automatiques.

– Diagramme états-transitions (State Machine Diagram)

– Diagramme de séquence (Sequence Diagram)

– Diagramme de communication (Communication Diagram)

– Diagramme global d'interaction (Interaction Overview Diagram)

– Diagramme de temps (Timing Diagram)

3- Les diagrammes d'activité

3-1 Définition

UML permet de représenter graphiquement les aspects dynamiques des systèmes à l'aide de plusieurs diagrammes comportementaux. Parmi eux, on distingue les diagrammes d'activité qui permettent de mettre l'accent sur les traitements. Ils sont utilisés pour représenter le comportement d'une méthode ou le déroulement d'un cas d'utilisation. [1]

Un diagramme d'activité est une variante des diagrammes d'états-transitions, dans lequel les états correspondent à l'exécution d'actions ou d'activités, et les transitions sont automatiques (les transitions

sont déclenchées par la fin d'une activité et provoquent le début immédiat d'une autre). Un diagramme d'activité peut être attaché à n'importe quel élément de modélisation afin de visualiser, spécifier, construire ou documenter le comportement de cet élément. [7]

Les diagrammes d'activité d'UML constituent un outil de modélisation des systèmes workflows, des modèles orientés service et des processus métiers (ils montrent l'enchaînement des activités qui concourent au processus). Un modèle d'activité consiste en activités liées par des flux de données et de contrôle. Une activité peut varier d'une tâche humaine à une tâche complètement automatisée.

La différence principale entre les diagrammes d'interaction et les diagrammes d'activité, est que les premiers modélisent le flot de contrôle entre objets, alors que les seconds sont utilisés pour modéliser le flot de contrôle entre activités**.**

3-2 Notation

Nous présentons dans cette sous section seulement quelques éléments de modélisation de base des diagrammes d'activité [8]

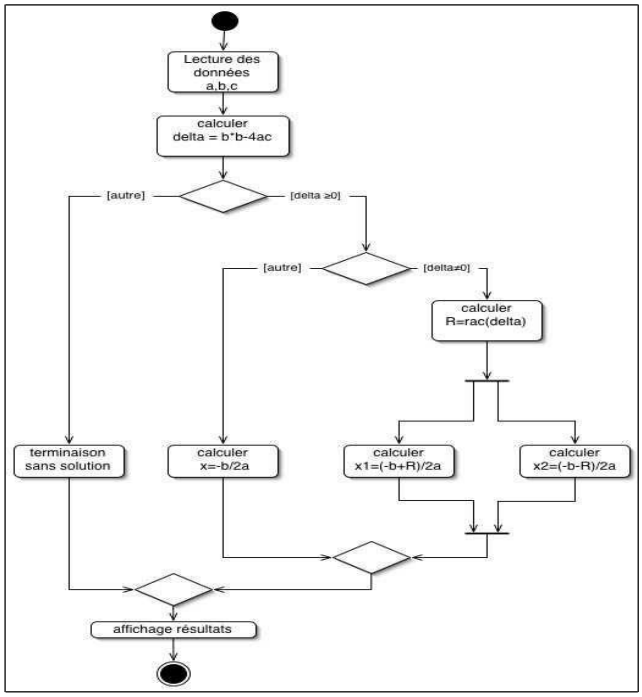

Figure 1. : Exemple de diagramme d'activité.

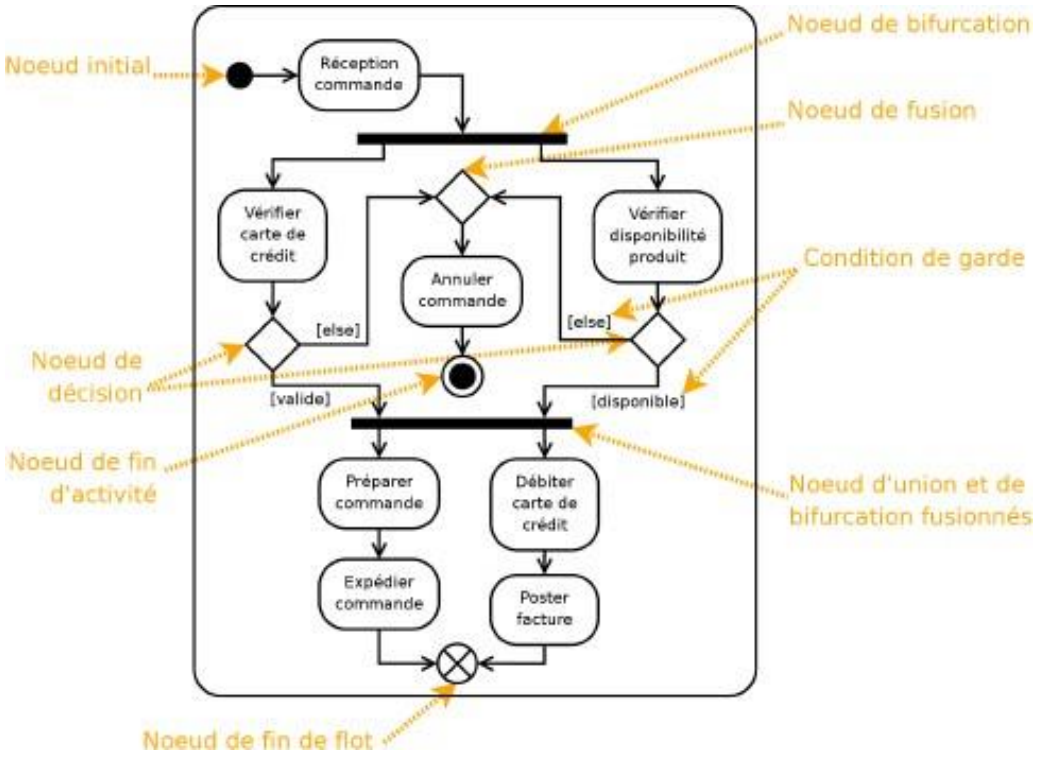

Figure 2 : Exemple illustre l'utilisation des nœuds de contrôle. [8]

1. **Action** : Une action est l'unité fondamentale de la spécification de comportement. L'action prend un ensemble d'entrées et les convertit en un ensemble de résultats, l'une ou les deux ensembles peuvent être vide. Une action ne peut être décomposée en actions plus simples. Une action peut être, par exemple :
   - une affectation de valeur à des attributs
   - la création d'un nouvel objet ou lien
   - l'émission d'un signal
   - la réception d'un signal
   - etc. La notation d'une action est un rectangle avec des coins arrondis. La figure 2-1 montre trois exemples d'actions.

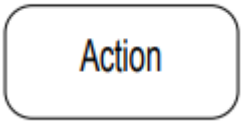

Figure 2-1 –Notation d action

2. Activité : Une activité contient des séquences d'actions et / ou d'autres activités. On utilise les activités pour grouper des séquences d'actions ensemble.

3. Nœud de contrôle : On utilise les nœuds de contrôle pour guider le flux de contrôle (et le flux d'objets) à travers un groupe d'activités et d'actions. Les nœuds de contrôle viennent dans une variété de formes, en fonction de ce qu'on a besoin, ils servent comme un agent de circulation pour le flux de contrôle et les flux d'objets. Les nœuds de contrôle sont les suivants :

Initial : Un nœud initial est l'endroit où le flux de contrôle commence quand une activité est invoquée. La Figure 2-2 montre la notation d'un nœud initial

Figure 2-2 notation de nœud initial

Final : Un nœud final est un nœud de contrôle dans laquelle un ou plusieurs flux au sein d'une activité donnée s'arrêtent. Il existe deux types de nœuds finaux :

(a) nœud de flux final.

(b) nœud d'activité final.

(a) Nœud d'activité final **(b) Nœud de flux final**

Figure 2-3 Nœud d'activité final et Nœud de flux final

Décision : offre un choix entre deux ou plusieurs activités sortantes, dont chacune a une expression booléenne qui doit résoudre à Vrai avant la prise de chemin. Un nœud de décision apparaît comme un diamant, comme le montre la figure2-4

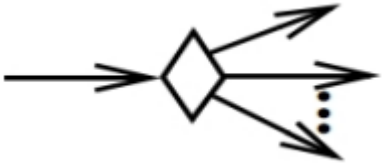

Figure 2-4 Notation de noeud de décision

- **Fusion** (**Merge**) : Regroupe des flux multiples.
- **Bifurcation** : Divise un flux en plusieurs flux simultanés. La figure2-5 montre sa notation.

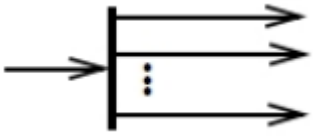

Figure 2-5 – Notation de noeud de bifurcation

Nœud de bifurcation (fork node) Un nœud de bifurcation est un nœud de contrôle qui sépare un flux d'entrée en plusieurs flots concurrents en sortie.

Nœud d'union (join node) Un nœud d'union (nœud de jointure) est un nœud de contrôle qui synchronise des flots multiples. Il possède plusieurs arcs entrants et un seul arc sortant. Ce dernier ne peut être activé que lorsque tous les arcs entrants sont activés. L'exemple suivant illustre l'utilisation des nœuds de contrôle.

Nœud d'objet : Un nœud d'objet apparaît généralement comme un rectangle avec son nom à l'intérieur. La figure 2-6 montre un exemple de sa notation. [8]

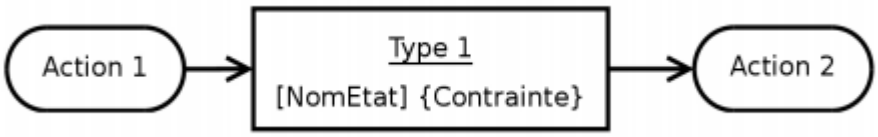

Figure2-6 notation de nœud objet

4- Notions générales de L'IDM

Système : Un système est un ensemble d'éléments liés entre eux par des relations dont si un élément soit modifié tout l'ensemble d'éléments sera modifié. Pour mieux connaître et étudier un système, on a besoin de le simplifier et le modéliser et avoir une abstraction du système qui est un modèle [9].

Modèle : Un « modèle » est une représentation simplifiée d'un système où il est construit avec l'intention de décrire le système et répondre aux questions que l'on se pose sur lui. Donc un système est modélisé avec un ensemble de modèles où chacun capture un aspect particulier. Par analogie avec les langages de programmation, un programme exécutable représente le système alors que le code source de ce programme représente le modèle [10].

Chaque modèle est exprimé à partir d'un langage de modélisation qui doit être clairement défini, et comme l'IDM prend la définition de tout est modèle donc le langage de modélisation prend la forme d'un modèle, appelé méta modèle [11]

Méta modèle : Un méta modèle est un langage d'expression (langage de modélisation) d'un modèle, ou un méta modèle représente une spécification formelle d'une abstraction. Le concept de méta modèle permet également d'aider, pour un système donné, à la construction d'un modèle. Par

conséquent, un modèle est une instance d'un méta modèle et un modèle est lié à son méta modèle par une relation de conformité.

Un méta modèle est exprimé à son tour avec un langage de méta modélisation spécifié par un méta méta modèle.

Méta-méta modèle : Comme l'IDM prend la définition de "tout est modèle", un méta-méta- modèle permet de décrire un modèle de méta modèles. Il est un langage de description de méta modèles.

Il permet d'exprimer les règles de conformité qui lient les entités du niveau modèle à celles du niveau méta modèle.

Un méta-méta modèle est conçu avec la propriété de méta-circularité, c'est â dire la capacité de se décrire lui-même. ça veut dire que le langage utilisé au niveau du méta-méta-modèle doit être suffisamment puissant pour spécifier sa propre syntaxe abstraite et ce niveau d'abstraction demeure largement suffisant. Le méta-méta modèle MOF est un exemple proposé par l'OMG [12].

Méta modélisation :

La méta modélisation est une activité qui consiste à définir le méta modèle d'un langage de modélisation. Il s'agit non seulement de produire des méta modèles mais aussi de définir la sémantique du langage, de mettre en œuvre des analyseurs, des compilateurs, des générateurs de code et plus généralement, à construire un ensemble d'outils exploitant les méta modèles.

5 Transformation des modèles :

La transformation de modèles est au cœur de l'IDM et y joue un rôle important. Elle peut être définie comme une génération d'un modèle cible à partir d'un modèle source, suivant une définition de transformation qui est un ensemble de règles de transformation qui décrivent comment un modèle dans un langage source peut être transformé en un modèle dans un langage cible. Une règle de transformation est une description de la manière d'une ou de plusieurs structures du langage source peuvent être transformées à une ou à plusieurs structures du langage cible.

Pour que la transformation soit appliquée à maintes reprises, elle doit être appliquée indépendamment du modèle source [13].

Les concepts de base de transformation sont présentés dans la figure 3 .

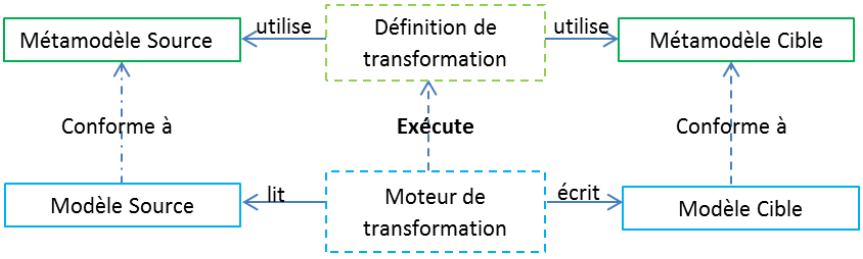

Figure 3 : Concepts de base de transformation de modèles

5 -1 Typologie de transformation :

Une transformation de modèle génère un modèle cible à partir d'un modèle source. ces deux modèles source et cible sont conformément aux même méta modèle ou à deux méta modèles différents aussi les modèles sources et cibles peuvent appartenir au même niveau d'abstraction (raffinement) ou à deux niveaux d'abstractions différents

5-1-1 Endogène et Exogène :

Une transformation de modèles endogène lorsque les modèles source et cible sont conforme au même méta modèle. Donc les règles de transformation sont présentées sur un seul méta modèle. On utilise les transformations endogènes dans le cadre d'optimisation de modèle ou d'une simplification d'un modèle ou dans un refactoring.

Une transformation de modèles exogène lorsque le méta modèle du modèle source est différent du méta modèle du modèle cible. Donc le formalisme qui va exprimer le modèle source est différent du formalisme qui va exprimer le modèle cible. On utilise les transformations exogènes dans le cadre de migration de modèle d'une plateforme à une autre en restant au même niveau d'abstraction, ou pour la génération de code et aussi les opérations de rétro-ingénierie ou rétro-conception.

5-1-2 Verticalité et horizontalité :

Une transformation de modèle peut aussi être classé selon un autre critère qui est le niveau d'abstraction si les modèles cible et source appartiennent au même niveau d'abstraction donc on a une transformation horizontale et si les modèles cible et source appartiennent à deux niveaux d'abstractions différents donc on a une transformation verticale. Par exemple,une génération de code est une transformation verticale.

Transformation	Horizontale	Verticale
Endogène	**Simplification Restructuration Refactoring**	**Raffinement**
Exogène	**Migration de langage**	**Génération de code Rétro-ingénierie**

Tableau 1 : Synthèse des transformations de modèles

5-2 Classification des approches de transformation :

Les approches de transformation de modèle ont été classées selon plusieurs axes. Chaque axe mène à une classification particulière. La classification des approches de transformation proposée par Czarnecki et Helsen [14] se base sur les techniques de transformation utilisées dans les approches et les facettes qui les caractérisent. Selon cette classification on peut distinguer deux types de transformation de modèles : les transformations de type modèle vers code et les transformations de type modèle vers modèles. Généralement, le premier type de transformation peut être vu comme un cas particulier du Propriétés de transformation de modèles Les principales propriétés qui caractérisent les transformations de modèles sont .

•**Traçabilité :** La traçabilité est la propriété d'avoir un dossier de liens entre les éléments de modèle source et ceux du modèle cible ainsi que les différentes étapes du processus de transformation. Les liens de traçabilité peuvent être stockés soit dans le modèle source, soit dans le modèle cible ou dans un modèle à part.

•**Directivité ou réversibilité** : une transformation est dite réversible si elle peut se faire dans les deux sens. Exemple : Model to text et text to Model. On dit en outre qu'une transformation est réversible s'il existe une transformation permettant de retrouver le modèle source à partir du modèle cible.
•**Réutilisabilité :** la Réutilisabilité peut être mesurée avec la possibilité d'adapter et de réutiliser les règles d'une transformation de modèle dans d'autres transformations. L'identification de patrons de transformation est un moyen pour mettre en œuvre cette réutilisabilité.
•**Ordonnancement** : l'ordonnancement consiste à représenter la suite des règles à exécuter lors d'une transformation. En effet, les règles de transformation peuvent déclencher d'autres règles.
•**Modularité** : une transformation modulaire permet de regrouper les règles de transformation en faisant un découpage du problème. Un langage de transformation de modèles supportant la modularité facilite la réutilisation des règles de transformation

6- Transformation de graphes :

Dans cette partie, nous allons présenter quelques notions sur les graphes , ensuite nous allons décrire les transformations et les grammaires de graphes.

6.1 Graphes et digraphes :

– **Définition1** : (**Graphes**) Un graphe est un ensemble de sommets ou nœuds (vertices) connectés par des liens appelés arêtes (edges). Plus formellement, un graphe G est le couple G = (V, E) où V est un ensemble de sommets et E est un sous ensemble de V x V représentant les arêtes, les graphes ainsi définis sont dits 'graphes non orientés'.
– **Définition2**. (**Digraphes**) On appelle graphe orienté ou digraphe un graphe pour lequel les arrêtes sont orientées (arcs). Un arc a = (x, y) peut être noté simplement xy, on dit que x est l'origine ou l'extrémité initiale et y est l'extrémité terminale de a. [15]

6 -2 Grammaire de graphes

Une grammaire de graphes distingue les graphes non terminaux, qui sont les résultats intermédiaires sur lesquels les règles sont appliquées, des graphes terminaux dont on ne peut plus appliquer de règles. Ces derniers sont dans le langage engendré par la grammaire de graphe, pour vérifier si un graphe G est dans les langages engendrés par une grammaire de graphe, il doit être analysé. Le processus d'analyse va déterminer une séquence de règles dérivant G [15].

6-2-1 Définition formelle :

Une grammaire de graphes est un quadruplet R = (T,N, V, P) où
• T est un alphabet d'étiquettes terminales,
• N est un alphabet d'étiquettes non-terminales ou variables,
• V est un ensemble quelconque de sommets,
• P est un sous-ensemble fini de règles (x,H) ou le membre gauche $x \in NV*$ est un hyper arc non-terminal et $H \subseteq NV *\rightarrow TVV$ est un hypergraphe fini d'hyper arcs non-terminaux et d'arcs terminaux.

6-2-2 Une règle est définie par une paire de graphes :

•**LHS** (Left Hand Side) est un graphe de côté gauche présentant la pré-condition de la règle et doit être un sous graphe de G.
•**RHS** (Right Hand Side) est un graphe de côté droit présentant la post- condition de la règle et doit remplacer LHS dans G.
Une grammaire de graphes est déterministe si deux membres gauches ont des étiquettes différentes. [15]

7- Triple Graphe Grammars (TGG) :

Triple Graph Grammars (TGGs) définissent la relation entre deux modèles.

Les TGGs ont été présentés par Andy Schürr en 1994 comme technique pour la transformation de modèle [16] où les modèles sont représentés comme graphes. Leur avantage principal est que TGGs permettent de définir une transformation bidirectionnel permettent aussi de définir la relation entre deux types de modèles , et calculer la correspondance entre deux modèles existants, ou pour maintenir l'uniformité entre les deux types de modèles qu'ils définissent par le TGGs.

Quand un des modèles est changé, les autres peuvent être changés en conséquence, ce qui signifie que les transformations ou les synchronisations peuvent être appliquées incrémentalement [16].

7-1 Fondements théoriques de TGG :

A fin de mieux comprendre TGG, nous explicitons à partir de [17] , les notions de base pour TGG.

•**Un Graphe** est défini comme G=(V,E,s,t) ou V ensemble de sommets (nœuds) et E ensemble des Arêtes et s,t: E→V sont des fonctions qui attribuent les sommets source et cible aux arêtes.

•**Un morphisme de graphe de G à G'** est une paire h =(hV , hE), ou hV = V→V', hE= E→E' sont définis de telle sorte qu'ils "préservent" les nœuds sources et cibles.

•**Une Grammaire de graphe** est une paire G=(G0, P), ou G0 ∈ G est l'axiome et P ={p: (L ↣ ^{l} K ↣ ^{r} R)} est un ensemble de règles de transformation.

•**Une production de graphe monotone** est une paire de graphes p= (L,R), ou L ⊂ R.

•**Une production de graphe** p est applicable à un graphe G s'il existe un morphisme h: L→G.

Un triple de graphes G= (SG $\xrightarrow{hSG}$ CG $\xrightarrow{hT\,G}$ TG ou SG graphe source, CG graphe de correspondance et TG graphe cible, h_{SG} est un morphisme de graphe entre les graphes SG ET CG et h_{TG} est morphisme de graphe entre les graphes CG et TG.

Un triple de productions est une structure p=(sp→^{hsp} cp→^{htp} tp), ou sp production source, cp production de correspondance, tp production cible, h_{sp} est un morphisme de graphe entre production source et de correspondance et h_{tp} est un morphisme de graphe entre productions de correspondance et cible. [15]

7.2 Règles de Triple Graph Grammar (TGGs) :

Une règle de transformation triple graphe de TGG= <pleft, pright, pmap> se compose de trois règles de transformation de graphe pleft, pright et pmap où pleft transforme le modèle de source, pright transforme le modèle de cible, et pmap transforme un modèle de relation de cette source de cartes aux éléments de cible. Chacun des trois productions de graphe est appliqué simultanément Le graphe évolue graduellement pendant la transformation en utilisant des règles de Triple Graph Grammar. Le graphe entier se conforme à son schéma à tout moment, aussi bien que les sous-graphes impliqués du côté gauche et droit. Le graphe de correspondance maintient les correspondances entre la source et les nœuds de graphe de cible.

Les Triple graph grammars soutiennent la transformation modèle bidirectionnelle et par accroissement en raison d'impliquer l'architecture sous-graphe de correspondance [18].

7-3- Spécification des transformations :

Modelware est l'espace technique des modèles, des méta modèles et leurs transformations dans l'IDM.

En revanche, Grammarware est l'espace technique des grammaires. Le pont technologique sert à trouver une communication ou une connexion entre l'espace technique des modèles et l'espace

technique des gram- maires, comme par exemple la traduction des éléments de Modelware en éléments de Grammarware.

Il existe une simultanéité entre une grammaire et un méta- modèle. La définition linguistique d'un méta modèle a une relation avec le concept d'une grammaire où un modèle conforme à un méta modèle est comme une chaîne reconnue par une grammaire. En outre, les propriétés syntaxiques et sémantiques en relation avec la conformité d'un modèle à un méta modèle, peuvent être prises en considération dans la sémantique d'une grammaire.

Notre pont technologique de la figure sert à lier l'espace technique de l'IDM et l'espace technique des grammaires où la transformation d'un méta modèle en une grammaire se fait d'une façon automatique. Les deux méta modèles source et cible sont transformé automatiquement en grammaires de graphes qui permettent de créer d'une manière incrémentale les règles de transformation et aussi la gram- maire de correspondance. A partir des correspondances sera généré un moteur transformation qui lit en entrée un modèle source et produit en sortie un modèle cible.

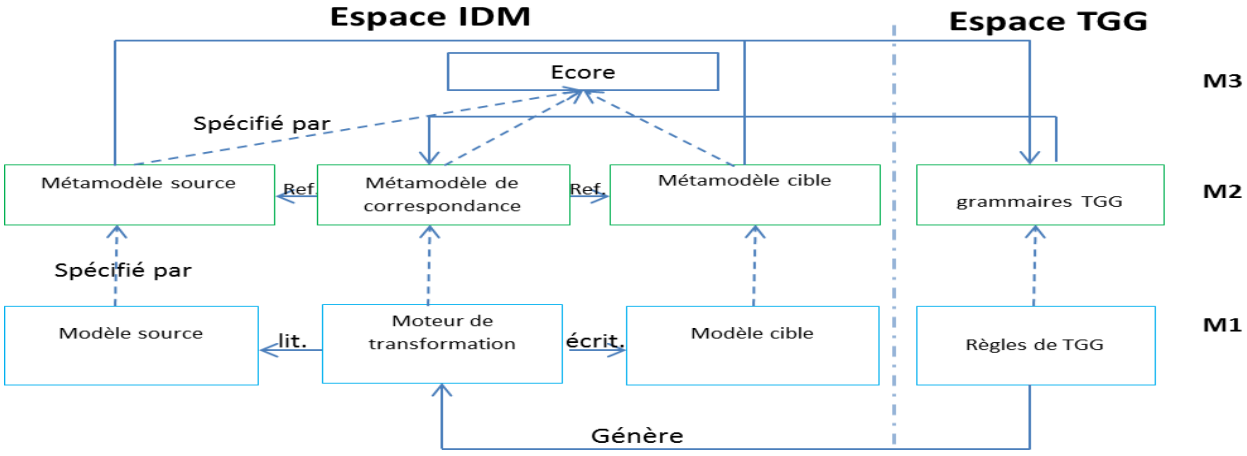

Figure 4: Transformation dans les espaces techniques IDM et TGG[15].

7-4- Outils de transformation :

Les outils de transformation présentés dans cette section sont orientés vers la transformation de graphes basée sur les grammaires de graphes. Il est important de noter que les outils choisis pour l'étude comparative sont des outils "open- source".

AGG :

AGG " Algebraic Graph Grammar" est un environnement à usage général pour le développement des systèmes de transformation de graphes attribués. Il est basé sur l'approche algébrique pour la transformation de graphes. Il vise à la spécification et le prototypage rapide d'applications complexes tel que le graphe de données structurées. Puisque la transformation de graphe peut être appliquée à des niveaux d'abstraction très différents, elle peut être attribuée ou non par des calculs simples ou par des processus complexes, selon le niveau d'abstraction. En raison de sa base formelle, AGG offre un support de validation, de vérification de la cohérence des graphes en fonction des contraintes graphiques [19].

ATOM3 :

ATOM3 "Domain Specific Visual Languages" [20] est un outil pour la conception de Domain

Specific Visual Languages (DSML). Il permet de définir la syntaxe abstraite et concrète d'un langage visuel au moyen de la métamérisation et d'exprimer la manipulation du modèle en utilisant la transformation de graphe. Avec les informations de méta modèle, ATOM3 génère un environnement de modélisation personnalisé pour le langage décrit.

Récemment, ATOM3 a été étendu avec des fonctionnalités pour générer des environnements avec des vues multiples de langages visuels (tels que UML) et Triple Graph Grammar (TGG) [5].

Ce dernier est utile pour exprimer l'évolution de deux modèles différents, liés par un modèle intermédiaire.

GROOVE :

GROOVE "GRaph of Object Oriented VErification" est un outil d'usage général basé sur la transformation de graphes pour modéliser la conception, la compila- tion et la structure d'exécution des systèmes orienté objet. La transformation de graphes avec GROOVE se base sur la transformation de modèles et la sémantique opérationnelle. Cela implique l'usage d'un fondement formel de transformation de modèles et de sémantique dynamique, et aussi la capacité de vérifier la transformation de modèle en utilisant le model checking. L'outil GROOVE a un éditeur pour la création des règles de production graphiques, un simulateur pour calculer visuellement les transformations de graphe induit par un ensemble de règles de production graphique. [21] [22].

GrGen :

GrGen est un outil de transformation de graphes utilisé dans différents domaines tels que la transformation de données graphiques complexes, la linguistique informatique, ou la construction de compilateur moderne. GrGen permet de travailler à un niveau très abstrait en utilisant le modèle déclaratif. Le code généré par GrGen s'exécute très rapidement. En termes de performance par rapport aux autres outils GrGen permet la manipulation et la transformation des systèmes complexes. Le système GrGen est écrit en Java et C.

Son noyau est un générateur de grammaires de graphe . [23].

Fujaba :

L'outil FUJABA (From UML to Java And Back Again) utilise les classes UML et les diagrammes pour produire du code Java, afin de fournir uns conception de systèmes formels et de langages spécifiques. Les fonctionnalités de la rétro ingénierie (reverse engineering) y compris la conception de reconnaissance de formes sont également disponibles.

Le méta modèle du FUJABA peut être étendu par héritage à la place de l'instanciation [24].

EMF Tigre :

EMF Tigre (Transformation génération) est un environnement pour la transformation de modèles EMF. Les transformations sont définies et exécutées directement sur des modèles EMF soit à l'aide d'un code généré ou soit avec un interpréteur afin d'assurer la sécurité et l'efficacité de types d'artéfacts.

Les règles basée sur Le langage de transformation EMF Tigre est inspiré par les concepts de transformation de graphes et combinent les deux aspects déclaratif et procédural.

En particulier, les règles de transformation sont déclaratives dans le sens où un modèle structurel est utilisé pour définir les pré-condition d'une règle. Le concept procédural est représenté par la

structure de contrôle de flux tels que les couches et les boucles peuvent être utilisés pour appliquer un ensemble de règles de transformation d'une manière contrôlée. [25]

Viatra :

Viatra est un outil basé sur Eclipse pour un usage général de l'ingénierie de transformation de modèles. C'est un Framework qui supporte un cycle de vie complet pour la spécification, la conception, l'exécution, la validation et la maintenance des transformations entre les différentes langages et les domaines de modélisation. VIATRA2 est apte à coopérer avec un système externe arbitraire et exécuter la transformation avec une transformation de modèle natif (plugin), qui est généré par VIATRA2. Le langage de spécification des règles combine la transformation de graphes et les machines d'état abstraites en un seul paradigme. Essentiellement, les étapes de transformation élémentaires sont capturés par des règles de transformation graphiques, tandis que des transformations complexes sont assemblés à partir des étapes élémentaires en utilisant les règles de machine d'états abstraites pour la spécification du flux de contrôle .

Exemple d'une règle TGG :

Le modèle A sur la figure 5 est le modèle de source ,et B est le modèle de cible. Au milieu est la partie de correspondance de la transformation. L'élément «A1» est l'élément de racine de Model A et les éléments « A2» et «A3» référence à lui. Du côté droit de la transformation l'élément «B1» est l'élément de racine. L'élément «A1» de Model A est tracé à l'élément «B1» de Model B par l'intermédiaire du MappingNode«Cor1

Le MappingNode«Cor1» contient l'information traçante des éléments «A1» et «B1» [26]

La figure 5 : montre un exemple d'une règle de TGG. Le graphe de source se conforme aux chaînes de processus entraînées par les événements, le graphe de cible

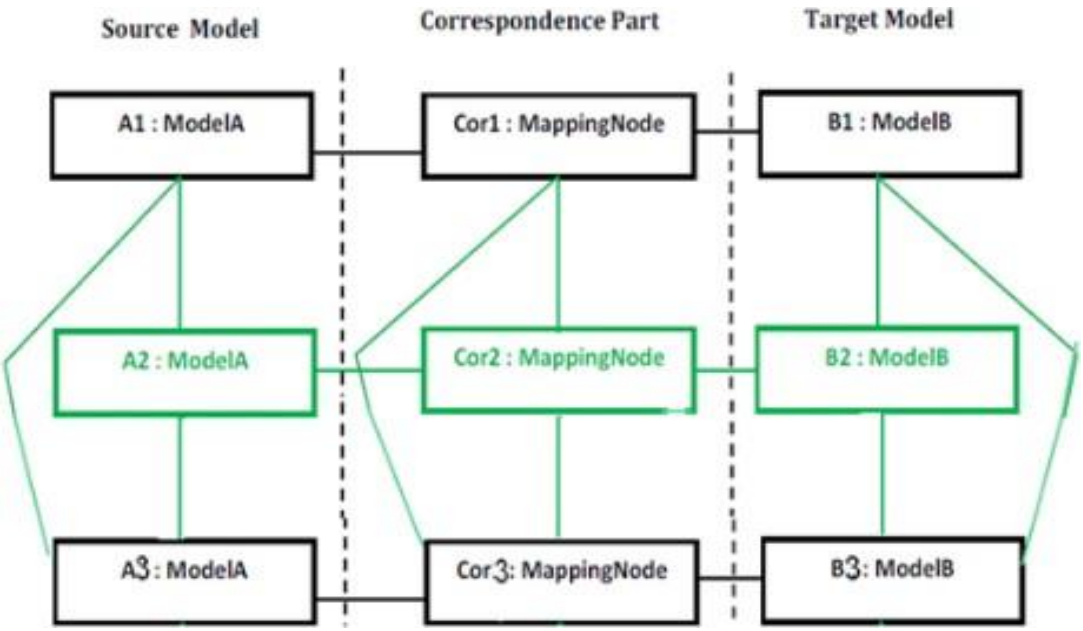

Figure 5 : Structure d'une règle possible de TGG. [15]

Tous les deux sont représentés comme graphe, avec les éléments de source du côté gauche et les éléments de cible du côté droit.

7-5 Création d'un nœud et désignation d'un arc :

On montre dans la figure 6, quand un nœud est créé avec l'outil s'accordant de la palette d'outil, un nom peut être écrit et le nœud de TGG doit être relié à un certain nœud de Domain Graph Pattern, ceci laisse alors choisir le type désiré de classe à partir du modèle particulier de classe de domaine.

D'autres arrangements du nœud peuvent alors être indiqués dans la vue de propriétés, par exemple si le nœud est un nœud de contexte/gauche ou si un objet d'exemple d'une sous-classe sera assorti ou pas.

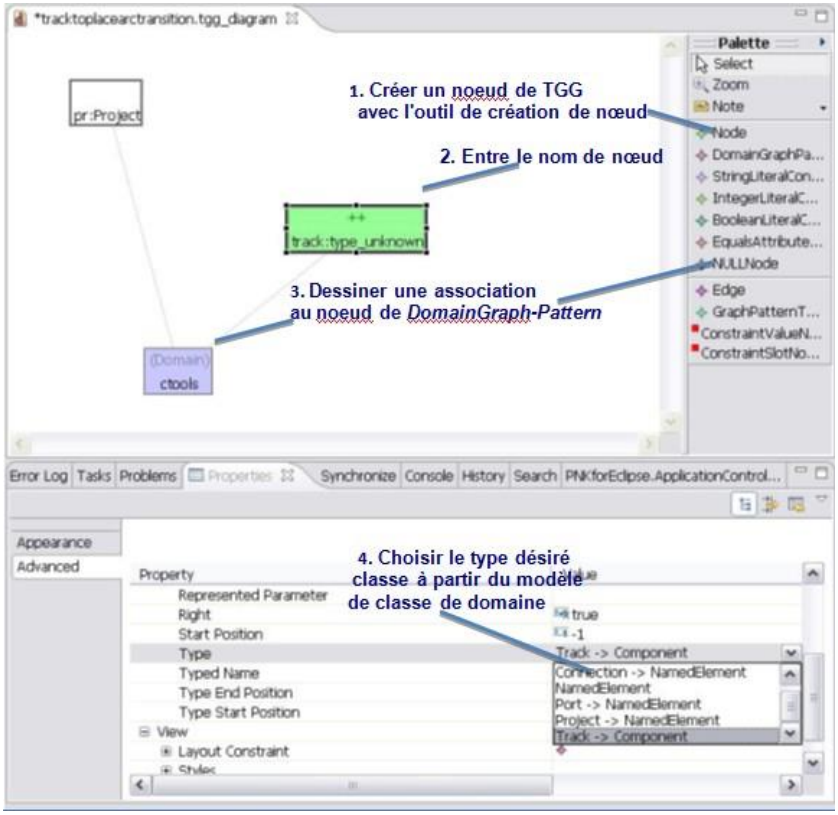

Figure 6 – Créer un nœud dans une règle de TGG.

De même, un arc peut être lié d'un nœud de TGG à l'autre comme il est représenté sur la figure 7. Premièrement, l'outil de création d'arc doit être choisi parmi la palette, puis un arc peut être lié d'un nœud de source à un nœud de cible. L'arc appartient automatiquement au même Domain- Graph Pattern comme nœud de source. Le choix dépend du type classes des nœuds de source et de cible.

Typiquement, il y a seulement un ou pas plus de quelques références qui existent entre deux classes. Dans le modèle d'exemple de ctools qui a proposé par le développeur de TGG pour faciliter a l'utilisateur à comprendre la manipulation de TGG .

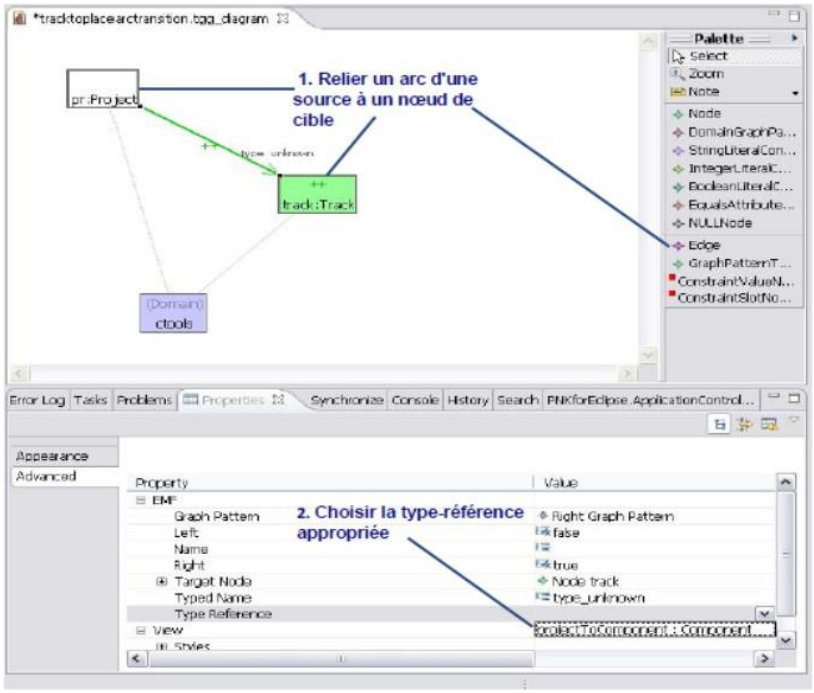

Figure 7 – graphes d'un arc entre les nœuds de TGG.

8- Object Constraint Language OCL :

Un méta-modèle permet de définir et de contraindre la structure des modèles qu'il décrit. Cependant, les langages de méta-modélisation ne permettent pas d'exprimer toutes les contraintes relatives à un méta-modèle. En pratique il est fréquent que des contraintes doivent être ajoutées à un méta modèle pour assurer la cohérence des modèles qu'il permet de définir. Les langages de méta-modélisation ne fournissent pas directement de solution à ce problème. Dans certains cas, les contraintes peuvent être simplement exprimées en langage naturel [27]. Lorsqu'il est nécessaire de vérifier automatiquement des contraintes, le langage de contraintes OCL (Object Constraint Language) est utilisé.

8.1 Description du langage :

Le langage OCL (Object Constraint Langage) est un langage déclaratif basé sur la théorie des ensembles et la logique des prédicats, il permet de décrire des règles et des contraintes qui peuvent être vu comme des restrictions sur le système. Il s'agit donc d'un langage formel qui fournit une expression textuelle pour les contraintes qui ne sont pas exprimable structurellement sur le méta modèle et donc définir des modèles EMF(Eclipse Modeling Framework) plus précis. Le langage OCL inclut un ensemble de constructions permettant de naviguer parmi les objets d'un modèle afin de vérifier des contraintes. Les constructions d'OCL ne permettent pas de créer, détruire ou modifier les objets d'un modèle : la vérification des contraintes se fait sans effet de bord **[**27**].**

8.2 Caractéristiques du langage :

OCL a des caractéristiques d'un langage d'expression, un langage de spécification et un langage formel[28] :

Langage d'expression : OCL est un langage d'expression pure. Une expression OCL est garantie sans effet de bord, elle ne peut rien changer dans le modèle c.à.d. que l'état du système ne sera jamais

changer à cause d'une expression OCL même si cette dernière est utilisée pour spécifier un tel changement d'état (dans le cas de post condition par exemple) en d'autre terme, l'évaluation de cette expression OCL livre juste une valeur, les instances ne sont pas modifiées par les contraintes.

Langage de spécification : OCL n'est pas un langage de programmation, il n'est pas possible d'écrire un programme logique avec OCL. On ne peut pas invoquer des processus ou bien des opérations autres que des requêtes dans OCL et on ne décrit pas le comportement à adopter si une contrainte n'est pas respectée.

Langage formel : OCL est un langage formel où tous les constructeurs ont un sens définit formellement.

Langage typé : c'est-à-dire que chaque expression dans OCL a un type et doit être d'accord avec les règles de conformité du langage. Les types en OCL peuvent être les types primitifs (comme string, integer, float, booléen etc.), ou types (classes) définis pour le système dans lequel l'expression est insérée.

8.3 Utilité du langage :

Le modèle graphique n'est pas suffisent pour une spécification précise et non ambigüe, on a besoin de décrire des contraintes sur les objets dans le modèle. Telles contraintes sont souvent décrites en langage naturel, mais le pratique a montré que ça donne toujours une ambigüité et pour éviter cette dernière, les langages formels sont développés[28]. Les langages formels traditionnels requièrent de la part des utilisateurs une bonne compréhension des fondements mathématiques. OCL se présente comme un langage formel et facile à lire et à écrire c.à.d. qu'il a été développé dans le but d'être : formel, précis et non ambigu.

8.4 Syntaxe des contraintes OCL

Le langage OCL a une spécification simple et facile à apprendre , elle se base sur [29] :

Contexte : OCL est toujours définie dans un « contexte » qui représente une instance d'une classe auquel est associée la contrainte.

Un invariant : représente une contrainte sur un objet (ou un groupe d'objet) qui doit être vérifiée à tout moment et on utilise pour cela le mot clé " inv"

Pré condition : Un prédicat qui doit être vérifié avant l'exécution de l'opération et permet de contraindre l'ensemble de valeurs d'entrée d'une opération.

Post condition : Un prédicat qui est vrai après l'exécution et permet de spécifier la sémantique d'une opération (ce qu'elle fait et non comment elle le fait).

Conclusion

Dans ce chapitre nous avons parlés d'UML, et de ses différents diagrammes nous avons description détaillée de l'un des diagrammes fondamentaux d'UML, qui est le diagramme d'activité. et aussi , nous avons présenté les notions de base de l'ingénierie dirigée par les modèles où nous avons mis l'accent sur la transformation de modèles et description détaillé l outil TGG.

Chapitre II : Les tests unitaires ou "tests de module

1-Introduction

Les tests unitaires, qui sont réalisés pendant la phase de développement, sont dissociables, selon le processus de sélection des tests utilisés, en deux stratégies :les tests fonctionnels (aussi appelés tests de boîtes noires), qui examinent les contraintes liées aux spécifications et évaluent les réactions du logiciel à certaines entrées, sans rentrer à l'intérieur des modules : l'aboutissement est un ensemble d'effets testés.

les tests structurels (aussi appelés tests de boîtes blanches), qui examinent la structure de chaque module, via le graphe de flot de contrôle; le code du programme est testé "petit bout par petit bout" (on parle de portions) et on nommera, à la fin de cette série de tests, couverture l'ensemble des portions de code qui auront été testées.

2- Génération boîte noire :

La génération des tests en boîte noire se base sur les spécifications fonctionnelles d'un programme (ou plus généralement, ses exigences), et impose a minima de pouvoir identifier le domaine des entrées du programme sous test ainsi que les oracles. Cette technique de génération ne présuppose en revanche aucune connaissance de la structure interne du programme, par exemple parce qu'on n'en dispose pas encore, ou qu'on ne cherche pas à l'exploiter. Il s'agit d'une technique applicable à tous les niveaux du cycle en V, et qui permet d'exploiter des exigences spécifiées informellement.

2-1. Procédures de test :

Une connaissance spécifique du code de l'application, de sa structure interne et de la connaissance de la programmation en général n'est pas nécessaire. [30] Le testeur est conscient de ce que le logiciel est censé faire mais ne sait pas comment il le fait. Par exemple, le testeur est conscient qu'une entrée particulière renvoie une certaine sortie invariable, mais ne sait pas comment le logiciel produit la sortie en premier lieu. [31]

2-2. Cas de test :

Les cas de test sont construits autour de spécifications et d'exigences, c'est-à-dire ce que l'application est supposée faire. Les cas de test sont généralement dérivés de descriptions externes du logiciel, notamment des spécifications, des exigences et des paramètres de conception. Bien que les tests utilisés soient essentiellement de nature fonctionnelle, des tests non fonctionnels peuvent également être utilisés. Le concepteur de test sélectionne les entrées valides et non valides et détermine la sortie correcte, souvent à l'aide d'un oracle de test ou d'un résultat précédent connu pour être bon, sans aucune connaissance de la structure interne de l'objet de test.

2-3 Comment faire des tests boite noire :

Voici les étapes génériques suivies pour effectuer tout type de test de boîte noire.

- Dans un premier temps, les exigences et spécifications du système sont examinées.
- Le testeur choisit des entrées valides (scénario de test positif) pour vérifier si SUT les traite correctement. De plus, certaines entrées non valides (scénario de test négatif) sont modifiées pour vérifier que le SUT est capable de les détecter.
- Le testeur détermine les sorties attendues pour toutes ces entrées.
- Le testeur de logiciel construit des cas de test avec les entrées sélectionnées.
- Les cas de test sont exécutés.
- Le testeur logiciel compare les sorties réelles avec les sorties attendues.
- Les défauts éventuels sont corrigés et testés à nouveau.

2-4. Types de test de la boîte noire :

Il existe de nombreux types de tests de la boîte noire, mais les suivants sont prédominants.

2-4-1. Test fonctionnel :

Ce type de test de boîte noire est lié aux exigences fonctionnelles d'un système; c'est fait par les testeurs de logiciels.

2-4-2. Test non fonctionnel :

Ce type de test de boîte noire n'est pas lié au test de fonctionnalités spécifiques, mais à des exigences non fonctionnelles telles que la performance, l'évolutivité et la convivialité.

2-4-3. Test de régression :

Le test de régression est effectué après que des correctifs de code, des mises à niveau ou toute autre maintenance du système ont été effectués pour vérifier que le nouveau code n'a pas affecté le code existant.

2-5. Les Outils utilisés pour le test de la boîte noire :

Les outils utilisés pour le test de la boîte noire dépendent fortement du type de test que vous effectuez.

- Pour les tests fonctionnels / de régression, vous pouvez utiliser - QTP, sélénium.
- Pour les tests non fonctionnels, vous pouvez utiliser - LoadRunner, Jmeter.

2-6. Techniques de test de la boîte noire :

Voici la stratégie de test en avant parmi les nombreuses stratégies utilisées dans le test de la boîte noire.

2-6-1. Test de classe d'équivalence :

Il est utilisé pour minimiser le nombre de tests possibles à un niveau optimal tout en maintenant une couverture de test raisonnable.

2-6-2 .Test des valeurs limites :

Le test des valeurs limites est axé sur les valeurs aux limites. Cette technique détermine si un certain intervalle de valeurs est acceptable ou non par le système. C'est très utile pour réduire le nombre de cas de test. Il convient particulièrement aux systèmes où une entrée se situe dans certaines plages.

2-6-3. Test de table de décision :

Une table de décision place les causes et leurs effets dans une matrice. Il y a une combinaison unique dans chaque colonne.

2-7. Les avantages de méthode boite noire :

Simplicité :

Ces tests sont simples à réaliser, car on se concentre sur les entrées et les résultats. Le testeur n'a pas besoin d'apprendre à connaître le fonctionnement interne du système ou son code source, qui n'est pas accessible. Cette méthode est donc également non intrusive.

Rapidité :

En raison du peu de connaissances nécessaires sur le système, le temps de préparation des tests est très court. Les scénarios sont relativement rapides à créer et à tester, puisqu'ils suivent les chemins utilisateurs, qui sont relativement peu nombreux selon la taille du système.

Impartialité :

On est ici dans une optique « utilisateur » et non « développeur ». Les résultats du test sont impartiaux : le système marche, ou il ne marche pas. Il n'y a pas de contestation possible, comme par exemple sur l'utilisation de tel processus plutôt qu'un autre selon l'opinion du développeur.

2-8. Les inconvénients boîte noire :

Superficialité :

Etant donné que le code n'est pas étudié, ces tests ne permettent pas de voir, en cas de problème, quelles parties précises du code sont en cause. De plus, les testeurs peuvent passer à côté de problèmes ou vulnérabilités sous-jacentes. Certains problèmes sont également difficilement repérables avec cette méthode, comme par exemple ceux liés à la cryptographie, ou à des aléas de mauvaise qualité. C'est donc l'un des tests les moins exhaustifs.

Redondance :

Si d'autres tests sont effectués, il est possible que celui-ci perde grandement de son intérêt, puisque son champ d'action a tendance à être inclus dans celui d'autres tests. [32]

3- Les tests en « boîte blanche » :

Les tests en « boîte blanche » consistent à examiner le fonctionnement d'une application et sa structure interne, ses processus, plutôt que ses fonctionnalités. Sont ici testés l'ensemble des composants internes du logiciel ou de l'application, par l'intermédiaire du code source, principale base de travail du testeur.

Pour réaliser un test en « boîte blanche », ce dernier doit donc avoir des compétences de programmation, afin de comprendre le code qu'il étudie. Il doit également avoir une vue globale du fonctionnement de l'application, des éléments qui la composent, et naturellement de son code source. Contrairement aux tests en « boîte noire », le testeur ici a un profil développeur, et non pas utilisateur.

En effectuant un test en « boîte blanche », on voit en effet quelle ligne de code est appelée pour chaque fonctionnalité. Cela permet de tester le flux de données ainsi que la gestion des exceptions et des erreurs. On s'intéresse également à la dépendance des ressources, ainsi qu'à la logique interne et justesse du code. C'est pourquoi ces tests sont surtout utiles pendant le développement d'une application, même s'ils peuvent être effectués durant de nombreuses phases de la vie d'un projet. La méthode en « boîte blanche » peut être appliquée pour les tests unitaires (majoritairement), les tests d'intégration et les tests système.

La méthode en « boîte blanche » utilise des scénarios de test, créés par le testeur selon ce qu'il a appris du code source de l'environnement. L'objectif est qu'en testant l'ensemble de ces scénarios, toutes les lignes de code soient vérifiées. Ce qui est regardé, c'est le processus effectué par l'application après une entrée (input) pour obtenir un résultat. On ne fait que vérifier si le code produit les résultats espérés.

3-1. Types de test de la boîte blanche :

Les tests de boîte blanche englobent plusieurs types de tests utilisés pour évaluer la facilité d'utilisation d'une application, d'un bloc de code ou d'un progiciel spécifique. Il y a énumérés ci-dessous –

3-1-1 Tests unitaires :

Il s'agit souvent du premier type de test effectué sur une application. Les tests unitaires sont effectués sur chaque unité ou bloc de code au fur et à mesure de son développement. Les tests unitaires sont essentiellement effectués par le programmeur. En tant que développeur de logiciels, vous développez quelques lignes de code, une seule fonction ou un objet, et vous le testez pour vous assurer qu'il fonctionne avant de poursuivre les tests unitaires, ce qui permet d'identifier la majorité des bogues, au début du cycle de développement du logiciel. Les bogues identifiés à cette étape sont moins chers et faciles à corriger.

3-1-2. Test des fuites de mémoire :

Les fuites de mémoire sont les principales causes de ralentissement des applications. Un spécialiste en assurance qualité expérimenté dans la détection des fuites de mémoire est essentiel dans les cas d'application lente d'un logiciel.

En dehors de ce qui précède, quelques types de tests font partie des tests des boîtes noires et des boîtes blanches. Ils sont énumérés ci-dessous

3-1-3. Test d'intrusion dans la boîte blanche :

Lors de ces tests, le testeur / développeur dispose des informations complètes sur le code source de l'application, des informations détaillées sur le réseau, les adresses IP concernées et toutes les informations de serveur sur lesquelles l'application est exécutée. L'objectif est d'attaquer le code sous plusieurs angles pour exposer les menaces à la sécurité.

3-1-4. Test de mutation par boîte blanche :

Le test de mutation est souvent utilisé pour découvrir les meilleures techniques de codage à utiliser pour développer une solution logicielle.

3-2 Outils de test de boîte blanche

Vous trouverez ci-dessous une liste des meilleurs outils de test des boîtes blanches. Veracode , EclEmma , RCUNIT , NUnit , JSUnit , JUnit , CppUnit

3-3. Techniques de test de la boîte blanche

Une technique majeure de test de la boîte blanche est l'analyse de couverture de code. L'analyse de couverture de code élimine les lacunes dans une suite de cas de test. Il identifie les zones d'un programme qui ne sont pas exercées par un ensemble de cas de test. Une fois les lacunes identifiées, vous créez des scénarios de test pour vérifier les parties non vérifiées du code, ce qui améliore la qualité du logiciel.

Il existe des outils automatisés pour analyser la couverture du code. Voici quelques techniques d'analyse de couverture

Couverture des instructions: - Cette technique nécessite de tester toutes les instructions possibles dans le code au moins une fois pendant le processus de test du génie logiciel.

Couverture de branche - Cette technique vérifie tous les chemins possibles (si-sinon et autres boucles conditionnelles) d'une application logicielle.

Outre ce qui précède, il existe de nombreux types de couverture tels que la couverture de condition, la couverture de condition multiple, la couverture de chemin, la couverture de fonction, etc. Chaque technique a ses propres mérites et tente de tester (couvrir) toutes les parties du code logiciel. En utilisant la couverture des relevés et des succursales, vous obtenez généralement une couverture de code de 80 à 90%, ce qui est suffisant.

3-4. Avantages du test de la boîte blanche :

Anticipation : effectuer ces tests au cours du développement d'un programme permet de repérer des points bloquants qui pourraient se transformer en erreurs ou Icone Amélioration et travail d'équipe problèmes dans le futur (par exemple lors d'une montée en version, ou même lors de l'intégration du composant testé dans le système principal).

Optimisation : étant donné qu'il travaille sur le code, le testeur peut également profiter de son accès pour optimiser le code, pour apporter de meilleures performances au système étudié (sans parler de sécurité…).

Exhaustivité : étant donné que le testeur travaille sur le code, il est possible de vérifier intégralement ce dernier. C'est le type de test qui permet, s'il est bien fait, de tester l'ensemble du système, sans rien laisser passer. Il permet de repérer des bugs et vulnérabilités cachées intentionnellement (comme par exemple des portes dérobées). C'est notamment pour cela que l'ARJEL impose des audits de code en « boîte blanche » aux opérateurs de jeux en ligne, afin de s'assurer qu'aucun mécanisme caché ne vienne désavantager les joueurs.

3-5. Inconvénients des tests boite blanche :

Complexité : ces tests nécessitent des compétences en programmation, et une connaissance accrue du système étudié.

Durée : de par la longueur du code source étudié, ces tests peuvent être très longs.

Industrialisation : pour réaliser des tests en « boîte blanche », il est nécessaire de se munir d'outils tels que des analyseurs de code, des débogueurs… Cela peut avoir un impact négatif sur les performances du système, voire même impacter les résultats.
Cadrage : il peut être très compliqué de cadrer le projet. Le code source d'un programme est souvent très long, il peut donc être difficile de déterminer ce qui est testé, ce qui peut être mis de côté… En effet, il n'est pas toujours réaliste de tout tester, ce qui prendrait trop de temps. Il est également possible que le testeur ne se rende pas compte qu'une fonctionnalité prévue dans le programme n'y a pas été intégrée. Il n'est donc pas dans le scope du testeur de vérifier si tout est là : il ne fait que tester ce qui est effectivement présent dans le code.

Intrusion : cette méthode est très intrusive. Il peut en effet être risqué de laisser son code à la vue d'une personne externe à son entreprise : il y a des risques de casse, de vol, voire même d'intégration de portes dérobées… Choisissez donc toujours des testeurs professionnels .

4- Les tests d'interface :

Les tests d'interface homme machine revêtent une importance croissante avec l'explosion d'applications web. Le développement du e-business en fait des outils stratégiques [33]

5- JUnit :

Junit est un framework de test unitaire pour le langage de programmation Java. Créé par Kent Beck et Erich Gamma, JUnit est certainement le projet de la série des xUnit connaissant le plus de succès.

JUnit définit deux types de fichiers de tests. Les cas de tests (TestCase) sont des classes contenant un certain nombre de méthodes de tests. Un cas de test sert généralement à tester le bon fonctionnement d'une classe. Une Suite de test (TestSuite) permet d'exécuter un certain nombre de cas de test déjà définis.

Les avantage de junit

- ➢ Utilisation d'un debugger (possibilité de redéfinir les expressions de test) sans avoir à recompiler les programmes.
- ➢ Moins de dépendance chronophage avec le debugger.
- ➢ automatiser les tests Junit améliore la qualité du code.

conclusion

Dans ce chapitre nous avons présenter les testes unitaire et détaillé les testes fonctionnel « boite noire » et les testes structurels « boite blanche » et défini les avantage et les inconvénients des testes et en fin présenter le outils de teste juint.

Chapitre III : Mise en œuvre de la transformation de modèles

1- Introduction :

Le but de ce chapitre est de proposer une approche automatique de transformation des diagrammes d activité d'UML2.0 vers cas de test, basée sur la transformation des graphes et réalisée à l'aide de l'outil TGG.

Ce chapitre est organisé comme suit :

- Dans la première partie, on présente Eclipse Modeling Framework et TGG interpréter
- Dans la deuxième partie, on présente les méta modèles nécessaire pour créer les règles TGGs.
- Dans la troisième partie, on présente toutes les règles TGG pour faire la transformation.
- Et enfin on termine par une conclusion.

2- Eclipse :2

Eclipse est une plateforme universelle d'intégration d'outils de développement. Il s'agit en effet d'un IDE (Environnement de Développement Intégré) ouvert, facile- ment extensible et qui n'est pas lié spécifiquement à un langage de programmation. Cette plateforme fournit à la base, un ensemble de services pouvant être éventuelle- ment enrichis par le biais de plugins.

2-1 Eclipse Modeling Framework :

Eclipe Modeling Framework (EMF) : est un framework Java de modélisation et un outil de génération de code pour construire des applications basées sur des modèles. Depuis un modèle de spécifications décrit en XMI, EMF fournit des outils et un support de moteur d'exécution pour produire des classes Java. De plus EMF permet de stocker les modèles sous forme de plusieurs fichiers reliées. Les modèles peuvent être spécifiés en utilisant des documents Java, UML, XSD, XML, puis sont importés dans EMF. Par contre EMF ne propose pas d'éditeur graphique pour la modélisation. Le plus important est qu'EMF fournit les fondements à l'interopérabilité avec d'autres outils ou applications basés sur EMF.

EMF utilise le langage Ecore qui est un langage de méta modélisation graphique et textuelle défini par IBM.

2-2 Ecore :

Le langage Ecore [34] est un langage de méta-modélisation graphique et textuelle défini par IBM et utilisé dans le framework de modélisation d'Eclipse EMF. Il est utilisé pour définir les métamodèles et le langage OCL est intégré pour d'écrire les contraintes dans le but de vérifier la conformité des modèles à leurs méta modèles. Nous choisissons d'utiliser ce langage à base des critères suivants :

- Le langage Ecore [35] est un langage graphique et textuel. La représentation graphique du langage permet de manipuler directement les concepts, sans passer par une formalisation textuelle abstraite et difficile à acquérir.

- Ecore intègre le langage formel OCL qui facilite l'implémentation et permet de d'écrire les contraintes. Une contrainte est une expression que l'on peut attacher à n'importe quel élément du méta modèle.

2-3 TGG interpréter :

Le TGG Interpréter [36] a été développé pour la transformation de modèles TGG et est un outil de mise à jour incrémentale résultant de la comparaison de TGG et dela norme de l'OMG bidirectionnel QVT (Query/View/Transformation)[37], [38] pour les transformations de modèles.

Ergonomie: Le TGG interpréter est basé sur Eclipse et peut être installé via l'Update

Manager Eclipse.

3-Les travaux connexes :

Nous nous sommes sur notre travail sur un ensemble des articles Nous faisons une description superficielle de cet articles:

3-1. Première article :

3-1-1.Génération de cas de test à partir de diagrammes d'activité UML

Un test important dans le test de logiciel est le test génération de cas. C'est particulièrement difficile quand un Le système contient des participants exécutant simultanément (ou objets), puisqu'un tel système peut présenter différentes réponses en fonction de la condition de concurrence.

L'activité UML (Unified Modeling Language) diagramme est un langage de modélisation approprié pour décrire les interactions entre les objets système depuis l'activité diagramme peut être facilement utilisé pour capturer des affaires processus, flux de travail et scénarios d'interaction. Dans le passé, il y avait plusieurs approches pour générer des cas de test à partir de diagrammes d'activité UML. Mais la plupart d'entre eux ne traitent pas de la concurrence problèmes et ont tendance à ne proposer qu'une idée conceptuelle pour la génération de tests.

Nous proposons une méthode basée sur une E / S modèle explicite de diagramme d'activités (IOAD), qui est un modèle d'abstraction obtenu à partir de la pleine expansion diagramme d'activité en exposant uniquement les entrées externes et sorties. L'avantage d'utiliser cet intermédiaire modèle est que les modèles IOAD nous permettent de nous concentrer uniquement sur comportement observable de sorte que seul et tout le comportement pertinent pour les tests sont conservés dans le modèle. Puis de Modèle IOAD, notre méthode construit un graphe dirigé extraire des scénarios de test et des cas de test. Pour augmenter un efficacité du test du système, nous utilisons le test de tous les chemins critère de couverture.

En utilise la algorithme DFS pour générer les scenario de teste

3-1-2. Diagramme d'activité explicite d'E / S

Les diagrammes d'activité UML sont généralement utilisés pour représentations du flux de travail la réalisation du fonctionnement de la phase de conception et affinement ou ordre de séquence et concurrence. Contrairement à la Diagramme d'activités UML.

IOAD est un modèle qui supprime les entrées et les sorties non externes dans le langage UML diagramme d'activité. Les activités sont classées en deux éléments externes: envoi du signal et accepte de l'événement. À illustrer cette fonctionnalité, toutes les activités sont traduites en envoyer un signal et accepter les notations d'événement. En attendant les objets de données tels que la facture et la commande sont supprimés parce que ces objets sont des tâches implicites. Il y a trois caractéristiques dans ce modèle. Cette modèle modifie l'activité d'origine entièrement spécifiée diagramme dans un IOAD. Nous représentons le modèle par entrées externes et sorties externes. Dans une activité diagramme, l'action peut être divisée en trois catégories: action d'entrée, action de sortie et action interne. Interne l'action est moins importante pour les testeurs qu'un résultat de le comportement de la fonction est considéré comme un plus résultat substantiel Basé sur ce modèle, séquence, décision et boucle les cas sont tous des constructions simples. Partout où envoyer le signal et accepte event sont situés, séquentiel c, booléen cas (true / false), et les cas boucle-non boucle sont simplement repensé. En outre, la construction fork join devrait être considéré.

3-1-3. Couverture de test :

Lors de l'analyse des graphiques, nous choisissons le chemin de base et utilisé le premier algorithme de recherche. Ces deux principes ont été prouvés dans les tests de graphes à générer des scénarios de test appropriés. Lors du calcul d'un chemin d'un diagramme d'activité, si chaque activité du chemin produit qu'une seule fois, nous appelons un tel chemin un chemin de base de le diagramme d'activité, Le schéma composé par tous les chemins de base d'un diagramme d'activité est appelé base diagramme d'activité du diagramme d'activité , DFS algorithme est utilisé pour rechercher tous les nœuds, du premier nœud au nœud final, pour calculer tous les chemins d'un diagramme d'activité.

Quand on traverse un diagramme d'activité de l'état d'activité initiale à l'activité finale état par algorithme DFS, nous nous assurons que les boucles sont exécuté un à la fois et que tous les états d'action et les transitions sont couvertes. Nous arrivons ainsi à un certain nombre de chemins de base.

Ce nombre de chemins de base est généralement acceptable dans la pratique. Nous définissons donc ces chemins de base critères de couverture fondés sur les critères de réalisation du test. Une approche intuitive consiste à exiger une suite de tests pour couvrir tous les chemins possibles dans un graphe de flux. Mais ce n'est pas un approche pratique, car de nombreux graphiques de flux contiennent un nombre énorme ou même un nombre infini de chemins , Pour adopter un critère pratique de couverture de chemin, il faut sélectionner un sous-ensemble représentatif de tous les chemins pour effectuer un nouveau test et la sélection doit supprimer redondant information dans un chemin ,Il y a deux manières: la première est de supprimer les nœuds redondants, et la seconde est de supprimer les bords redondants.

En théorie des graphes, un chemin élémentaire est un chemin avec pas de répétition d'aucun nœud et un chemin simple est un chemin sans récurrence d'aucun bord , Avec ces restrictions, étant donné un graphe de flux, il y a généralement un nombre très limité de chemins élémentaires et simples. [39]

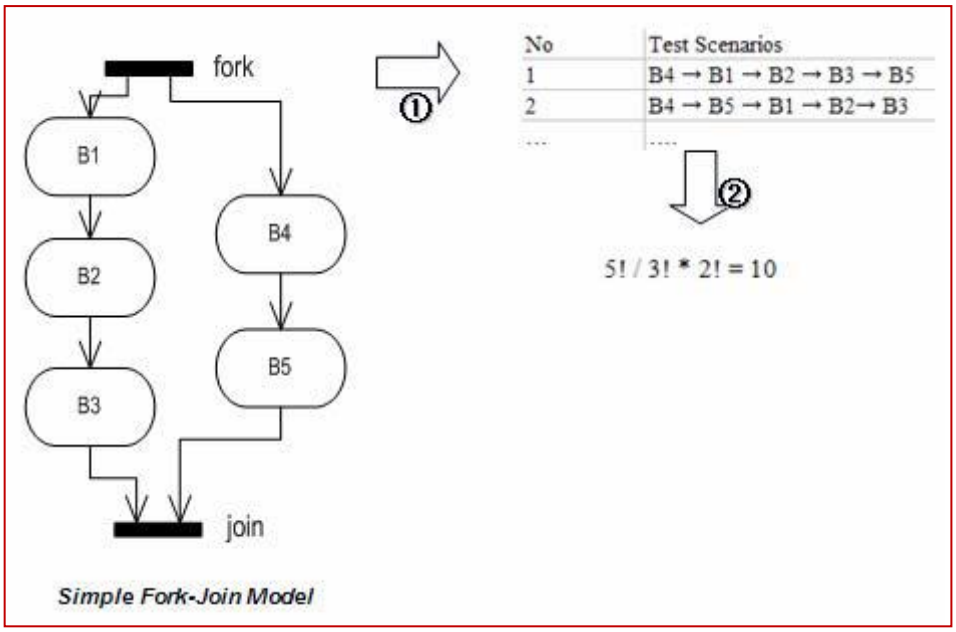

Figure 8: model fork-join

Exemple : IOAD

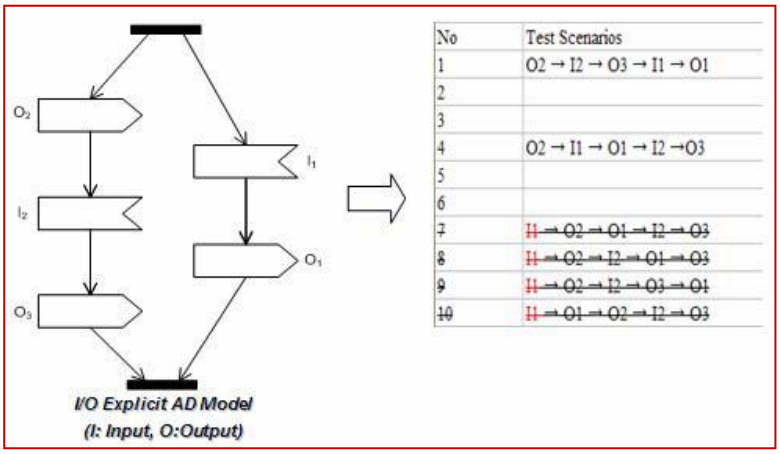

Figure 9 : exprimer les test scénario fork –join

3-2-Deuxième article

3-2-1 Génération de cas de test à partir du diagramme d'activité UML l'utilisateur Approche perspective :

Dans cet article, une nouvelle approche est présentée pour générer des cas de tests de niveau utilisateur à partir d'une activité UML Diagramme L'importance majeure de cette approche est donner à l'utilisateur final la possibilité de tester le logiciel indépendamment des développeurs de logiciels.

Le test de la perspective de l'utilisateur a pour objectif de s'assurer que le système prend totalement en charge la gestion quotidienne. scénarios d'affaires ils sont utilisés dans le génération de scénarios d'utilisation et de scénarios de test. Les diagrammes d'activités UML ont suscité une attention particulière dans génération de cas de tes.

Le test de logiciel est un processus de vérification et valider qu'une application logicielle ou un programme répond au métier et aux exigences qui ont guidé son conception et développement.

Cela nécessite l'utilisation d'outils et de mécanismes appropriés pour assurer un haute niveau de qualité du système en développement.

Il existe de type de teste trois dimensions. Une dimension montre l'échelle de System Under Test (SUT), qui va de la petite unité jusqu'à un système entier. La seconde dimension montre les différentes caractéristiques que nous peut vouloir tester. La troisième dimension montre le genre des informations que nous pouvons vouloir utiliser pendant le logiciel essais.

Dans cette article , nous nous sommes concentrés sur le test de la perspective de l'utilisateur

La perspective de l'utilisateur : Les tests visent à vérifier et à valider les systèmes mis en œuvre dans le monde réel par l'utilisateur final selon les besoins d'origine.

Test : d'acceptation utilisateur (UAT) peut être considéré comme exemple typique de test de perspective d'utilisateur.

UAT : est la phase de test utilisée pour déterminer si un système satisfait à la exigences spécifiées dans l'analyse des exigences phase

Le diagramme : d'activité UML a été utilisé plus fréquemment pour modéliser le processus métier de haut niveau car cela demande un effort et un temps minimaux pour se comprendre. Il représente également le flux d'affaires en forme plus abstraite que d'autres diagrammes UML.

3-2-2. La solution proposée :

Dans cette approche, les diagrammes d'activité UML sont utilisés générer des cas de tests au niveau utilisateur pour tester un logiciel système du point de vue de l'utilisateur final.

Le prototype : est un système basé sur fenêtre qui est développé à l'aide de Microsoft Visual Studio 2010 et C Langage de programmation Sharp. Visual Studio 2010 fournit un Framework. net 4 intégrant LINQ qui est un outil efficace pour analyser le fichier XMI. SQL La base de données Server 2008 est également utilisée dans mise en œuvre pour stocker le intermédiaire représentation à différentes étapes du système processus.

Ce prototype est conçu pour prendre en compte les informations Outils de modélisation UML au format XMI [40]

3-3. troisième article :

3-3-1. Génération de cas de test à partir de modèles UML comportementaux :

- utilisé une approche intégrée pour générer des cas de test à partir de UML diagrammes de séquence et d'activité
- Nous transformons ces UML diagrammes dans un graphique
- nous proposons un algorithme pour générer des tests scénarios du graphe construit

Cette article présente méthode pour générer des cas de test automatiquement à partir de Séquence et diagrammes d'activité UML. Nous convertissons les modèles en un représentation intermédiaire appelée Model Flow Graph (MFG) qui est une intégration de la représentation intermédiaire de la séquence et de l'activité des diagrammes.

Couverture de tous les chemins d'activité: donnée à l'ensemble de test T et au diagramme d'activité AD, T doit faire en sorte que chaque chemin d'activité possible dans AD soit pris au moins une fois

Tous les activités permettant d'exécuter la méthode peuvent être affichées à l'aide d'un diagramme d'activité.

Nous utiliser séquence et diagramme d'activité, nous pouvons couvrir le message.

Savoir Théorème 1: La couverture du chemin de message est une technique de test plus puissante par rapport à la couverture des messages.

Preuve : Un chemin de message dans un MFG est un chemin allant du nœud racine à un autre nœud dans le MFG, et la couverture du chemin de message est un technique de test plus forte comparée à la couverture de message.

3-3-2. représentation intermédiaire :

Pour générer des données de test, il est d'abord nécessaire de transformer le diagramme en une représentation intermédiaire fiable. La séquence UML et les diagrammes d'activité représentent les aspects comportementaux de la phase de conception.

3-3-3. notre approche pour générer un test cases :

Le processus de génération de test est divisé en trois phases principales. Le premier la phase consiste à générer le MFG à partir du diagramme de séquence et d'activité séparément. La deuxième phase consiste à générer des séquences de test à partir de MFG. Correspondant aux diagrammes de séquence et d'activité. Les séquences de test sont un ensemble de chemins théoriques allant de l'initialisation à la fin, tandis que prendre en compte les conditions (pré-condition et post-condition).

Chaque séquence de test générée correspond à un scénario particulier du cas d'utilisation considéré. La troisième phase consiste à générer un scénario de test à partir de la séquence générée satisfaisant l'adéquation du test de chemin d'activité de message critères. [41]

4- Les méta modèles :

Pour transformer les diagrammes d'activité en cas de teste, nous allons proposer 3 méta modèle.

4-1. Méta modèle de diagrammes d'activités UML

Model source : méta model diagramme d activité

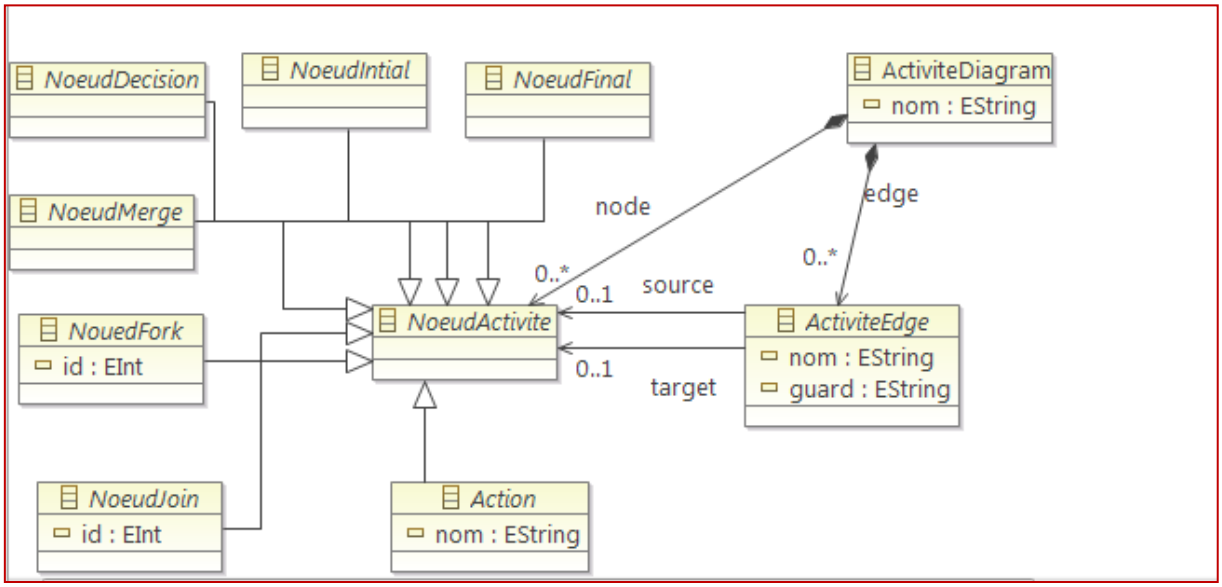

Figure 10 : met modèle diagramme d'activité

Présente le méta modèle d'un diagrammes d'activités UML. Il indique qu'un Diagram Activité se compose d'ActivityEdge et d'Activité. On distingue en outre sept types différents d'Activité: nœud Initial, Action, noeudFork, etc. Ces nœuds sont représentés comme les classes dérivées de la classe Activité.

les classes :

Classe nœud Initial : représente le début d'un diagramme d'activité. Graphiquement, elle est représentée par un petit cercle plein.

Classe noued Fork : elle représente un nœud de synchronisation qui possède un seul arc entrant et plusieurs arcs sortants qui doivent être déclenchés simultanément.

Classe nœud Merge : cette classe rassemble plusieurs flots entrants en un seul flot sortant.

Classe noued Join : elle représente un nœud de synchronisation qui ne peut être franchit que lorsque toute les transitions en entrée ont été déclenchées. Elle est reliée par plusieurs transitions en entrée et une seule transition en sortie.

Classe nœud Final : indique une terminaison avec succès. Elle possède un ou plusieurs arcs entrants et aucun arc sortant, elle est représentée visuellement par un cercle vide contenant un petit cercle plein.

Classe Action : est une étape de l'activité**,** cette classe représente une opération atomique non décomposable et non interruptible. Elle est représentée visuellement par un ovale qui contient sa description textuelle

Classe nœud Décision : cette classe spécifie les différentes alternatives possibles. Elle a un seul arc entrant et deux ou plusieurs arcs sortants.

4-2 cas de teste

Model cible : les cas de teste

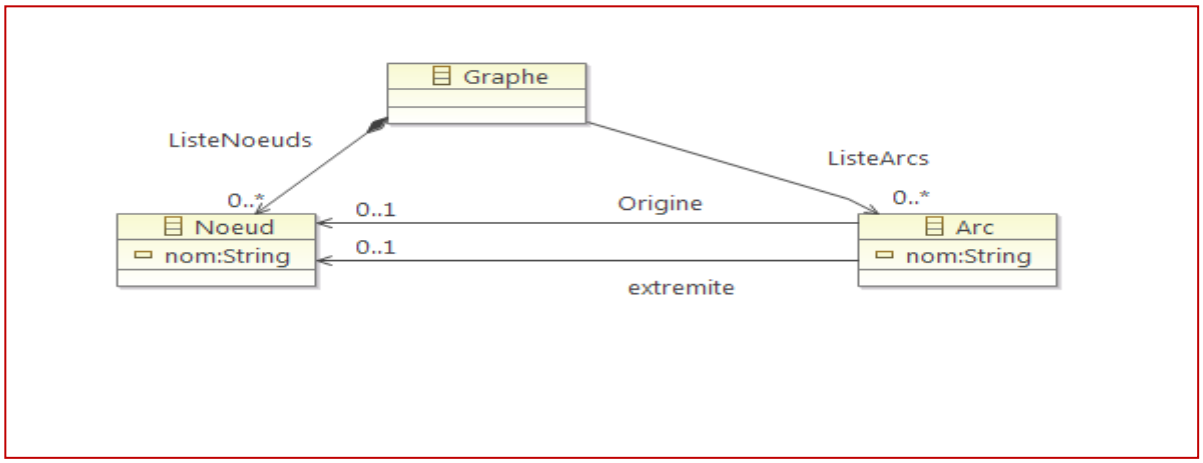

Figure 11: metamodel de cas de teste

4-3. Le méta-modèle de correspondance :

On définissant la relation entre les diagrammes d'activité et les cas de tetse , nous établissons la relation par les nœuds additionnels qui se rapportent à des éléments des deux diagrammes. Ces noeuds s'appellent les nœuds de correspondance

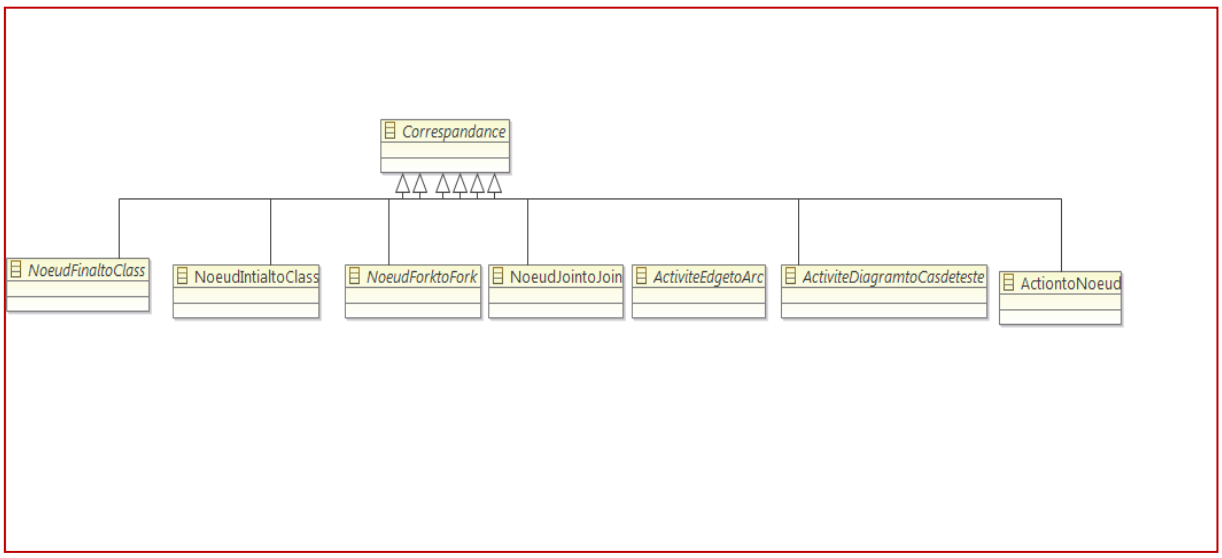

Figure 12 : met model de correspondance

5- La transformation des diagrammes d'activités vers les cas de teste :

On commence par transformer le modèle des diagrammes d'activité vers le modèle de cas de teste (modèle vers modèle) on utilisant l'outil TGG. Dans cette section, on définit la relation réelle entre les diagrammes d'activité et cas de teste , pour se faire, on proposant quelque règle pour la transformation :

5-1. Définir la relation "les règles TGG" :

Dans cette section, nous définissons la relation réelle entre les diagrammes d'activité et les cas de teste

5-1-1. L'axiome :

présente l'axiome de la transformation d'un diagramme d'activités en generation des cas de teste chaque diagramme d'activité transformé ver une graphe , Sur le côté gauche, il montre un objet racine d'un diagramme d'activités, sur le coté droit, il montre un objet racine d'un génération des cas de teste , et dans la partie médiane, il montre un nœud de correspondance relative des deux.

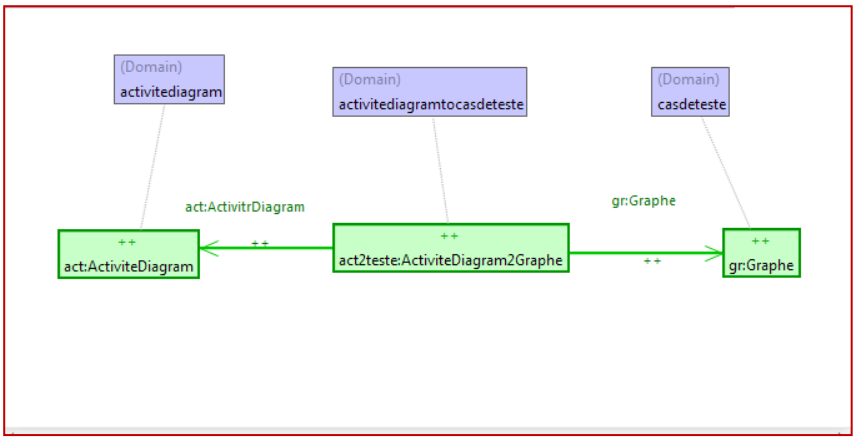

Figure 13 : L'Axiome

A partir de cet axiome, nous discutons maintenant les autres constructions qui se produisent dans le diagramme d'activités et montrons comment les états correspondants sont créés dans les cas de teste .

5-1-2. Transformation d'un "ActivityEdge" à un "arc" :

L'idée de base de cette règle de transformation est que chaque ACtivityEdge d'un diagramme d'activités se transforme à un arc dans le cas de teste

Cette idée est illustrée dans la figure 13 . Sur le côté gauche, il montre un ActivityEdge du diagramme, la droite montre l'arc qui lui correspond de le cas de teste , comme indiqué par le point. Cette partie sera rempli par d'autres règles plus tard. Maintenant, nous supposons que ce ActivityEdge vient d'être ajouté au diagramme d'activités, qui est indiqué par la couleur verte et l'étiquette supplémentaire ++; Nous indiquons également, sur le côté droit, que l'arc correspondant doit être ajouté à la partie de cas de teste .

Par conséquent, ils sont présentés comme nœuds de contexte noirs. En dessous, il montre les pièces nouvellement ajoutées, indiquées en vert et marquées avec ++: Pour le domaine de diagramme d'activités , on a l'ActivityEdge, pour le domaine de cas de teste , on aura un nouveau arc . En outre, il y a un nouveau nœud de correspondance qui relie ces deux éléments. Le nœud avec les bords arrondis est une contrainte supplémentaire qui garantit que le nom de l'arc est le même que l'ActivityEdge.

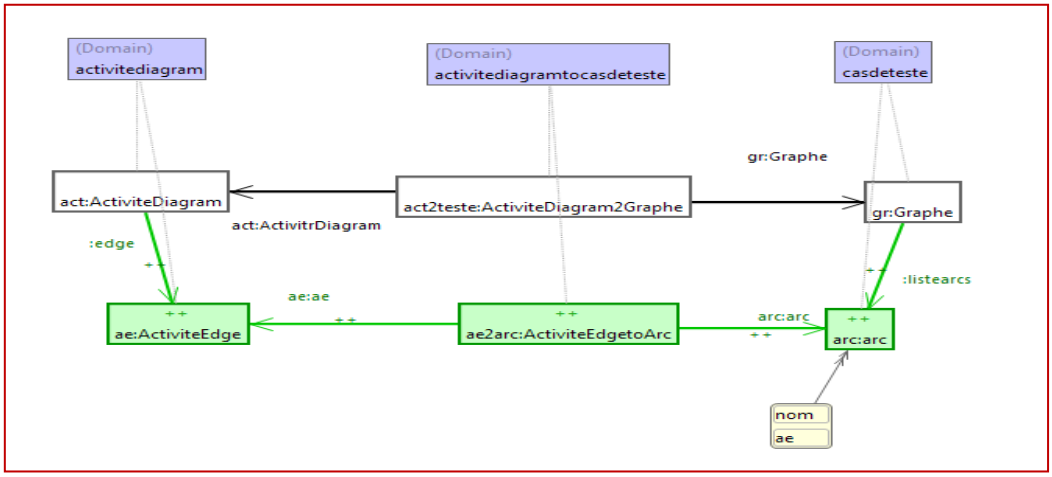

Figure 14: Règle TGG pour « activityEdge »

5-1-3. Transformation d'un "nœud Initial" en "classe" :

Maintenant, on va commencer les règles pour les nœuds la première règle est pour le nœud initial, nœud initial sera remplacé par une classe.

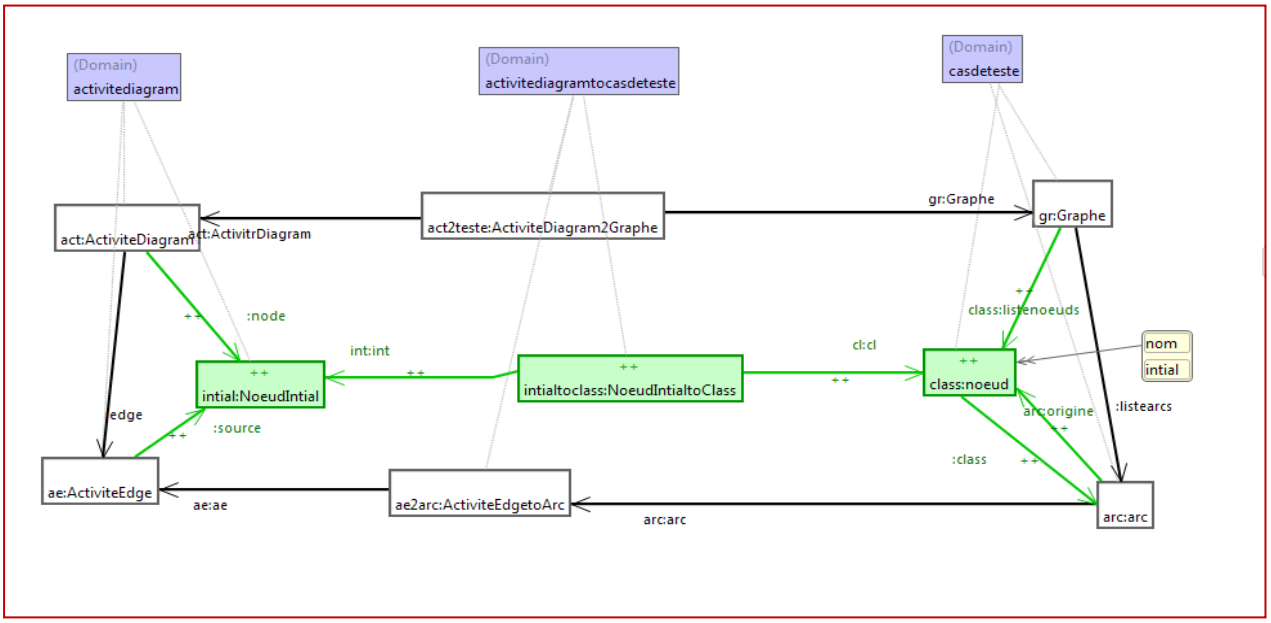

Figure 15 : règle TGG pour « nœud initial »

Dans ce cas, on aura une classe qui n'a pas d'arc entrant. La figure 14 montre cette idée en termes d'une règle TGG. Le nœud Initial nouvellement inséré correspond à une classe dans le cas de teste.

5-1-4. Transformation d'une "Action" en "classe" :

L'idée de transformer une action est très similaire

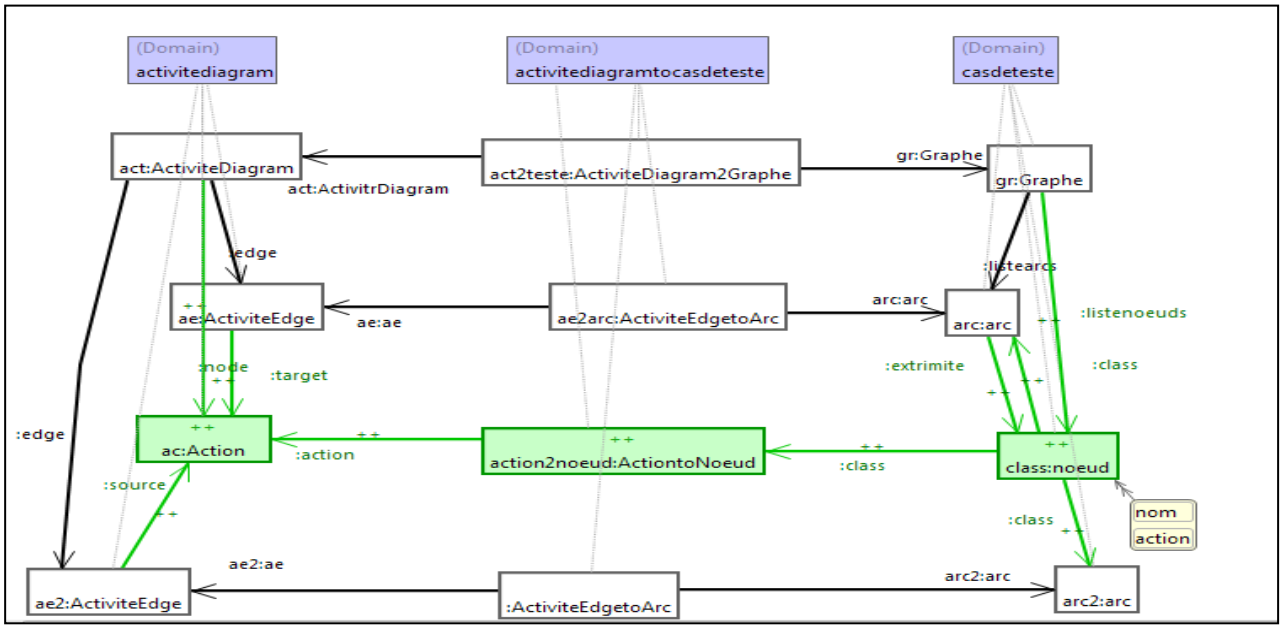

Figure 16 : règle TGG pour « Action »

5-1-5. transformation d'un "nœud Final" en "classe" :

L'idée de la transformation le nœud final sera remplacé par une classe qui n'a pas des arcs sortant.

Il peut y arriver qu'un nœud final à plusieurs edges entrants. Dans ce cas, la règle ci-dessus ne fonctionne pas. Depuis le nœud final est utilisé comme nœud de production pour la première règle. Dans le second, il est plus nouveau. Par conséquent, nous avons besoin d'une règle supplémentaire qui utilise seulement un "nouvelle connexion" d'un edge vers le nœud final et affecte l'arc à la classe qui le correspond. Cette règle est illustrée dans la figure 16. Il est le même que ci-dessus, sauf que le nœud final est un nœud de contexte.

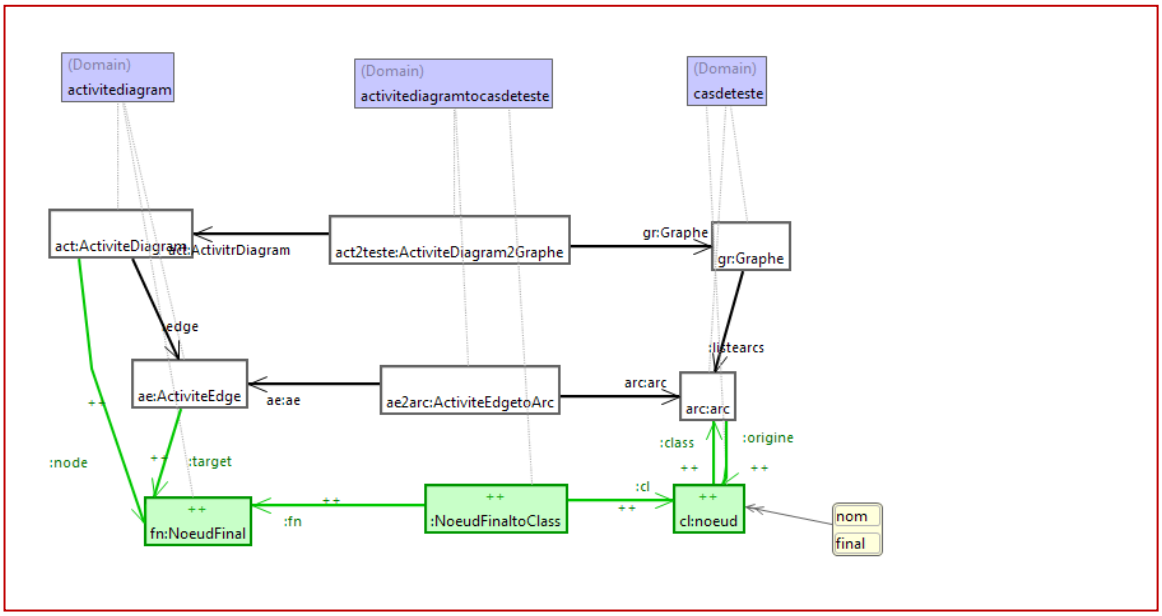

Figure 17 : règle TGG pour « nœud final »

Conclusion

Dans ce chapitre nous avons proposé une approche automatique pour transformer les diagrammes d'activité d'UML vers les cas de teste. La méthode proposée se base sur la transformation de graphes, on utilisant l'outil de modélisation et de méta modélisation multi formalismes TGG.

Nous avons proposé trois méta modèles, un pour les diagrammes d'activité, le deuxième pour les générations des cas de testes et troisième pour méta modèle correspondance présenter l'ensemble des règles TGGs (grammaire du graphe) permettant la réalisation de la transformation du diagramme d' activité UML2.0 vers génération des cas de teste .

Dans ce chapitre, nous avons présenté la mise en œuvre de la transformation de modèles basée sur l'utilisation des grammaires de graphes. Notre travail s'est axé spécialement sur la spécification des transformations avec le formalisme TGG et leur implémentation avec l'outil TGG Interpréter intégré dans l'environnement EMF.

Conclusion Générale

Le travail présenté dans ce mémoire s'inscrit dans le domaine de l'ingénierie dirigée par les modèles. Il se base essentiellement sur l'utilisation combinée de méta modélisation et de transformation de modèle. Plus précisément, la transformation de graphe est utilisée comme outil de transformation de modèles. Le résultat de notre travail est une approche automatique pour transformer les diagrammes d'activité d'UML2.0 vers cas de teste.

L'approche proposée est basée sur la transformation de graphe , est réalisée à l'aide de l'outil TGG.

Le travail est réalisé dans deux étapes : La première étape consiste à proposer trois méta modèles, un pour les diagrammes d'activité, le deuxième pour cas de teste et un autre pour la correspondance. La deuxième étape propose un ensemble des règles TGGs (grammaire du graphe) permettant la réalisation de la transformation du diagramme d'activité UML vers cas de teste La méthode proposée est basée sur la transformation de graphes.

BIBLIOGRAPHIE

[1] G. BOOCH, J. RUMBAUGH AND I. JACOBSON, “THE UNIFIED MODELING LANGUAGE USER GUIDE”, PUBLISHER: ADDISON WESLEY, FIRST EDITION OCTOBER 20, 1998.

[2] K. HAMILTON AND R. MILES, LEARNING UML 2.0. O’REILLY, APRIL 2006.

[3] J. RUMBAUGH, I. JACOBSON, AND G. BOOCH, THE UNIFIED MODELING LANGUAGE REFERENCE MANUAL. ADDISON WESLEY LONGMAN, INC, 1998.

[4] S. DIAW, R. LBATH, B. COULETTE, “ ETAT DE L’ART SUR LE DEVELOPPEMENT LOGICIEL BASE SUR LES TRANSFORMATIONS DE MODELES”. LABORATOIRE IRIT-UTM, UNIVERSITE DE TOULOUSE 2-LE MIRAI, 2009

[5] ANDY SCHÜRR. SPECIFICATION OF GRAPH TRANSLATORS WITH TRIPLE GRAPH GRAMMARS. IN GRAPH-THEORETIC CONCEPTS IN COMPUTER SCIENCE, PAGES 151–163. SPRINGER, 1995.

[8] O. M. OMG, “UNIFIED MODELING LANGUAGE : SUPERSTRUCTURE,” TECH. REP., OMG, MARCH ,2005.

[9] BENOIT COMBEMALE. APPROCHE DE METAMODELISATION POUR LA SIMULATION ET LA VERIFICATION DE MODELE–APPLICATION A L’INGENIERIE DES PROCEDES. THESE DE DOC- TORAT, INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE-INPT, 2008.

[10] AMINE LAJMI. USINE LOGICIELLE DE COMPOSANTS DE SIMULATION DE PROCEDES CAPE-OPEN. THESE DE DOCTORAT, PARIS 6, 2010.

[11] ALAIN PLANTEC. TERMINOLOGIE DOCUMENTEE DE L’INGENIERIE DIRIGEE PAR LE MODELES. TECHNICAL REPORT,UNIVERSITE DE BRETAGNE OCCIDENTALE, 2013.

[12] OMG. META OBJECT FACILITY (MOF) CORE SPECIFICATION VERSION 2.0, 2006.

[13] CYRIL FAUCHER. MODELISATION D’EVENEMENTS COMPOSITES REPETITIFS, PROPRIETES ET RELATIONS TEMPORELLES. THESE DE DOCTORAT, UNIVERSITE DE LA ROCHELLE, 2012.

[14] K. CZARNECKI AND S. HELSEN, “CLASSIFICATION OF MODEL TRANSFORMATION APPROACHES”, OOPSLA’03 WORKSHOP ON GENERATIVE TECHNIQUES IN THE CONTEXT OF MODEL- DRIVEN ARCHITECTURE, 2003.

[15] MECHERI NACERA, UNE APPROCHE HYBRIDE POUR TRANSFORMER LES MODELES, UNIVERSITE D’ ORAN, 2015

[**16**] SCHURR. SPECIFICATION OF GRAPH TRANSLATORS WITH TRIPLE GRAPH GRAMMARS.

[17] FRANCISCO DE LA PARRA. APPLICATION OF GRAPH GRAMMARS TO MODEL TRANSFORMA- TIONS. TECHNICAL REPORT: 2013-604, 2013.

[18] GGS FOR TRANSFORMING UML TO CSP : CONTRIBUTION TO THE ACTIVE ,2007.

[19] GABRIELE TAENTZER. AGG: A TOOL ENVIRONMENT FOR ALGEBRAIC GRAPH TRANSFOR- MATION. IN APPLICATIONS OF GRAPH TRANSFORMATIONS WITH INDUSTRIAL RELEVANCE, PAGES 481–488. SPRINGER, 2000.

[20] JUAN DE LARA AND HANS VANGHELUWE. ATOM3: A TOOL FOR MULTI-FORMALISM AND META MODELLING. IN FUNDAMENTAL APPROACHES TO SOFTWARE ENGINEERING, PAGES 174–188. SPRINGER, 2002.

[21] AMIR HOSSEIN GHAMARIAN, MAARTEN DE MOL, AREND RENSINK, EDUARDO ZAMBON, AND MARIA ZIMAKOVA. MODELLING AND ANALYSIS USING GROOVE. INTERNATIONAL JOURNAL ON SOFTWARE TOOLS FOR TECHNOLOGY TRANSFER, 14(1):15–40, 2012.

[24] TIHAMER LEVENDOVSZKY AND HASSAN CHARAF. APPLYING METAMODELS IN SOFTWARE MODEL TRANSFORMATION METHODS. PHD THESIS, BUDAPEST, 2005.

[25] ENRICO BIERMANN, CLAUDIA ERMEL, LEEN LAMBERS, ULRIKE PRANGE, OLGA RUNGE, AND GABRIELE TAENTZER. INTRODUCTION TO AGG AND EMF TIGER BY MODELING A CONFERENCE SCHEDULING SYSTEM. INTERNATIONAL JOURNAL ON SOFTWARE TOOLS FOR TECHNOLOGY TRANSFER, 12(3-4):245–261, 2010.

[26] AYR, G. SCHMIDT, AND G. TINHOFER, EDITORS, GRAPH-THEORETIC CONCEPTS IN COMPUTER SCIENCE, 20TH INTERNATIONAL WORKSHOP, WG ’94, VOLUME 903 OF LNCS, PAGES 151-163, HERRSCHING, GERMANY, JUNE 1994,

[30] MILIND G. LIMAYE (2009). SOFTWARE TESTING. TATA MCGRAW-HILL EDUCATION. P. 216. ISBN 978-0-07-013990-9.

[31] PATTON, RON (2005). SOFTWARE TESTING (2ND ED.). INDIANAPOLIS: SAMS PUBLISHING. ISBN 978-0672327988.

[34] FRANCISCO DE LA PARRA. APPLICATION OF GRAPH GRAMMARS TO MODEL TRANSFORMA- TIONS. TECHNICAL REPORT: 2013-604, 2013

[35] FRANK BUDINSKY, STEPHEN A. BRODSKY, AND ED MERKS. ECLIPSE MODELING FRAME- WORK. PEARSON EDUCATION, 2003.

[37] HOLGER GIESE, STEPHAN HILDEBRANDT, AND STEFAN NEUMANN. MODEL SYNCHRO- NIZATION AT WORK: KEEPING SYSML AND AUTOSAR MODELS CONSISTENT. IN GRAPH TRANSFORMATIONS AND MODEL-DRIVEN ENGINEERING, PAGES 555–579. SPRINGER, 2010.

[38] JOEL GREENYER AND EKKART KINDLER. RECONCILING TGGS WITH QVT. IN MODEL DRIVEN ENGINEERING LANGUAGES AND SYSTEMS, PAGES 16–30. SPRINGER, 2007.

[39] WANG. L., YUAN, J., YU, X., , HU, J., LI , X., ZHENG G., “GENERATING TEST CASES FROM UML ACTIVITY DIAGRAM BASED ON GRAY-BOX METHOD,” NATIONAL NATURAL SCIENCE FOUNDATION OF CHINA, 2005.

[40] HENOK BEKELE, ETHIOPIA MESFIN KIFLE ‘TEST CASE GENERATION FROM UML ACTIVITY DIAGRAM: THE USER’ PERSPECTIVE APPROACH, SAVE THE CHILDREN, ADDIS ABABA

[41] INTERNATIONAL JOURNAL OF COMPUTER APPLICATIONS TEST CASE GENERATION FROM BEHAVIORAL UML MODELS (NO.8, SEPTEMBER 2010

BIBLIO WEB

[6] L.Perochon, “UML : langage graphique de modélisation”, Juin 2009, http://www.projet-plume.org/ressource/uml(INRA)

[7] L. AUDIBERT, “UML 2”, EDITION 2007-2008, ADRESSE DU DOCUMENT : HTTP://WWW-LIPN.UNIV- PARIS13.FR/AUDIBERT/PAGES/ENSEIGNEMENT/COURS.HTM

[22] GROOVE. HOMEPAGE. HTTP://GROOVE.CS.UTWENTE.NL/ABOUT/.

[23] GRGEN. HOMEPAGE. HTTP://WWW.INFO.UNI-KARLSRUHE.DE/SOFTWARE/GRGEN/.

[27] WWW.TELECOM-LILLE1.EU/PEOPLE/VANWORMHOUDT/SITEEMFOCL.

[28] CLIPSE MODELING FRAMEWORK HOME PAGE. http://WWW.ECLIPSE.ORG /MODELING /EMF/.

[29] HTTP:/COMBEMALE.PERSO.ENSEEIHT.FR/TEACHING/METAMODELIN

G0708 /IDMOCLV1. 1PDF.

[32] BLACK BOX . HTTPS://WWW.GURU99.COM/BLACK-BOX-TESTING.HTML?

[33] WHITE BOX HTTP://SOFTWARETESTINGFUNDAMENTALS.COM/WHITE-BOX-TESTING/

[36] TGG HOME PAGE : WWW.INFORMATIK.UNI-MARBURG.DE