**RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE**
**Ministère de l'Enseignement Supérieur et de la Recherche Scientifique**

**Université Saad Dahleb de Blida**
**Faculté des Sciences**
**Département d'Informatique**

**MÉMOIRE DE FIN D'ÉTUDES POUR L'OBTENTION DU DIPLÔME DE MASTER EN INFORMATIQUE**
Option : Ingénierie des logiciels

***THEME :***

# Transformation de modèles : d'une spécification de missions en MDL vers une spécification SysML



**Présenté par :**

- **KACHOUANE Riad**
- **ZEFFANE Mehdi**

**Encadreur : M^me CHERFA Imene**
**Président : M^r BAOUYA Abdelhakim**
**Examinateur : M^me CHIKHI Imene**



**Soutenu le : 02 Juillet 2018**

**2017 - 2018**

# Remerciements

En préambule à ce mémoire, nous remercions tout d'abord ALLAH de nous avoir donné le courage et la patience afin d'élaborer notre travail, et ainsi achever nos études.

Nous tenons à remercier sincèrement Madame Cherfa Imane, en tant que promotrice et encadreur, qui s'est montrée à l'écoute et étant toujours disponible tout au long de la réalisation de ce mémoire, ainsi pour l'inspiration, l'aide et le temps qu'elle a bien voulu nous donner.

Nous exprimons notre gratitude et reconnaissance aux professeurs de département d'informatique qui nous ont tant donné pour être ce que nous sommes aujourd'hui.

On n'oublie pas les personnes qui nous ont apporté leur aide, et leur soutien moral, nos parents, nos amis, nos proches, même par un mot d'encouragement, de loin ou de prés.

Merci à tous et à toutes.

# Dédicaces

*A mes chers parents pour leur soutien moral,*

*encouragements et sacrifices*

*A ma chère famille*

*Pour leur affection et tendresse*

*A mon binôme Riad pour sa bonne collaboration*

*A tous mes amis*

*Au bonheur des plus chers*

*Je dédie cet humble travail*

*Zeffane Mehdi*

*Le parcours d'une vie est jalonné d'opportunités qui dépendent de nous, mais également des personnes qu'il nous a été donné de rencontrer : des personnes qui nous guident, qui nous conseillent et qui nous font confiance, j'ai eu de la chance de rencontrer quelques-uns d'entre eux et à qui je tiens à leur dédier ce mémoire*

*A tous ceux qui m'ont soutenu tout au long de ce projet.*
*A mes chers parents, que nul dédicace ne puisse exprimer ce qu'on leurs doit, pour leur bienveillance, leur affectation et leur soutien morale durant l'élaboration de ce travail, en témoignage de notre profond amour et nos sincères reconnaissances pour les efforts qu'ils ont consenti pour l'accomplissement de nos études.*
*« Que dieu vous préserve et vous procure santé »*
»اللهم اجعل عملنا هذا في□يزان حسناتهم«

*A ma chère famille, à mes chers amis, à mon binôme Mehdi, aux personnes qui m'ont soutenu, qui m'ont aidé, qui m'ont encouragé par des efforts ou par un mot, tout au long de la réalisation de ce projet*

*A notre encadreur à l'université de Blida Madame Cherfa Imane, pour sa gentillesse, sa disponibilité, et ses efforts remarquables.*

*Je leurs dédie ce modeste travail*

*Kachouane Riad*

## Résumé :

Un système de systèmes (SoS) est un ensemble de systèmes constitutifs, il a comme objectif l'accomplissement d'une mission.

Pendant le développement du système, un expert du domaine de l'application de la mission définit les exigences du système, ensuite viendra le rôle de l'architecte du système qui doit le concevoir.

Notre outil permet d'assurer la communication entre l'étape de définition des éxigences et l'étape de génération d'architecture abstraite en proposant la transformation automatique entre les deux modèles. Cette transformation est implémentée par le langage de trasformation ATL.

Mots clés : SoS, mission, architecture abstraite, transformation de modèles, ATL

## Abstract:

A system of systems (SOS) is a set of constituent systems; it has as objective the accomplishment of a mission.

During the development of the system, an expert in the field of the application of the mission defines the requirements of the system, and then the architect of the system has the role of designing it.

Our tool provides communication between the requirement definition step and the abstract architecture generation step by proposing the automatic transformation between the two models. The ATL transformation language implements this transformation.

Keywords: SOS, mission, abstract architecture, model transformation, ATL

**ملخص:**

أنظمة الأنظمة (SOS) هي مجموعة من الأنظمة المستقلة، الهدف منها هو إنجاز المهام المتعلقة بنظام الأنظمة.

اثناء تطوير النظام، يقوم خبير في مجال تطبيق معين بتحديد متطلبات النظام، ثم يقوم مهندس النظام بدور تصميم هذا النظام.

توفر أداتنا التواصل بين خطوة تحديد المتطلبات وخطوة إنشاء المخطط المجرد عن طريق اقتراح التحويل التلقائي بين النموذجين .يتم تنفيذ هذا التحويل بواسطة لغة تحويل النماذج ATL.

الكلمات الرئيسية: SOS، المهام، المخطط المجرد، تحويل النموذج، ATL

## Table des matières

**Chapitre 3 : Source et cible de la transformation**



**Partie II : Contribution**

**Chapitre 4 : Conception**



**Chapitre 5 : Implémentation**

## Liste des figures

### Partie I : Etat de l'art

### Chapitre 1 : Les systèmes de systèmes (SOS)



### Chapitre 2 : Méta modélisation et transformation de modèles



### Chapitre 3 : Source et cible de la transformation

**Partie II : Contribution**

**Chapitre 4 : Conception**



**Chapitre 5 : Implémentation**



**Liste des tableaux**

## Liste des abréviations

| | |
|---|---|
| **ADL** | Architecture Description Language |
| **ADT** | ATL Development Tool |
| **ATL** | Atlas Transformation Language |
| **BOTL** | Bidirectional Object oriented Transformation Language |
| **CIM** | Computational Independent Models |
| **CMM** | Capability Modeling Metamodel |
| **CMOF** | Complete Meta Object Facility |
| **DTD** | Document Type Definition |
| **EMF** | Eclipse Modeling Framework |
| **EMOF** | Essential Meta Object Facility |
| **IDM** | Ingénierie Dirigée par les Modèles |
| **INCOSE** | International Council On Systems Engineering |
| **MDA** | Model Driven Architecture |
| **MDE** | Model Driven Engineering |
| **mKAOS** | Model of Keep All Objectives Satisfied |
| **MOF** | Meta Object Facility |
| **OCL** | Object Constraint Language |
| **OMG** | Object Management Group |
| **PIM** | Platform Independent Models |
| **PSM** | Platform Specific Models |
| **QVT** | Query Views Transformation |
| **QVTE** | Query Views Transformation Eclipse |
| **SOS** | System Of Systems |
| **SOSE** | System Of Systems Engineering |
| **SYSML** | SYStem Modeling Language |

| | |
|---|---|
| **UML** | Undefined Modeling Language |
| **UMT-QVT** | UML Model transformation Tool-QVT |
| **XMI** | XML Metadata Interchange |
| **XML** | eXchange Metadata Language |

# Introduction générale

## Contexte

Ces dernières années, il y a eu un intérêt croissant et récent pour la recherche et le développement de systèmes complexes et de grande taille résultant de l'intégration de plusieurs systèmes indépendants sur le plan opérationnel, conduisant à une nouvelle classe de systèmes appelés systèmes de systèmes « SoS ». Un SoS peut être compris comme un ensemble étendu de systèmes constitutifs indépendants et hétérogènes pour former un système plus vaste afin d'accomplir une mission donnée. Chaque système constitutif accomplit sa propre mission individuelle et peut contribuer à l'accomplissement de la mission globale du SoS. Par conséquent, les systèmes constitutifs et leurs objectifs individuels contribuent à la réalisation des objectifs établis pour le SoS.

Une mission est généralement considérée comme un but, une fonctionnalité ou un ensemble de tâches à réaliser par les systèmes constituants, qui peuvent être conscients de la mission du système et doivent coopérer et échanger des informations pour l'accomplir. Les initiatives existantes en termes de missions de SoS sont encore dans une phase initiale de développement, vu le manque de moyens appropriés pour traiter les caractéristiques inhérentes d'un SoS.

## Problématique

Au cours du processus de développement du SoS, les choix et les décisions sont prises à chaque étape. Par exemple, l'expert du domaine d'application prend des décisions sur les aspects métier, tandis que l'architecte système est responsable des choix d'implémentation. Ces deux actionnaires couvrent les étapes concernant la définition des exigences et la conception. Ces deux étapes sont cruciales pour le développement du système car elles forment sa base. Ainsi, les choix effectués lors de la conception doivent être cohérents avec les décisions prises lors de la définition des exigences. Ceci est nécessaire pour faciliter l'évolution du système. Si la définition des exigences n'est pas formelle, le risque de déviation de la conception par rapport aux objectifs initiaux est réel à chaque évolution du système.

## Objectifs

Pour résoudre ce problème, nous proposons de créer un lien fort entre l'étape de définition des exigences et l'étape de conception SoS. L'idée est de permettre à l'expert du domaine d'application de définir de manière formelle ses objectifs et ses exigences qui serviront de guide et de contrôleur des choix proposés par l'architecte système lors des phases de conception et d'évolution. Nous suggérons qu'après la spécification de la mission par l'expert du domaine d'application l'architecture abstraite soit générée automatiquement.

Notre travail est basé sur l'approche MDA (Model Driven Architecture). C'est une approche de conception basée sur la transformation des modèles, supportant le développement de systèmes complexes et distribués. L'approche MDA est basée sur la transformation de modèles, de sorte que la construction du système soit un séquencement de modèles et de transformations entre ces modèles.

## Organisation du mémoire

Ce mémoire est constitué de cinq chapitres organisés en deux principales parties.

- Partie 1 : Etat de l'art qui est composée à son tour de trois chapitres :
  - Le premier chapitre introduit la notion du système de systèmes (SoS) avec ses différentes caractéristiques, et traite la spécification de la mission ainsi que ses objectifs et ses utilisations.
  - Dans le deuxième chapitre nous présentons la notion de modèles, de la métamodélisation et les différents concepts liés à la métamodélisation, ainsi qu'une exploration vaste sur la transformation de modèles en définissant ses langages et ses propriétés.
  - Le troisième chapitre est dédié à l'état de l'art de la transformation prévue à la fin du projet, présentant théoriquement les métamodèles utilisés ainsi que leurs types et leurs composants.

- Partie 2 : Contribution qui est composée de deux chapitres :

  - Le quatrième chapitre consiste à adapter les métamodèles pour la spécification de mission, puis concevoir la transformation depuis la spécification de mission vers une architecture abstraite du SoS.
  - Dans le cinquième chapitre, nous présentons l'implémentation de notre application en utilisant le langage ATL sous Eclipse. Nous avons présenté les outils utilisés pour implémenter notre application ainsi qu'une étude de cas montrant la transformation et ses règles de transformation sur un exemple concret.

# Partie I : Etat de l'art

- Chapitre 1 : SOS
- Chapitre 2 : Métamodélisation et transformation de modèles
- Chapitre 3 : Source et destination de la transformation

# Chapitre 1

## *LES SYSTEMES DE SYSTEMES (SOS)*

# 1. Introduction

Bien que l'expression système de systèmes (SoS) soit communément observée, il y a moins d'accord sur ce qu'ils sont, sur la façon dont ils peuvent être distingués des systèmes conventionnels ou sur la façon dont leur développement diffère des autres systèmes.

Dans ce chapitre, nous présentons une étude générale des SoS, des missions de SoS, et la relation entre les deux, en se focalisant sur les langages de description de missions et sur la définition d'architecture d'un SoS.

# 2. Présentation des systèmes de systèmes (SOS)

On commence tout d'abord par définir ce qu'est un système.

## 2.1 Définition d'un système

Un système est un ensemble de composants inter reliés qui interagissent les uns avec les autres d'une manière organisée pour accomplir un objectif défini [1].

## 2.2 L'ingénierie des systèmes

Le terme ingénierie des systèmes peut être retracé au moins jusqu'aux années 1940, mais à ce jour aucune définition unique et universelle du terme n'existe. Souvent, l'ingénierie des systèmes est définie par le contexte dans lequel elle est incorporée. Une définition de la pratique classique de l'ingénierie des systèmes est «une approche interdisciplinaire de la traduction des besoins des utilisateurs dans la définition d'un système, de son architecture et de sa conception à travers un processus itératif qui se traduit par une efficacité opérationnelle du système ». L'ingénierie de systèmes s'applique sur tout le cycle de vie, du développement de concept à l'élimination finale [2].

## 2.3 Définition d'un SOS

Un système de systèmes (SoS) est un ensemble ouvert de systèmes complémentaires et interactifs appelés « systèmes constitutifs » avec des propriétés, des capacités et des comportements émergents qui résultent de l'interaction entre l'ensemble des systèmes [3].

## 2.4 Types de SOS

### 2.4.1 SOS virtuel

Ce type de SoS manque d'une autorité de gestion centrale et d'un but clair de SoS. Il est souvent ponctuel et les systèmes constitutifs ne sont pas nécessairement connus. Un exemple de ce type de SoS est l'Internet et tous les services que l'on peut trouver et intégrer de manière ponctuelle [4].

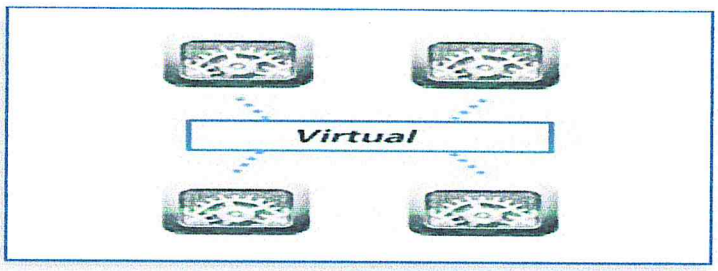



**Figure 01 :** SOS virtuel [5]

### 2.4.2 SoS collaboratif

Dans un SoS collaboratif, les équipes d'ingénierie d'un système constitutif travaillent ensemble plus ou moins volontairement pour remplir des objectifs centraux convenus. Dans ce type de SoS, il n'y a pas d'équipe d'ingénierie SoS pour orienter ou gérer les activités liées aux SoS. Un exemple de ce type de SoS pourrait être le système régional d'intervention en cas de crise où chaque organisme qui participe aux types de situations des premiers intervenants est responsable de ses propres systèmes [4].

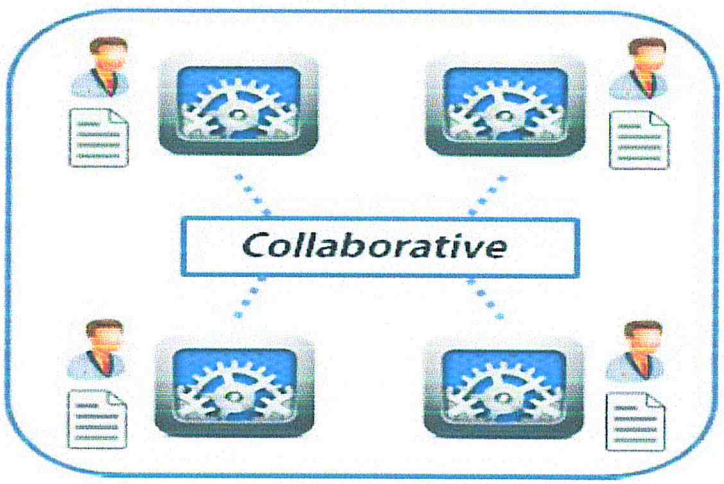



**Figure 02 :** SOS collaboratif [5]

### 2.4.3 SOS reconnu

Les SoS reconnus ont des objectifs, un gestionnaire désigné et des ressources au niveau SoS. Par exemple, une équipe SoSE. Mais l'équipe SoSE n'a pas l'autorité complète sur les systèmes constitutifs. Les systèmes constitutifs maintiennent leur propriété indépendante, leurs objectifs, leur financement et leurs approches de développement. Un exemple de ce type de SoS pourrait être un SoS de commandement et de contrôle militaire qui est passé d'un SoS collaboratif à un SoS reconnu en raison de l'importance des missions appuyées par le SoS ou de la complexité des capacités transversales de SoS [4].


**Figure 03 :** SoS reconnu [5]

### 2.4.4 SOS dirigé

Un SoS dirigé est géré de façon centralisée par une équipe de gouvernement ou d'entreprise et est construit pour réaliser des objectifs spécifiques. Les systèmes constitutifs maintiennent leur capacité d'opérer indépendamment, mais l'évolution est principalement contrôlée par l'organisation de gestion de SoS. Des exemples de ce type de SOS pourraient être le SOS de soins de santé [4].

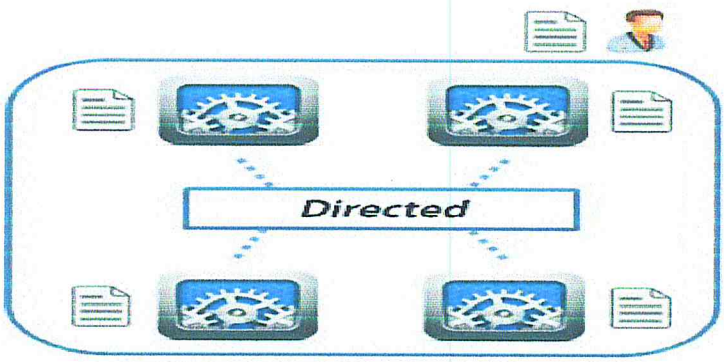



**Figure 04 :** SOS dirigé [5]

### 2.5 Caractéristiques des SOS

Les SoS se distinguent des systèmes monolithiques par l'indépendance des systèmes constitutifs, ainsi que par la nature évolutive et le comportement émergent du SOS dans son ensemble. La composition du système forme les qualités qui s'élèvent de la collaboration entre les constituants. Cinq caractéristiques ont été identifiées dans [5] pour les SoS (acronyme «ABCDE») [4] :

- **A**utonomie : Chaque système est libre et indépendant avec son propre but d'opération.
- Appartenance « **B**elonging » : Les systèmes fonctionnent en collaboration pour atteindre un objectif plus élevé commun.
- **C**onnectivité : La synergie est activée par le réseau distribué très dynamique.
- **D**iversité : Les constituants sont des systèmes auto-suffisants hétérogènes qui sont ouverts à l'amélioration par l'évolution et l'adaptation.
- **É**mergence : Les actions cumulatives et les interactions entre les mandants d'un SoS donnent lieu aux comportements qui peuvent être attribués au SoS dans son ensemble.

### 2.6 Comparaison entre un Système et un SoS

Le tableau suivant montre les points similaires ainsi que les points différents entre un système classique et un SoS :

| **Aspect d'environnement** | **Système** | **SoS** |
|---|---|---|
| **Gestion et supervision** | | |
| **Implication des intervenants** | Un ensemble plus clair d'intervenants | Les intervenants, tant au niveau du système que des niveaux de SoS (y compris les propriétaires du système), avec des intérêts et des priorités contradictoires, dans certains cas, l'intervenant du système n'a aucun intérêt direct dans le SoS, toutes les parties prenantes peuvent ne pas être reconnues |
| **Environnement opérationnel** | | |

| | | |
|---|---|---|
| **Focus opérationnel** | Conçu et développé pour atteindre les objectifs opérationnels | Appelé pour but d'atteindre un ensemble d'objectifs opérationnels à l'aide de systèmes dont les objectifs peuvent ou non être alignés sur les objectifs de SoS |
| **Implémentation** | | |
| **Test et évaluation** | Le test et l'évaluation du système sont généralement possibles | Les tests sont plus difficiles en raison de la difficulté de synchroniser les cycles de vie de plusieurs systèmes, compte tenu de la complexité de toutes les pièces mobiles et du potentiel de conséquences imprévues |
| **Considérations d'ingénierie et de conception** | | |
| **Limites et interfaces** | Se concentre sur les frontières et les interfaces pour le système unique | Mettre l'accent sur l'identification des systèmes qui contribuent aux objectifs de SoS et permettre le flux de données, le contrôle et la fonctionnalité à travers le SoS tout en équilibrant les besoins des systèmes |
| **Performance et comportement** | Performance du système pour atteindre les objectifs fixés | Performance à travers le SoS qui répond aux besoins de la capacité des utilisateurs de SoS tout en équilibrant les besoins des systèmes |

**Tableau 01 :** Comparaison entre un système et un SoS [3]

### 2.7 Exemple de SOS

La défense aérienne intégrée : les défenses aériennes des forces militaires modernes sont communément considérées comme des exemples de systèmes de systèmes. Un système intégré de défense aérienne est composé d'un réseau géographiquement dispersé d'éléments semi-autonomes. Ce sont notamment les radars de surveillance, les systèmes de surveillance passive, les batteries de lancement de missiles, les sites de repérage et de contrôle des missiles, les radars de surveillance et de repérage aéroportés, les avions de chasse, et de l'artillerie antiaérienne. Toutes les unités sont liées entre elles par un réseau de communication avec commande et contrôle appliqués aux centres locaux, régionaux et nationaux [4].

## 3. Les outils de spécification des missions

Dans les systèmes de systèmes (SoS), une mission est une information essentielle qui peut guider l'ensemble du processus de développement de SoS. Grâce aux modèles et aux langages de spécification des missions, il est possible d'identifier les capacités requises pour le système constitutif, les opérations, les connexions et le comportement émergent entre les éléments qui caractérisent un SoS [6].

### 3.1 Modèle des missions pour SOS

Dans le domaine des SoS, la mission est encore un terme ambigu dont les éléments sont toujours pas bien définis. [7] a fait une première proposition de modèle pour caractériser la mission d'un SoS. Dans ce modèle, une mission peut être affinée en sous-missions, qui à leurs tours peuvent être représentées comme un ensemble ordonné de tâches qui peuvent utiliser certains paramètres liés à une telle mission. Il est important de savoir que les détails de la mise en œuvre ne devraient pas être inclus dans les niveaux de la mission, en particulier dans le cas de SoS dans lequel la description de la mission est plus complexe car elle doit englober les missions de SoS et les systèmes constitutifs [7].

Dans [7], une mission est caractérisée par cinq concepts principaux :

- La priorité : souvent représentée comme un nombre entier qui définit le niveau d'engagement du système au sein de la mission.
- Le déclencheur : qui définit une condition nécessaire et suffisante pour l'exécution de la mission.

- Les contraintes : divisées en invariants, contraintes qui doivent être satisfaites, et heuristiques, souhaitées, mais non nécessaires.
- Les paramètres, qui peuvent être fournis en entrée ou en sortie pour la mission.
- Les tâches : opérations fonctionnelles qui mettent en œuvre la mission.

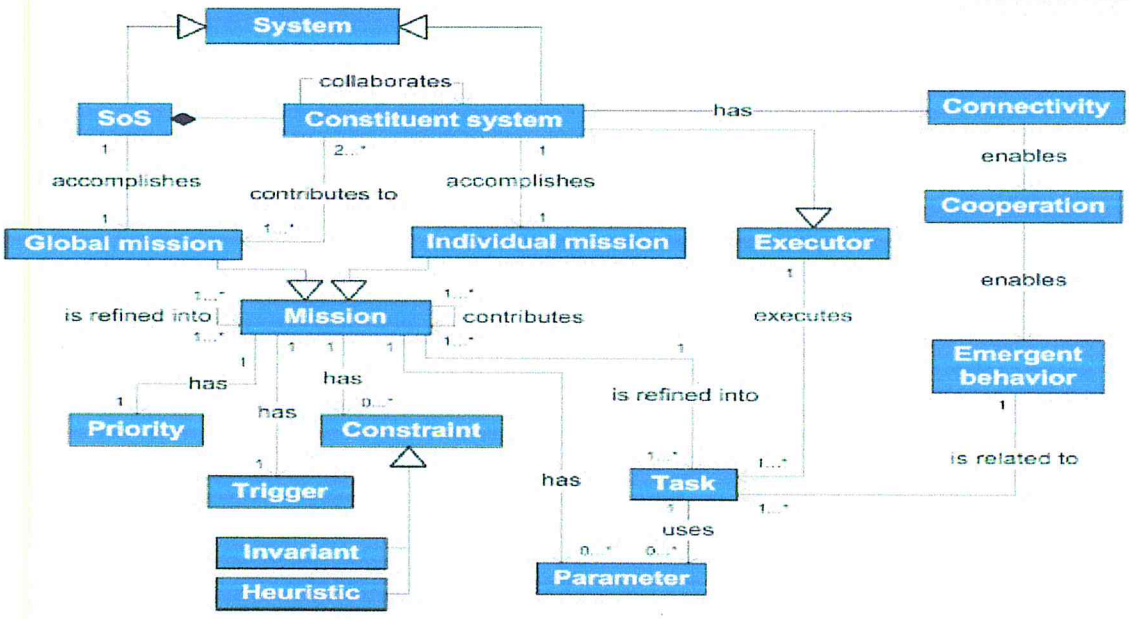



**Figure 05 :** Modèle conceptuel des missions pour SOS [7]

### 3.2 mKAOS

Selon [6], mKAOS est un langage pour la description de la mission dans SOS qui permet aux concepteurs :

(i) De décrire des missions globales et individuelles.

(ii) De relier ces concepts à divers aspects du système, tels que les systèmes constitutifs, les comportements émergents, et les capacités des systèmes.

(iii) Il favorise une séparation claire de la description de la mission de l'architecture du système, ce qui est un avantage puisque, dans les SoS, le concepteur n'a souvent pas accès à l'architecture des systèmes constitutifs ou les détails de mise en œuvre.

L'objectif principal mKAOS est de permettre la représentation explicite de ses concepts, et qui peuvent être utilisés pendant l'ensemble du processus de développement et d'entretien d'un SOS.

## 4. Architecture des SoS

[8] définit l'architecture en tant que les composants qui composent un système, les spécifications comportementales pour ces composants, et les modèles et les mécanismes d'interactions entre eux. Notons qu'un seul système est généralement composé de plus d'un type

de composant : modules, tâches, fonctions, etc. Une architecture peut choisir le type de composant le plus approprié ou instructif à montrer, ou il peut inclure plusieurs vues du même système, chacune illustrant différents composants.

Les langages de description d'architecture (ADLs) sont des langages formels qui peuvent être utilisés pour représenter l'architecture d'un système. Comme l'architecture devient un thème dominant dans le développement et l'acquisition de systèmes importants, les méthodes de spécification univoque d'une architecture deviendront indispensables.

- **SoSADL (Architecture Description Language)**

SoSADL [9] apparaît comme un langage formel pour décrire les architectures software SoS avec le soutien d'une analyse rigoureuse des mécanismes. Les fondements formels de SoSADL s'appuient sur une extension du processus π-calculus algebra[1], étant ainsi un modèle universel de calcul amélioré avec les préoccupations de SoS.

SosADL utilise le concept d'architectures abstraites et concrètes dans le sens que les systèmes constitutifs et ses communications seront réalisés au moment de l'exécution. Une architecture abstraite définit un SoS en termes de types de systèmes constitutifs, de sorte que plusieurs systèmes concrets d'un type donné pourraient faire partie des SoS et former son architecture concrète.

La figure 06 montre un exemple partiel d'un système constitutif de SoSADL. Le système de passerelle a un portail appelé notification, qui est composé de deux connexions, mesure (pour recevoir des données) et d'alerte (pour l'envoi de données). La garantie de ce système définit un protocole indiquant que le portail reçoit des valeurs via la connexion d'entrée de mesure et qu'il envoie des valeurs via la connexion de sortie d'alerte. Ces actions sont exécutées à plusieurs reprises, comme exprimé par la construction de répétition.

[1] π-calculus algebra est un modèle mathématique de processus dont les interconnexions changent à mesure qu'elles interagissent en envoyant des liens de communication entre eux

```
system Gateway(lps: Coordinate) is {
  ...
  gate notification is {
    connection measure is in{MeasureData}
    connection alert is out{MeasureData}
      // alert sent by the gateway
  } guarantee {
    protocol notificationpact is {
      repeat {
        via notification::measure receive any
        repeat {anyaction}
      }
      via notification::alert send any
    }
  }
```

**Figure 06 :** Exemple partiel d'un système constitutif en SosADL [9]

## 5. Conclusion

Les systèmes de systèmes (SoS) sont des systèmes dont la définition est basée sur des systèmes indépendants préexistants dans l'environnement d'exécution appelés « les systèmes constitutifs ». Les missions sont des informations essentielles dans un contexte de systèmes de systèmes, la définition de ces missions et les capacités des systèmes constitutifs de SOS sont nécessaires à la satisfaction des missions et la communication entre les systèmes.

Même s'il y'a eu des tentatives de définition de missions et des propositions pour décrire l'architecture d'un SoS, le dynamisme de l'environnement d'un SoS fait qu'il faut créer un lien fort entre la définition de mission, et la génération d'architecture. Ce lien doit permettre de générer et d'adapter une architecture automatiquement à partir d'une description de la mission. L'ingénierie dirigée par les modèles offre la possibilité de faire des transformations automatiques d'un modèle à un autre, c'est ce que nous allons détailler dans le chapitre qui suit.

# Chapitre 2

## METAMODELISATION ET TRANSFORMATION DE MODELES

# 1. Introduction

La modélisation en informatique est l'étape la plus importante dans le développement d'un logiciel. Elle facilite la compréhension du fonctionnement d'un système avant sa réalisation en produisant un modèle.

Nous commençons ce chapitre par introduire les notions de base de la modélisation et la méta-modélisation. Nous présentons ensuite la transformation de modèles en abordant ses objectifs ainsi que les langages dédiée pour elle.

# 2. Bases de l'ingénierie dirigée par les modèles (MDE)

Avant de parler de transformation de modèles, nous présentons l'architecture dirigée par les modèles.

## 2.1 Object Management Group (OMG)

L'Object Management Group (OMG) [10] est un consortium international, ouvert à tous, à but non lucratif et spécialisé dans les technologies à but non lucratif.

Fondé en 1989, les normes OMG sont dirigées par des fournisseurs, des utilisateurs finaux, des institutions académiques et des agences gouvernementales, dont l'objectif principal est d'établir des standards pour résoudre les problèmes d'interopérabilité entre les systèmes d'information. Ces standards sur lesquels repose le MDE (Ingénierie Dirigée par les Modèles) sont centrés sur les notions de métamodèles et de métamétamodèles.

## 2.2 L'architecture dirigée par les modèles (MDA)

D'après [11], l'OMG a proposé une variante particulière de l'IDM, c'est l'architecture dirigée par les modèles (MDA : Model Driven Architecture), qui est une approche complète de développement du logiciel basée sur le développement des modèles. MDA a pour objectif de résoudre les problèmes d'interopérabilité et de portabilité dès le niveau modélisation, elle essaye d'offrir une solution au problème continu de l'émergence des technologies software par la séparation entre les spécifications fonctionnelles et les spécifications d'implémentation sur une plate-forme donnée. Pour cela [11] propose trois classes de modèles : Computational Independent Models (CIM), Platform Independent Models (PIM) et Platform Specific Models (PSM).

- Le modèle CIM : Le CIM (Computational Independent Model) est un modèle de haut niveau qui représente l'application par la spécification des besoins (exigences) du client. Ces modèles ont pour objectif de créer et de refléter la relation entre les fonctionnalités de l'application et les autres entités avec lesquelles elle interagit. Les CIM, ne contiennent pas d'informations sur la réalisation de l'application ni sur les traitements ou le comportement intérieur d'une l'application.
- Le modèle PIM : Le PIM (Platform Independent Model) a pour objectif de structurer l'application en modules et sous-modules. En faisant le lien entre le modèle des besoins et le code de l'application. Les modèles d'analyse et de conception doivent être indépendants de toute plate-forme ou technologie de mise en œuvre.
- Le modèle PSM : Le PSM (Platform Specific Model) est la phase la plus délicate du MDA. La génération de code peut commencer à partir des modèles d'analyse et de conception. La différence principale entre un modèle de code et un modèle d'analyse et de conception réside dans le fait que le modèle de code est lié à une plate-forme spécifique.

### 2.3 L'ingénierie dirigée par les modèles (MDE)

L'Ingénierie Dirigée par les Modèles « IDM » ou « MDE : Model Driven Engineering » est une discipline récente du génie logiciel qui met les modèles au premier plan au sein du processus du développement logiciel.

Elle a apporté plusieurs améliorations significatives dans le développement des systèmes logiciels complexes en fournissant des moyens permettant de passer d'un niveau d'abstraction à un autre ou d'un espace technologique à un autre. Cependant, la gestion des modèles peut s'avérer lourde et coûteuse. Pour pouvoir mieux répondre aux attentes des utilisateurs, il est nécessaire de fournir des outils flexibles et fiables pour la gestion automatique des modèles ainsi que des langages dédiés pour leurs transformations [12].

La figure suivante représente les différentes couches de spécification de la démarche MDA.

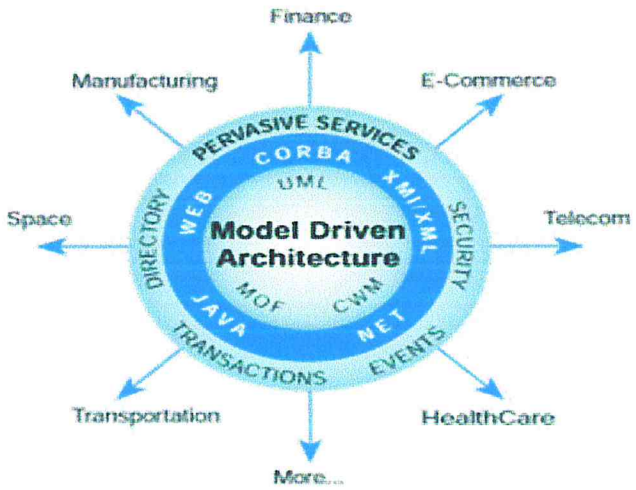



**Figure 07 :** Model Driven Architecture (OMG) [12]

### 2.4 Le modèle

Un modèle est une représentation abstraite qui contient un ensemble restreint d'informations sur un système réel et un point de vue différent ou perspective de ce système. D'autre part, la modélisation offre des avantages considérables aux concepteurs des systèmes tels que : la facilité de compréhension du fonctionnement des systèmes avant sa réalisation et un bon moyen de maîtriser sa complexité et d'assurer sa cohérence. En informatique la modélisation est vue comme une séparation entre les différents besoins fonctionnels et non fonctionnels (tels que : la sécurité, la fiabilité, l'efficacité, la performance, la flexibilité, ..., etc.).

Par ailleurs, la modélisation d'un système est venue à signifier ce qui représente un système en utilisant une sorte de notation graphique [11].

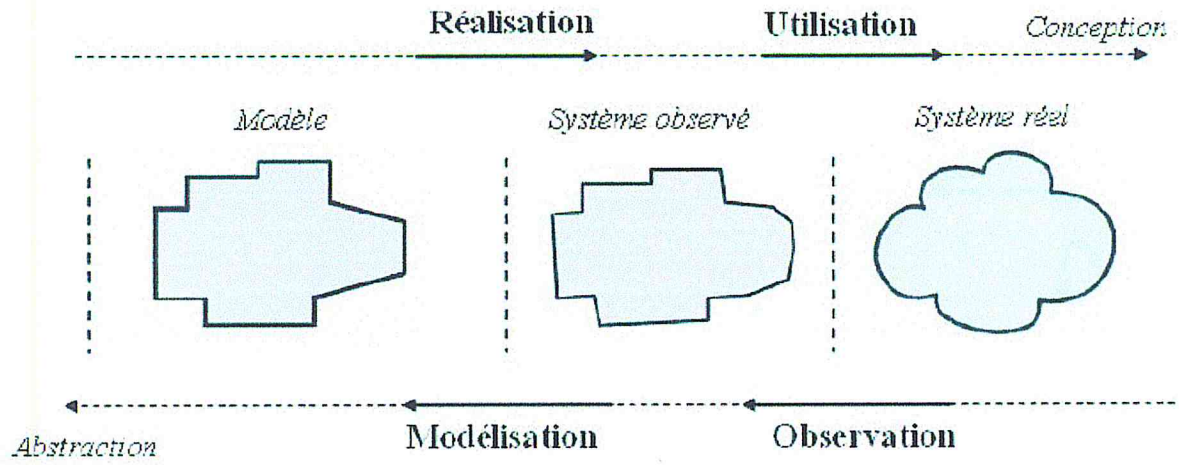



**Figure 08 :** Niveau d'abstraction en modélisation [11]

## 2.5 Types de modélisation

La modélisation peut se classer selon le degré du formalisme des langages ou des méthodes employées dans le processus de la modélisation. La modélisation peut être considérée comme étant formelle, semi-formelle ou informelle [12].

### 2.5.1 Modélisation informelle

Le processus de modélisation informelle à base d'un langage informel, se justifie pour plusieurs raisons [12] :

- La facilité de compréhension d'un langage permet des consensus entre les personnes qui spécifient et celles qui commandent un logiciel.
- Elle représente une manière familière de communication entre personnes.

### 2.5.2 Modélisation semi-formelle

Le processus de modélisation semi-formelle est basé sur un langage textuel ou graphique pour lequel une syntaxe précise est définie avec une sémantique. La sémantique d'un tel langage est souvent assez faible. Néanmoins, ce type de modélisation permet d'effectuer des contrôles et de réaliser des automatisations pour certaines tâches. La puissance expressive du modèle graphique est utilisée dans la plupart des méthodes de modélisation semi formelles [12].

Par ailleurs, la modélisation semi-formelle s'appuie sur des langages graphiques, tels que : UML qui permet la production de modèles assez faciles à interpréter.

### 2.5.3 Modélisation formelle

La modélisation formelle est un processus de développement rigoureux basé sur des notations formelles avec une sémantique précise, ainsi que sur des vérifications formelles. Le principal avantage des spécifications formelles est leur capacité à exprimer une signification précise, en permettant de cette manière des vérifications de la cohérence et de la complétude d'un système [12].

### 2.6 Un méta-modèle

Un métamodèle [11] est un modèle qui permet de définir le langage d'expression ou la structure d'un modèle. Autrement dit, la méta-modélisation modélise les entités d'un système, le lien existant entre un modèle et le système qu'il représente avec les contraintes existantes, c'est-à-dire un méta-modèle est une spécification de la syntaxe et la sémantique d'un système. Pour illustrer la notion de métamodèle on peut citer l'exemple suivant : Un programme source Java est un modèle pour toutes ses exécutions possibles. Le méta-modèle d'un programme source Java est la grammaire de Java. La grammaire définit un ensemble de programmes syntaxiquement valides. Un programme source conforme à la grammaire appartient à cet ensemble.

### 2.7 Un méta-méta-modèle

Le méta-méta-modèle [11] est un méta-modèle pour les méta-modèles utilisé tout naturellement pour désigner ce métamodèle particulier. Le langage utilisé au niveau du métamétamodèle doit être suffisamment puissant pour spécifier sa propre syntaxe abstraite et ce niveau d'abstraction demeure largement suffisant (métacirculaire).
La relation entre un méta-méta-modèle et un méta-modèle est analogue à la relation entre un méta-modèle et un modèle. Cette relation est illustrée dans la figure 09.

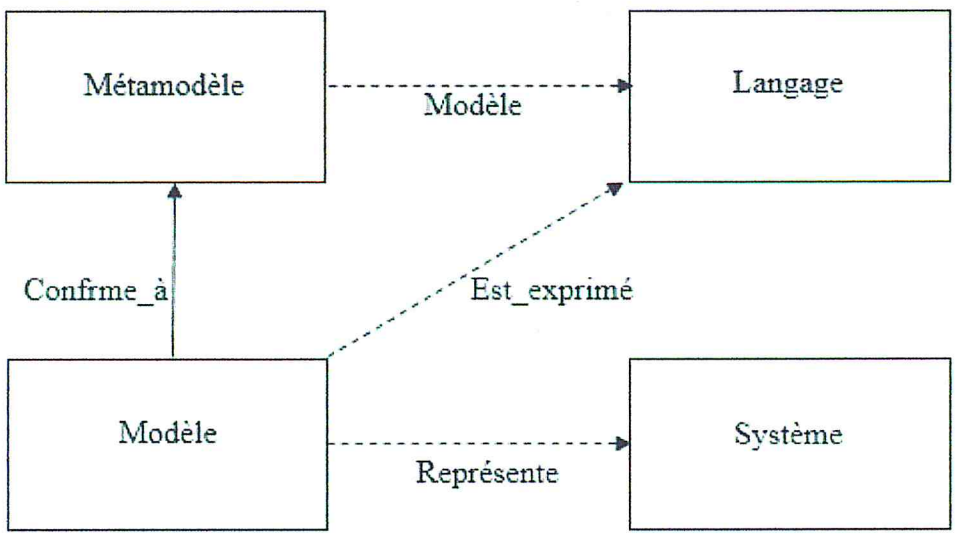



**Figure 09 :** Relations entre système, modèle, méta-modèle et langage [12]

### 2.8 Le modèle d'architecture MDA à quatre niveaux

Le modèle d'architecture MDA à quatre niveaux d'abstraction, a été défini par l'OMG comme cadre général pour l'intégration des métamodèles, en se basant sur MOF.
Dans cette architecture, les modèles à deux niveaux adjacents sont liés par une relation d'instanciation [11] :

- Le niveau M0 : Ce niveau contient des informations que l'on souhaite modéliser. Ces informations sont des données réelles, donc elles sont des instances du modèle. Il est représenté à la base de la pyramide.

- Le niveau M1 : Ce niveau représente toutes les instances d'un méta-modèle. Il décrit certains aspects du système que l'on veut étudier. Il peut contenir des modèles d'informations (PIM, PSM), des diagrammes de classes UML, un modèle conceptuel de traitement MERISE,..., etc. Le modèle doit être exprimé dans un langage qui est définit ou fourni explicitement dans le niveau M2.

- Le niveau M2 : Ce niveau représente toutes les instances d'un méta-métamodèle. Il est composé de langages de spécifications ou de modélisation des modèles d'information (Le méta-modèle). Le méta-modèle UML décrit dans le standard UML définit la structure interne des modèles UML appartenant au niveau M2.

- Le niveau M3 : Ce niveau est composé d'un langage unique pour la définition des méta-modèles (Méta-méta modèle ou MOF (Meta ObjectFacility)). Il doit être assez générique pour définir les différents langages de modélisation existants et assez précis pour exprimer les règles que chaque langage doit respecter pour pouvoir être traité automatiquement. Le MOF élément réflexif du niveau M3, définit la structure de tous les méta-modèles du niveau M2. Il est représenté au sommet de la pyramide.

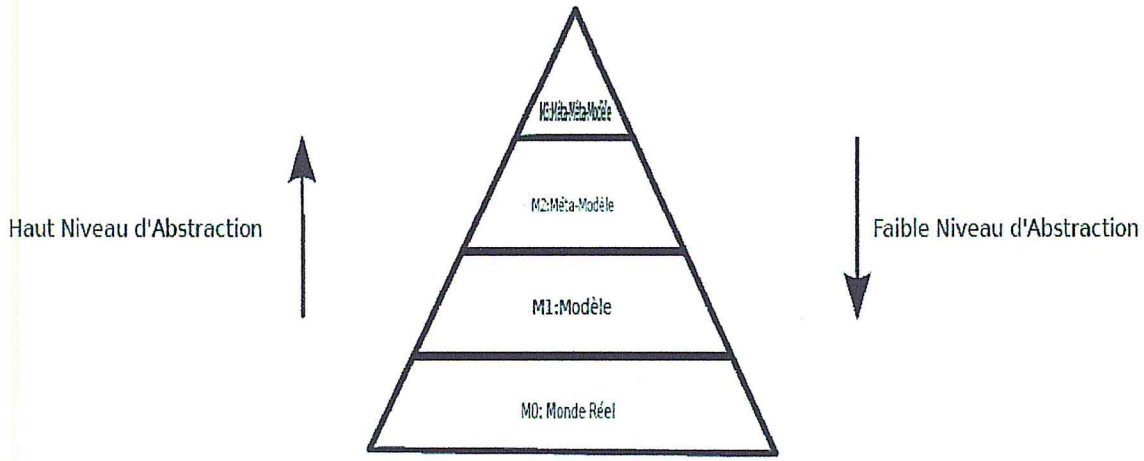



**Figure 10 :** pyramide du modèle MOF [11]

### 2.9 Meta-Object Facility (MOF)

MOF [11] se situe au sommet dans l'architecture à quatre niveaux de l'OMG (voir figure 06). MOF est un méta-formalisme c'est-à-dire un formalisme pour établir des langages de modélisation permettant eux-mêmes d'exprimer des modèles. Dans sa version 2.0, le métamodèle MOF est constitué de deux parties : EMOF (Essential MOF), pour l'élaboration des métamodèles sans association, et CMOF (Complete MOF) pour les métamodèles avec associations.

## 3. La transformation des modèles

D'après [11], la transformation de modèles est un processus qui consiste à transformer un ou plusieurs modèles sources conformément à leur métamodèle vers des modèles cibles conformément à leur métamodèle. En outre, les métamodèles source et cible peuvent être les mêmes dans certaines situations. En MDA la transformation de modèles est basée sur les métamodèles et contient deux étapes successives, qui sont :

- La spécification de règles de transformation permettant la définition de correspondance entre les concepts du métamodèle du modèle source et les concepts du métamodèle du modèle cible.

- L'application des règles de transformation permettent de passer du modèle source au modèle cible automatiquement. Un outil de transformation est nécessaire pour l'exécution de ces modèles.

Cette section introduit le standard QVT, les principes de la transformation de modèles, et les langages utilisés pour la transformation de modèles.

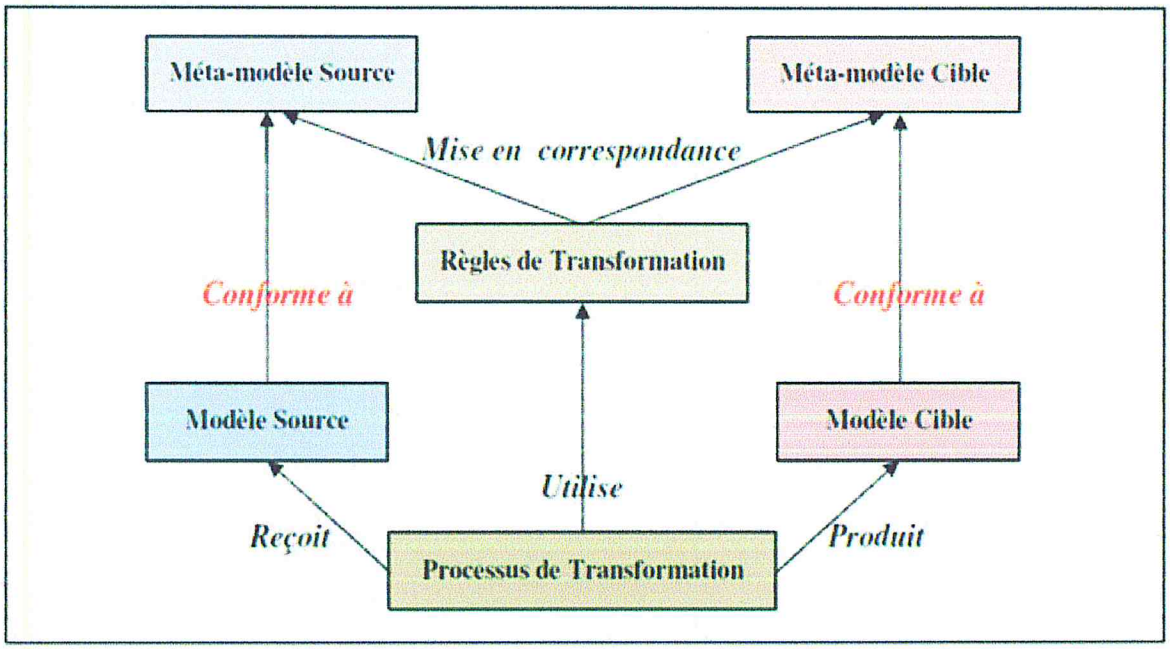



**Figure 11 :** Concepts de base de la transformation des modèles [12]

### 3.1 Objectifs de la transformation des modèles

La transformation de modèles est une opération très importante dans toute approche orientée modèle. En effet, les transformations assurent les opérations de passage d'un ou plusieurs modèles d'un niveau d'abstraction donné vers un ou plusieurs autres modèles du même niveau (transformation horizontale) ou d'un niveau différent (transformation verticale).

On peut citer comme exemple de transformation verticale, la transformation PIM vers PSM dans l'approche MDA. Ce qui a comme but de rendre les modèles opérationnels dans une approche IDM, et augmenter considérablement la productivité des applications [12].

### 3.2 La norme QVT

Les transformations de modèles étant au cœur de l'IDM, un standard dénommé QVT (Query, Views, Transformation) a été établi pour modéliser ces transformations. Ce standard définit le métamodèle permettant l'élaboration des modèles de transformation. L'appel à proposition de l'OMG d'avril 2002 pour la norme QVT visait à atteindre les objectifs suivants [12] :

- Normaliser un moyen d'exprimer des correspondances (transformations) entre langages définis avec MOF.
- Exprimer des requêtes (Query) pour filtrer et sélectionner des éléments d'un modèle (notamment sélectionner les éléments source d'une transformation).
- Proposer un mécanisme pour créer des vues (Views) qui sont des modèles déduits d'un autre pour en révéler des aspects spécifiques.
- Formaliser une manière de décrire des transformations (Transformations).

### 3.3 Object Constraint Language (OCL)

Le but du langage OCL est de permettre l'ajout de contraintes pour exprimer la sémantique statique des modèles et métamodèles. Un métamodèle n'a pas toujours l'expressivité suffisante pour décrire toutes les relations entre les méta-éléments qu'il contient.

Dans le contexte des normes de l'OMG, OCL est utilisé pour décrire des contraintes non capturées dans le métamodèle, sous forme d'invariants. L'ajout de ces contraintes est fondamental pour obtenir des métamodèles clairement définis. En outre, OCL définit des pré et post conditions sur les opérations pour que le système modélisé reste dans un état cohérent quand on exécute ces opérations. Les constructions d'OCL ne permettent pas de créer, de détruire ou modifier les objets d'un modèle : la vérification des contraintes se fait sans effet de bord.

En plus de la définition de contraintes, OCL peut être utilisé pour spécifier de manière déclarative le comportement de propriétés dérivées et d'opérations, sans toutefois pouvoir exprimer impérativement des modifications de modèles. OCL fournit donc une bonne solution pour exprimer des contraintes sur les modèles mais pas pour décrire le comportement ou la sémantique des modèles [12].

### 3.4 XML Metadata Interchange (XMI)

Les modèles étant des entités abstraites au niveau conceptuel, l'OMG a décidé de standardiser XMI qui offre une représentation concrète des modèles sous forme de documents XML. Cette représentation concrète des modèles se fait par des mécanismes appelés sérialisation et génération. La génération consiste à transformer un métamodèle en un DTD (Document Type Definition) alors que la sérialisation permet de représenter les modèles sous forme de document XML [12].

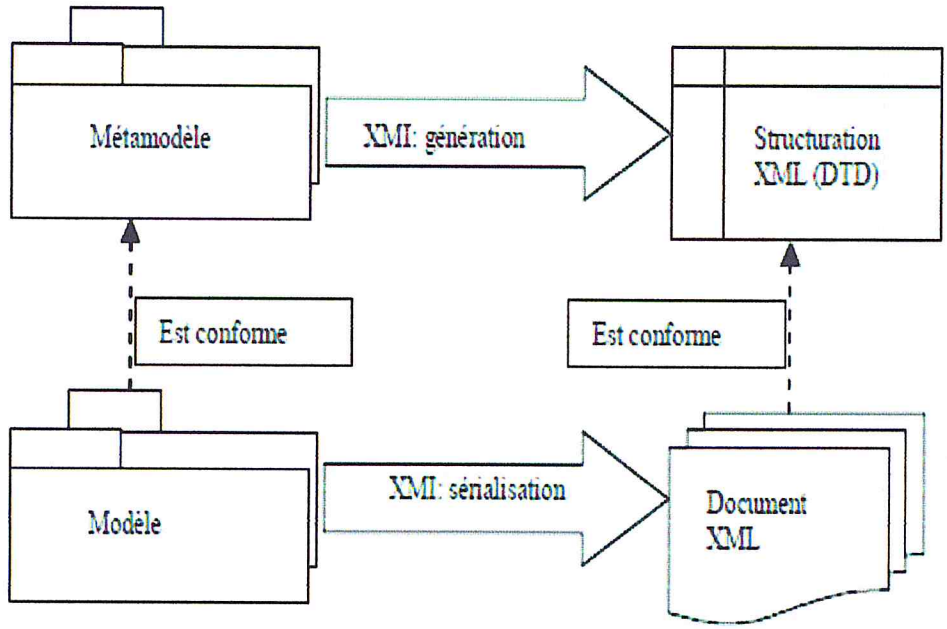



**Figure 12 :** XMI et la structuration des balises XML [12]

### 3.5 Types de transformation de modèles

Une transformation de modèles met en correspondance des éléments des modèles cible et source. On distingue les types de transformation suivants [13] :

- Une transformation simple (1 vers 1) qui associe à tout élément du modèle source au plus un élément du modèle cible. Un exemple typique de cette transformation est la transformation d'une classe UML munie de ses attributs en une table d'une base de données relationnelle.

- Une transformation multiple (M vers N) prend en entrée un ensemble d'éléments du modèle source et produit un ensemble d'éléments du modèle cible. Les transformations de décomposition de modèles (1 vers N) et de fusion de modèles (N vers 1) sont des cas particuliers de transformations multiples.

- Une transformation de mise à jour, encore appelée transformation sur place, consiste à modifier un modèle par ajout, modification ou suppression d'une partie de ses éléments. Dans ce type de transformation, les modèles source et cible sont confondus. Une telle transformation agit directement sur le modèle source sans créer de modèle cible.

### 3.6 Taxonomie des transformations de modèles

Partant de la nature des métamodèles source et cible, on distingue les transformations dites endogènes et exogènes combinées à des transformations dites verticales et horizontales. Une transformation est dite endogène si les modèles cible et source sont conformes au même métamodèle, exogène dans le cas contraire. Une transformation simple ou multiple peut être exogène ou endogène selon la nature des métamodèles source et cible impliqués dans la transformation. Par contre une transformation sur place implique un même métamodèle donc elle est endogène [13].

Le passage de PIM vers PSM ou rétro-conception est une transformation exogène et verticale alors que le raffinement est une transformation endogène et verticale. Une transformation est dite horizontale lorsque les modèles source et cible impliqués dans la transformation sont au même niveau d'abstraction.

Il est important de noter que les modèles source et cible peuvent appartenir à des espaces technologiques différents. Un exemple typique est la transformation d'un modèle de classes persistantes en un schéma de base de données relationnelle qui fait intervenir respectivement l'espace technologique de la modélisation, en général UML, et celui des bases de données. La figure 13 ci-après résume les combinaisons possibles entre transformations de modèles.

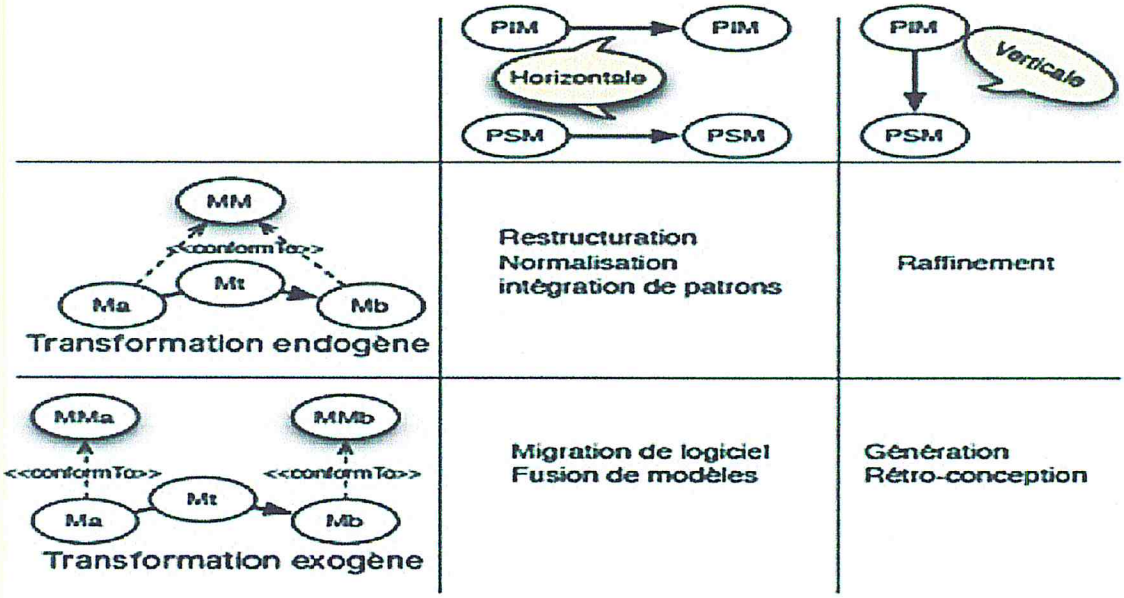



**Figure 13 :** Taxonomie des transformations de modèles [13]

### 3.7 Propriétés des transformations

Les principales propriétés qui caractérisent les transformations de modèles sont : la réversibilité, la traçabilité, la réutilisabilité, l'ordonnancement et la modularité [13].

- Réversibilité : une transformation est dite réversible si elle se fait dans les deux sens. Exemple : Model to Text et Text to Model.

- Traçabilité : la traçabilité permet de garder des informations sur le devenir des éléments des modèles au cours des différentes transformations qu'ils subissent.
  Dans un contexte d'ingénierie dirigée par les modèles, il est normal que l'information relative à la traçabilité soit considérée comme un modèle. Un modèle est donc associé à chaque exécution d'une transformation tracée. La définition d'un métamodèle de traces permet de structurer les traces qui seront générées par la plate-forme de traçabilité et ainsi de mieux les manipuler.

- Réutilisabilité : la réutilisabilité permet de réutiliser des règles de transformation dans d'autres transformations de modèles.

- Ordonnancement : l'ordonnancement consiste à représenter les niveaux d'imbrication des règles de transformation. En effet, les règles de transformations peuvent déclencher d'autres règles.

- Modularité : une transformation modulaire permet de mieux modéliser 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.

### 3.8 Les langages de transformation de modèles

Dans cette section, on retrouve les langages conçus spécifiquement pour faire de la transformation de modèles et prévus généralement pour être intégrables dans les environnements de développement standard (Eclipse par exemple).
Comme exemples des langages existants nous retrouvons BOTL (Bidirectional Object oriented Transformation Language), Kermeta, Coral (outil pour créer, éditer, et transformer des modèles à l'exécution), Viatra, QVTEclipse (une implantation préliminaire du standard QVT dans Eclipse), UMT-QVT (UML Model Transformation Tool) et le langage ATL (Atlas Transformation Language) [12].

#### 3.8.1 Kermeta

Kermeta est un langage permettant de spécifier des modèles, des modélisations et des transformations de modèles conformes à la norme MOF. Le modèle MOF orienté objet prend en charge la définition des méta-modèles en termes de structures orientées objet (packages, classes, propriétés et opérations). Il fournit également des constructions spécifiques de modèle telles que des endiguements et des associations entre les classes. Kermeta étend le MOF avec un langage d'action impératif pour spécifier les contraintes et la sémantique opérationnelle des modèles [10]. Kermeta est mis sur « Eclipse Modeling Framework (EMF) » dans l'environnement de développement Eclipse. Le langage d'action de Kermeta fournit des mécanismes pour la liaison dynamique, la réflexion et la gestion des exceptions. Il inclut également des structures classiques de contrôle telles que des blocs, des conditionnels, et des boucles [15].

### 3.8.2 Le langage QVT

QVT est un langage capable d'exprimer les requêtes, vues et transformations sur des modèles dans le contexte de l'architecture de modélisation MOF 2.0 [15].
Les trois sous-langages de QVT forment collectivement un langage de transformation hybride : avec des constructions déclaratives et impératives. Ces langages sont appelés Relations, Core et Operational Mappings. Ils sont organisés en une architecture présentée à la figure ci-dessous.

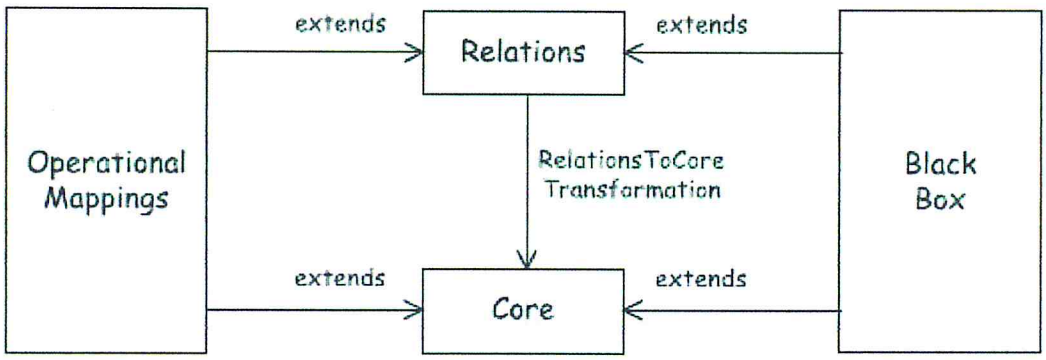



**Figure 14 :** L'architecture en couches des sous-langages de QVT [15]

Les langages Relations et Core sont tous deux déclaratifs mais placés à différents niveaux d'abstraction.

Il est parfois difficile de définir une solution complètement déclarative à un problème de transformation donné. Pour adresser cette question, QVT propose deux mécanismes pour étendre Relations et Core : un troisième langage appelé Operational Mappings et un mécanisme d'invocation de fonctionnalités de transformation implémentées dans un langage arbitraire (boîte noire ou black box) [15].

### 3.8.3 Viatra2

Viatra2 est un langage de transformation unidirectionnel principalement basé sur des techniques de transformation de graphes. Il n'est pas basé sur les métamétamodèles MOF.
Le langage opère sur des modèles exprimés à l'aide des approches de modélisation où il est possible d'avoir un nombre arbitraire de niveaux de modélisation et la relation de typage entre éléments de modèles et éléments de métamodèles est représentée explicitement [15].

### 3.8.4 ATL

- **Présentation du langage ATL**

ATL est l'acronyme d'ATLAS Transformation Language ; c'est un langage à vocation déclarative, mais qui est en réalité hybride et permet de faire des transformations de modèles aussi bien endogènes qu'exogènes. Les outils de transformation liés à ATL sont intégrés sous forme de plug-in ADT (ATL Development Tool) pour l'environnement de développement Eclipse. Un modèle de Développement logiciel à base de modèles transformation ATL se base sur des définitions de métamodèles au format XMI [16].

- **Règles de transformation en ATL**

ATL est défini par un modèle MOF pour sa syntaxe abstraite et possède une syntaxe concrète textuelle. Pour accéder aux éléments d'un modèle, ATL utilise des requêtes sous forme d'expressions OCL. Une requête permet de naviguer entre les éléments d'un modèle et d'appeler des opérations sur ceux-ci. Une règle déclarative d'ATL, appelée Matched Rule, est spécifiée par un nom, un ensemble de patrons sources (InPattern) mappés avec les éléments sources, et un ensemble de patrons cibles (OutPattern) représentant les éléments créés dans le modèle cible. Depuis la version 2006 d'ATL, de nouvelles fonctionnalités ont été ajoutées telles que l'héritage entre les règles et le multiple pattern matching (plusieurs modèles en entrée). Le style impératif d'ATL est supporté par deux constructions différentes. En effet, on peut utiliser soit des règles impératives appelées Called Rule, soit un bloc d'instructions impératives (ActionBlock) utilisé avec les deux types de règles. Une Called Rule est appelée explicitement en utilisant son nom et en initialisant ses paramètres. La figure 10 ci-dessous présente la syntaxe abstraite d'une règle de transformation ATL [16].

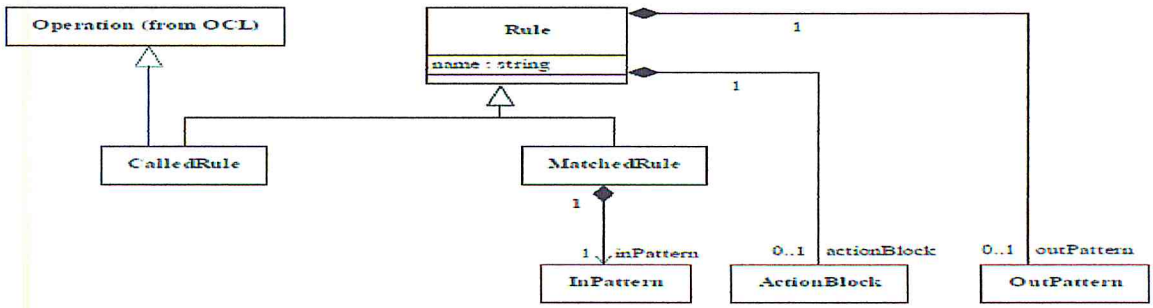



**Figure 15 :** Syntaxe abstraite d'une règle de transformation en ATL [15]

- **Structure du langage ATL :**

- En-tête :

En ATL [16], une transformation s'appelle un module. Un module contient un en-tête, un ensemble d'importation de bibliothèques de fonctions et un ensemble de fonctions et de règles de transformation. Les fonctions sont appelées helper en ATL.
L'en-tête donne le nom du module de transformation et déclare les modèles source et cible.

```
1 module SimpleClass2SimpleRDBMS;
2 create OUT : SimpleRDBMS from IN : SimpleClass;
```

**Figure 16 :** Exemple d'un en-tête en ATL [16]

L'en-tête commence par le mot-clé module suivi du nom du module. Ensuite, les modèles source et cible sont déclarés comme des variables typées par leurs métamodèles. Le mot-clé « create » indique les modèles cible. Le mot-clé « from » indique les modèles sources. Dans notre exemple, le modèle cible est représenté par le variable OUT à partir du modèle source représenté par IN.
Les modèles source et cible sont respectivement conformes aux métamodèles SimpleClass et SimpleRDBMS. En général, plus d'un modèle source et d'un modèle cible peuvent être listés dans l'en-tête.

- Helpers :

Les fonctions ATL sont appelées helpers d'après le standard OCL sur lequel ATL se base. OCL définit deux sortes de helpers : opération et attribut. En ATL, un helper peut être spécifié dans le contexte d'un type OCL (par exemple String ou Integer) ou d'un type source (venant de l'un des métamodèles source). Les modèles cibles ne sont en effet pas navigables.

Les helpers opération peuvent être utilisés pour définir des opérations dans le contexte d'un élément de modèle ou du module de transformation. Le rôle principal des helpers opération est de réaliser la navigation des modèles source. Ils peuvent avoir des paramètres et peuvent utiliser la récursivité. Les helpers opération définis dans le contexte d'éléments de modèles permettent les appels polymorphiques. Puisque la navigation n'est autorisée que sur les modèles

sources en lecture seule, une opération retourne toujours la même valeur pour un contexte et un ensemble d'arguments donnés [16].

```
1 helper context SimpleClass!Class def: allAttributes : Sequence(SimpleClass!Attribute) =
2   self.attrs->union(
3     if not self.parent.oclIsUndefined() then
4       self.parent.allAttributes->select(attr |
5         not self.attrs->exists(at | at.name = attr.name)
6       )
7     else Sequence {}
8     endif
9   )->flatten();
```

**Figure 17 :** Exemple d'un Helper en ATL [16]

- Règles déclaratives (Matched rules)

Une matched rule est composée d'un motif source et d'un motif cible. Le motif source d'une règle déginit un ensemble de types source (venant des métamodèles source) et une garde sous la forme d'une expression OCL booléenne. Un motif source est évalué en un ensemble de tuples dans les modèles source [16].

Dans la figure 13 on remarque que le nom de la règle (PersistentClass2Table) est donné après le mot-clé rule (ligne 1). Le motif d'entrée définit une variable de type SimpleClass!Class (ligne 3). La garde (ligne 4) spécifie que seules les classes persistantes sans class mère sont retenues. Le motif cible contient un élément de type SimpleRDBMS!Table (lignes 7-9). Cet élément a une liaison (ligne 8) qui définit une expression utilisée pour initialiser l'attribut name (le nom de la table).

Le symbole <- est utilisé pour délimiter la propriété à initialiser à gauche de l'expression servant à l'initialiser à droite.

```
1  rule PersistentClass2Table{
2    from
3      c : SimpleClass!Class (
4        c.is_persistent and c.parent.oclIsUndefined()
5      )
6    to
7      t : SimpleRDBMS!Table (
8            name <- c.name
9      )
10 }
```

**Figure 18 :** Exemple d'une règle déclarative en ATL [16]

### 3.9 Exemple existant de transformation de modèles pour la génération automatique d'architecture

- **M2Arch**

M2Arch [9] est un processus basé sur un modèle pour affiner les modèles de mission en descriptions d'architecture. M2Arch est préoccupé par la génération automatique des descriptions d'architecture dans SosADL, depuis mKAOS. M2Arch est également livré avec un outil associé prenant en charge à la fois la modélisation de mission et la description de l'architecture ainsi que la validation et la simulation des architectures résultantes.

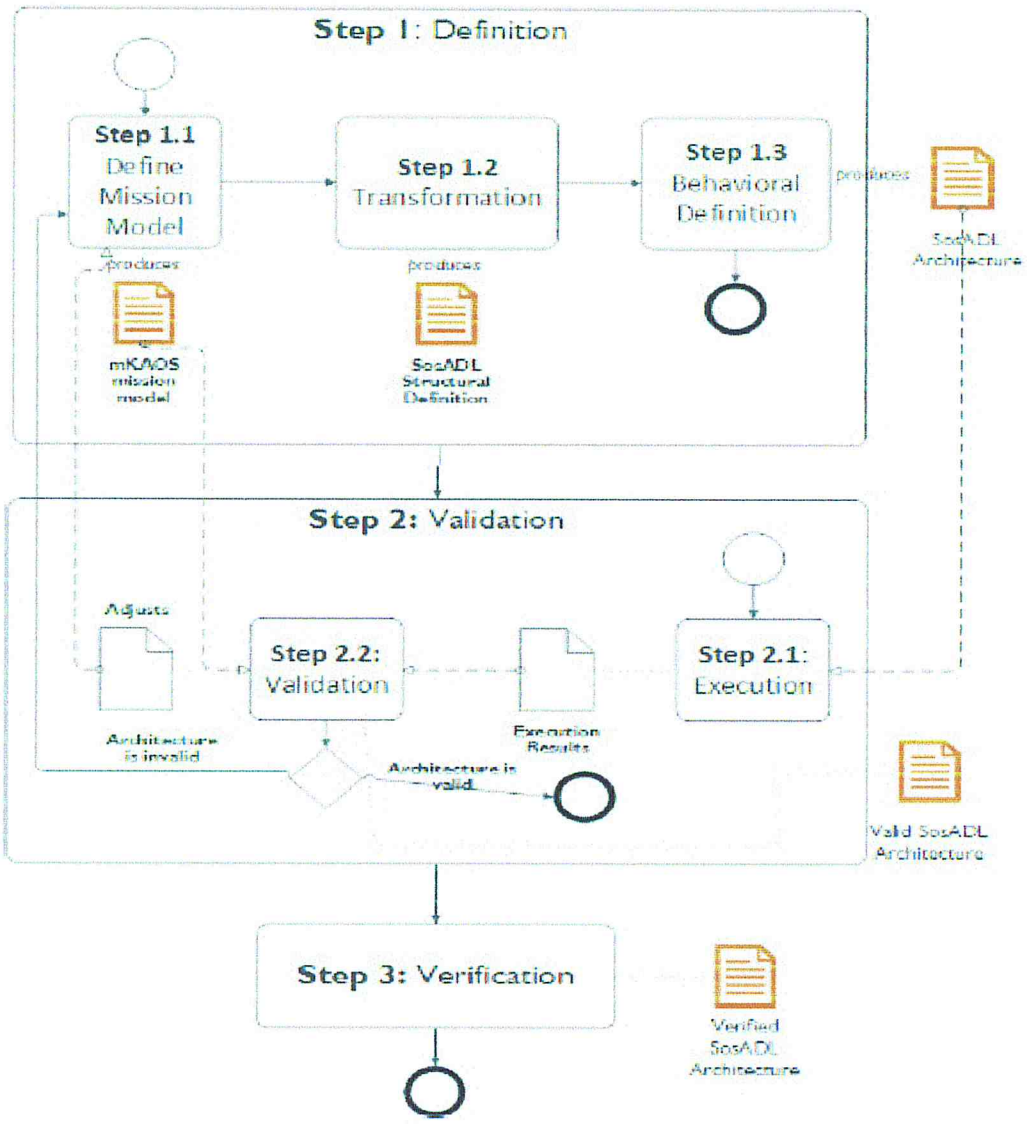



**Figure 19 :** vue d'ensemble du processus M2Arch [9]

## 4. Conclusion

Dans ce chapitre, nous nous sommes intéressés à l'ingénierie dirigée par les modèles, ses bases et ses objectifs, on a détaillé les principes de la modélisation, la métamodélisation et la transformation des modèles. Cette dernière fait l'objet de notre travail. A la fin du chapitre, on s'est concentré sur les langages de transformation de modèles, spécifiquement le langage ATL, qu'on va utiliser pour notre transformation implémentée dans la partie II, pour raison de sa syntaxe textuelle concrète, les mappages simples qui peuvent être exprimés simplement et aussi pour la correspondance claire et facilement gérable, entre les éléments du modèle source et la navigation, et ceci est pour but de créer et initialiser les éléments des modèles cibles.

Dans le chapitre suivant, nous allons présenter les deux langages impliqués dans la transformation, le modèle source et le modèle cible.

# Chapitre 3

## *SOURCE ET CIBLE DE LA TRANSFORMATION*

## 1. Introduction

Après avoir abordé la spécification des missions et la transformation des modèles dans les chapitres précédents, nous allons présenter dans ce chapitre les langages source et cible de la transformation prévue qui fait l'objet de ce travail.

## 2. Le langage SysML

SysML est un langage de modélisation graphique développé par l'OMG et l'INCOSE. La majorité des concepts présents dans SysML sont issus d'UML. Les concepteurs de ce langage insistent sur le fait que SysML tout comme UML n'est pas une méthode (description des étapes de mise en œuvre) de modélisation mais bien un langage de modélisation graphique. Il est composé de huit graphes.

Les différents diagrammes de SysML sont :

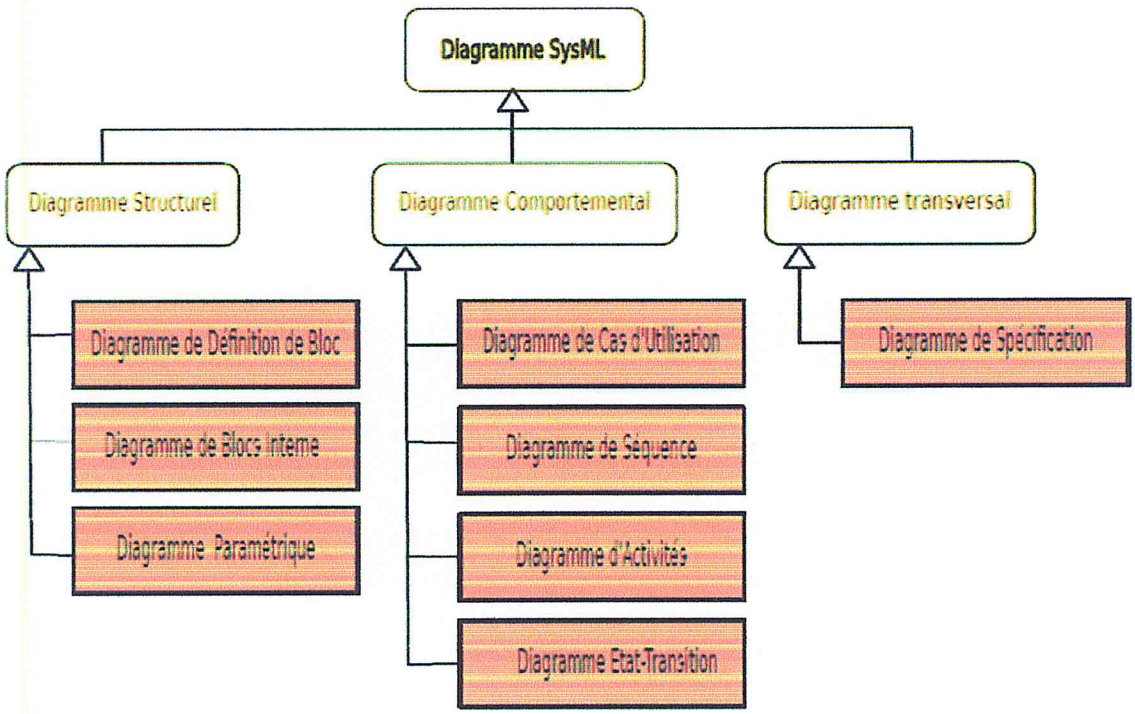



**Figure 20 :** Les diagrammes de SysML [17]

Dans ce qui suit, on va présenter les deux diagrammes de SysML dont on a besoin dans la transformation à savoir : le diagramme d'activité et le diagramme de définition de blocs.

### 2.1 Le diagramme d'activité SysML

Les diagrammes d'activité de SysML 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 [18].

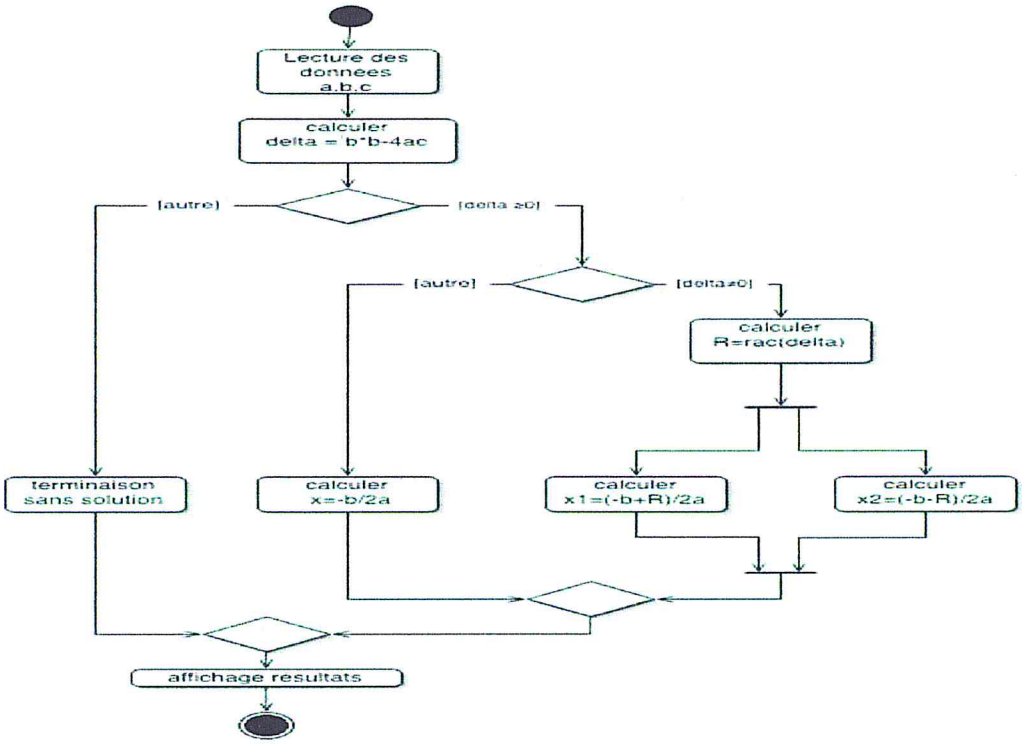



**Figure 21 :** Exemple d'un diagramme d'activité [18]

#### 2.1.1 Une activité

Une activité est la spécification du comportement paramétré par un séquencement organisé d'unités subordonnées, dont les éléments simples sont les actions. Le flux de contrôle reste dans l'activité jusqu'à ce que les traitements soient terminés, et le flot d'exécution est modélisé par des nœuds reliés par des arcs.

Une activité est un comportement, et à ce titre peut être associée à des paramètres. Une activité regroupant des nœuds et des arcs est appelée un groupe d'activités [18].

### 2.1.2 Intérêts des diagrammes d'activité [19]

- Représenter graphiquement le comportement interne d'une opération, d'une classe ou d'un cas d'utilisation sous forme d'une suite d'actions.
- Utiliser le mécanisme de synchronisation pour représenter les successions d'états synchrones, alors que les diagrammes d'états-transitions sont utilisés principalement pour représenter les suites d'états asynchrones.
- Utiliser des transitions automatiques évite la nécessité d'existence d'évènement de transition pour avoir un changement d'états.
- Modéliser un workflow dans un cas d'utilisation, ou entre plusieurs cas d'utilisations.
- Définir avec précision les traitements qui ont cours au sein du système, Certains algorithmes ou calculs nécessitent de la part du modélisateur une description poussée.
- Spécifier une opération (décrire la logique d'une opération).

### 2.1.3 Composition d'un diagramme d'activité

Ci-dessous, un métamodèle partiel représentant les éléments essentiels du diagramme d'activité du SysML :

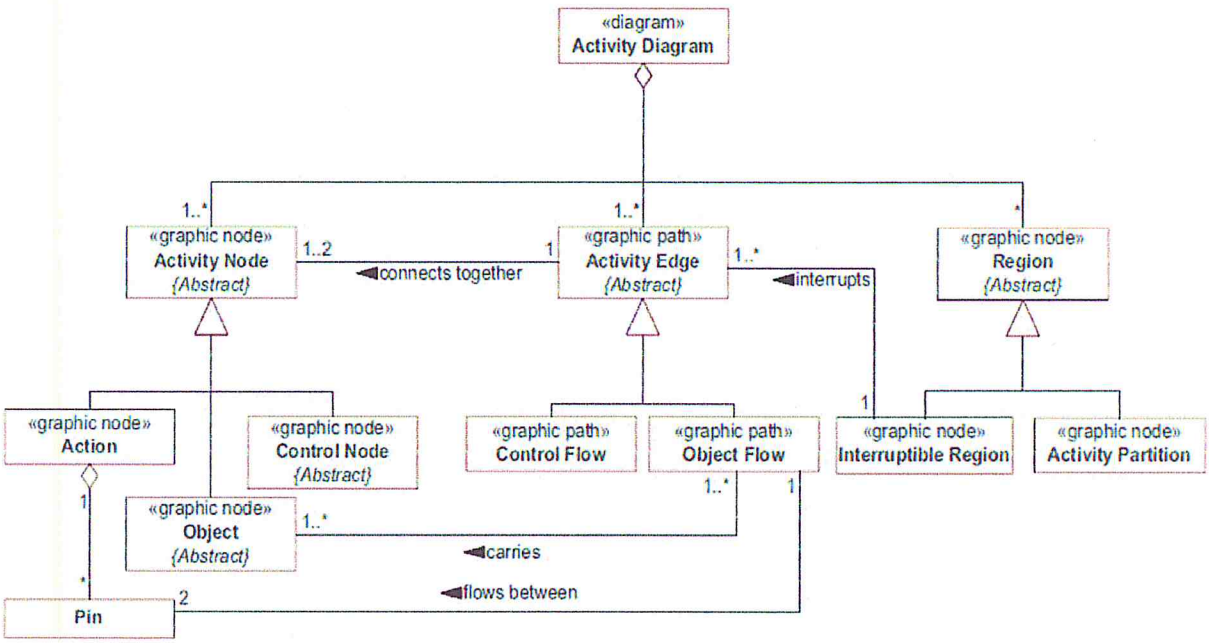



**Figure 22 :** Un métamodèle partiel du diagramme d'activité SysML [18]

- **Les nœuds (nodes)**

**A. Nœud d'activité (activity node)**

D'après [19], un nœud d'activité est une classe abstraite permettant de représenter les étapes le long du flux d'une activité. Un nœud d'activité peut être l'exécution d'un comportement subordonné, comme un calcul arithmétique, un appel à une opération, ou la manipulation du contenu d'un objet. Les nœuds d'activité comprennent également le flux de contrôle des constructions, tel que la synchronisation, la décision et la concurrence.

Il existe trois types de nœuds d'activités :

- Les nœuds d'exécutions (executable node).
- Les nœuds objets (object node).
- Les nœud de contrôle (control nodes).

La figure ci-dessus représente graphiquement les nœuds d'activité.

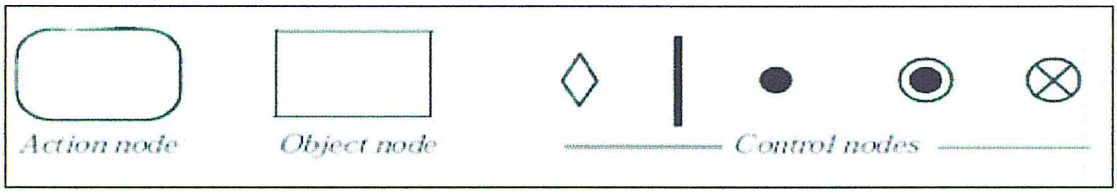



**Figure 23 :** Notation nœuds d'activité [19]

- **Nœud d'objet (object node)**

Un nœud d'objet est une méta-classe abstraite permettant de définir les flux d'objets dans les diagrammes d'activité. Il représente l'existence d'un objet généré par une action dans une activité et utilisé par d'autres actions [19].

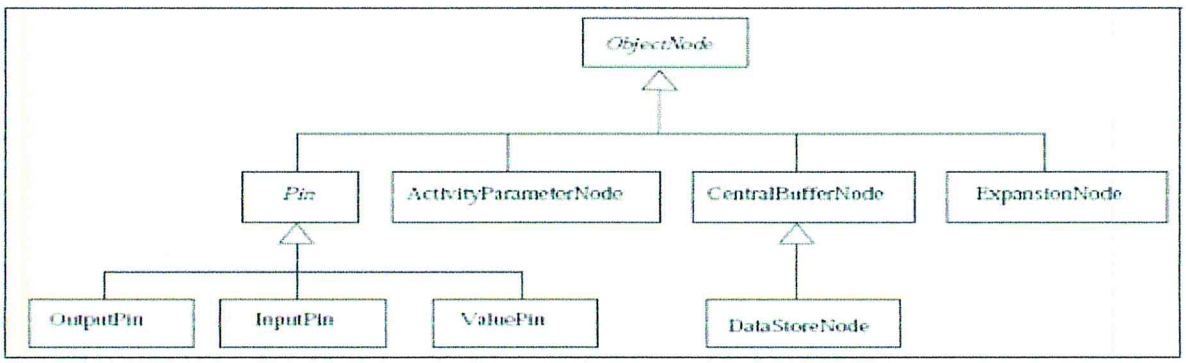



**Figure 24 :** Arbre de spécialisation du nœud d'objet [19]

Un nœud d'objet est noté par un rectangle contenant le nom du nœud. Le nœud d'objet avec un signal comme type, est affiché par le symbole à droite de la figure 25.

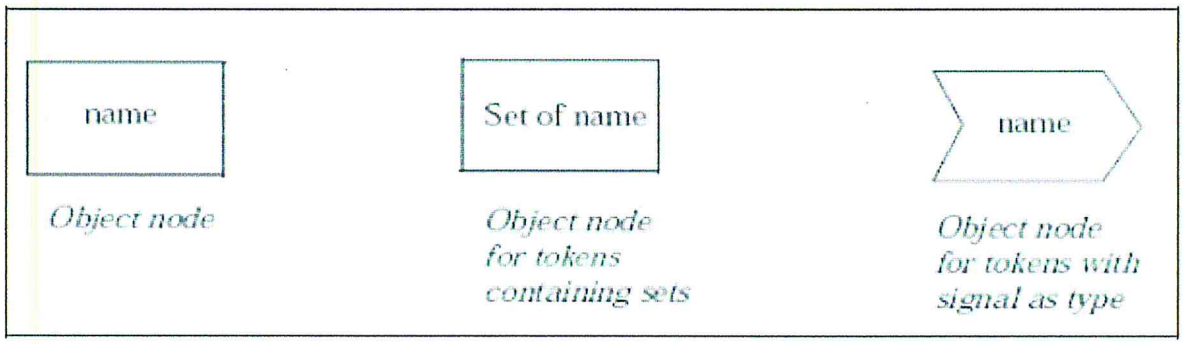



**Figure 25 :** Notation nœud d'objet [19]

- **Broche (pin)**

Une broche (Pin) [19] est un nœud objet connecté en entrée ou en sortie d'une activité. L'activité ne peut débuter qu'après l'affectation d'une valeur à chacun de ses pins d'entrée. Quand l'activité se termine (après une modification des valeurs d'entrée), une valeur doit être affectée à chacun de ses pins de sortie. Les traitements ne sont visibles qu'à l'intérieur de l'activité. La figure ci-dessous présente graphiquement les deux types de pin : pin d'entrée et pin de sortie.

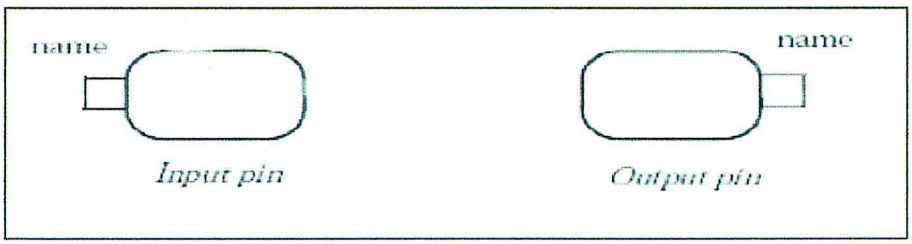



**Figure 26 :** Notation pin [19]

Conceptuellement, la notation de nœud objet ne devrait pas avoir d'instances dans les modèles (le nœud d'objet est une classe abstraite), La figure 21 donne deux représentations équivalentes de flux d'objets entre deux actions. La première représentation utilise des pins, alors que la deuxième utilise la notation d'un nœud objet.

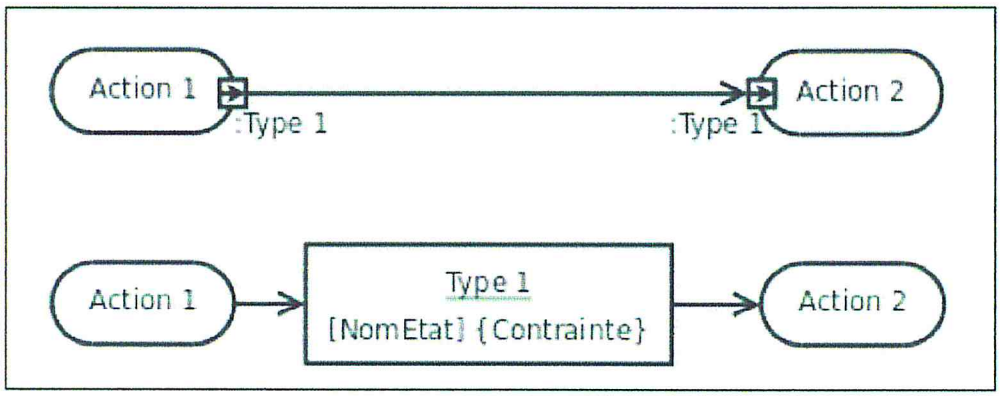



**Figure 27 :** Deux représentations équivalentes pour représenter un flux d'objet [19]

➢ **Nœud paramètre d'activité (activity parameter node)**

C'est l'un des nœuds objet, il décrit les entrées ou les sorties des activités. Il est toujours associé avec un paramètre de l'activité [19].

➢ **Nœud central de mémoire tampon (central buffer node)**

Un nœud central de mémoire tampon est un nœud d'objet, destiné pour la gestion des flux provenant de multiples sources. Il peut avoir plusieurs arcs entrants et plusieurs arcs sortants. Graphiquement, on utilise le mot clé < centralBuffer > associé à la notation d'un nœud objet [19].

➢ **Nœud d'expansion (expansion node)**

Un nœud d'expansion ou expansion node est un nœud d'objet qui peut être utilisé pour indiquer un flux à travers les limites d'une région d'expansion [19].

- **Nœud de contrôle (control node)**

Un nœud de contrôle est un nœud d'activités abstrait utilisé pour coordonner les flux entre les nœuds d'une activité. La figure ci-dessous présente l'arbre de spécialisation des nœuds de contrôle [19].

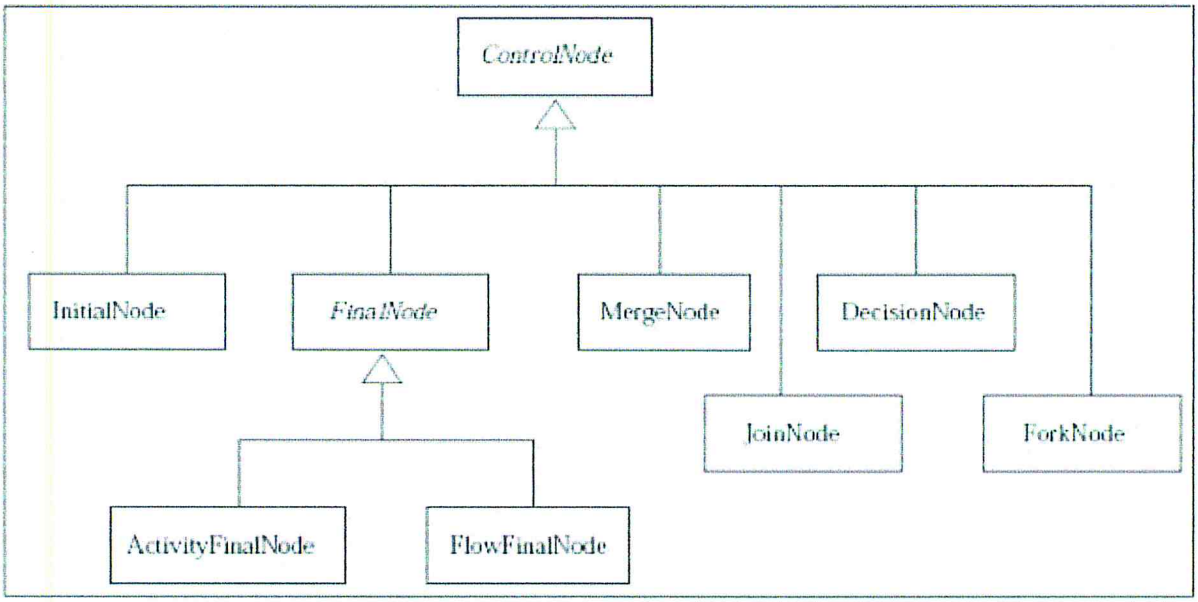



**Figure 28 :** Arbre de spécialisation des nœuds de contrôle [20]

Graphiquement, les nœuds de contrôle sont présentés comme suit :

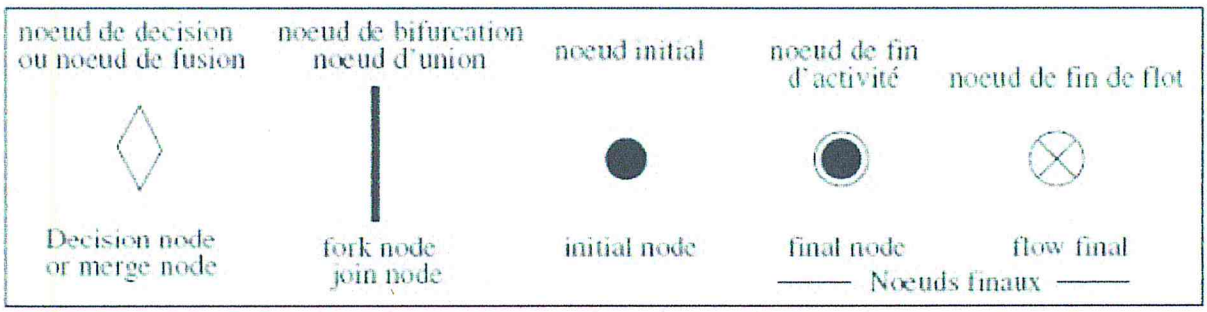



**Figure 29 :** Représentation graphique des nœuds de contrôles [20]

- **Nœud initial (initial node)**

Un nœud initial est un nœud de contrôle à partir duquel le flot débute. Il possède un arc sortant et pas d'arc entrant. Dans une activité on peut avoir plusieurs nœuds initiaux [19].

- **Nœud final (final node)**

Un nœud final est un nœud de contrôle dans lequel le flux d'activité s'arrête. Un nœud final peut avoir un ou plusieurs arcs entrants et aucun arc sortant. On peut distinguer deux types de nœuds finaux [19] :

- **Les nœuds finaux d'activité (activity final node)**

Dans un nœud final d'activité, Lorsque l'un de ses arcs entrants est activé, l'exécution de l'activité en cour s'achève, et tout nœud ou flux actif au sein de cette activité est abandonné [19].

- **Les nœuds finaux de flux (flow final node)**

L'arrivé du flux d'exécution à un nœud final de flux, implique la terminaison du flux de ce dernier. Mais cette fin n'a aucun effet sur les autres flux actifs de l'activité [19].

➢ **Nœud de fusion ou interclassement (merge node)**

Un nœud de fusion est un nœud de contrôle, il rassemble plusieurs flots alternatifs entrants en un seul flot sortant. L'utilité de ce nœud n'est pas pour synchroniser des flux concurrents mais pour accepter un flux (en sortie) parmi plusieurs flux entrants [19].

➢ **Nœud de décision (decision node)**

Un nœud de décision est un nœud de contrôle, il permet de faire un choix entre plusieurs flux sortants. Les flux sortants sont sélectionnés en fonction de la condition de garde qui est associée à chaque arc sortant. (Possibilité d'existence du problème du choix indéterministe) [19].

Si aucun arc en sortie n'est franchissable, le modèle est mal formé, et l'utilisation d'une garde [else] est recommandée. La notation du nœud de décision est présentée par la figure 30.

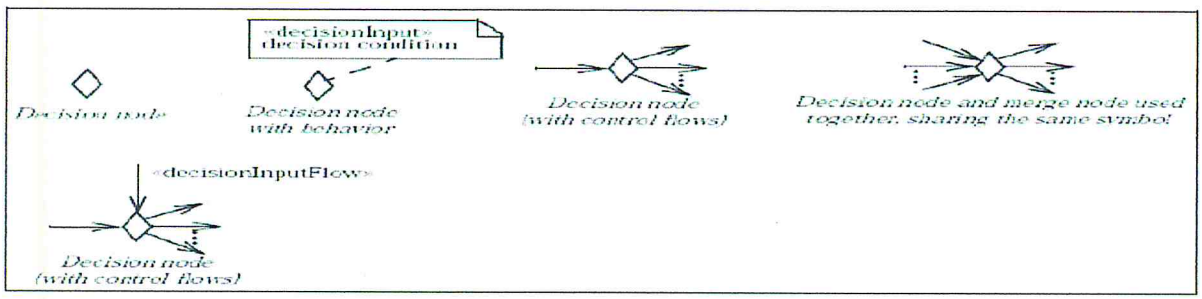



**Figure 30 :** Notation nœud de décision [19]

➢ **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 [19].

➢ **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 [19].

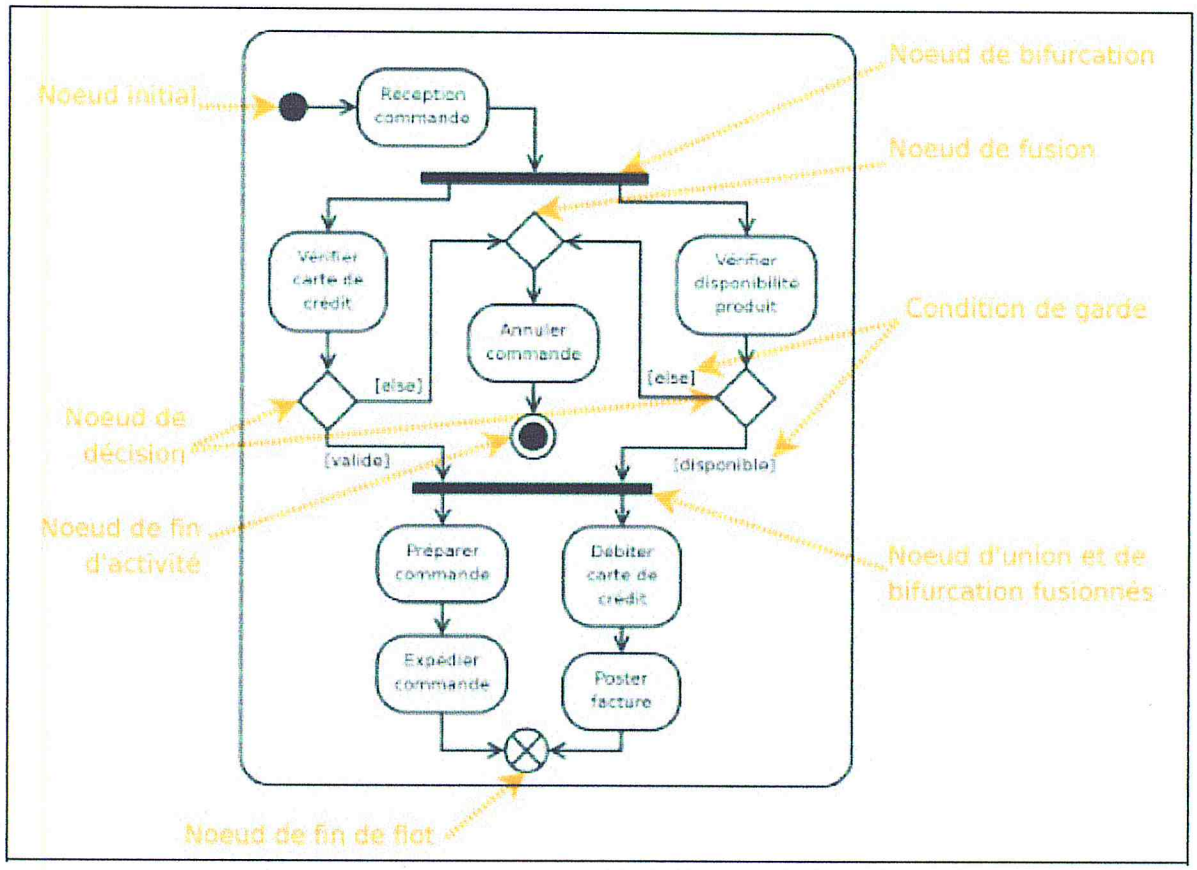



**Figure 31 :** Exemple illustre l'utilisation des nœuds de contrôle

- **Nœud exécutable (executable node)**

Un nœud exécutable est une classe abstraite pour les nœuds d'activité qui peuvent être exécutés. Il possède un gestionnaire d'exception qui peut capturer les exceptions levées par le nœud, ou par l'un de ses nœuds imbriqués [19].

**B. Une partition d'activité (activity partition)**

Les partitions d'activité appelées aussi couloirs ou lignes d'eau (swimlane), divisent l'espace des nœuds et des arcs afin de montrer explicitement l'entité dans quelle les actions peuvent être effectuées. Cette division permet de faire des regroupements dans les diagrammes

d'activité. Une partition peut être décomposée en sous-partitions et regrouper d'autres partitions selon une autre dimension.

Les partitions correspondent souvent à des unités organisationnelles dans un modèle de business, comme le montre l'exemple de la figure suivante [19].

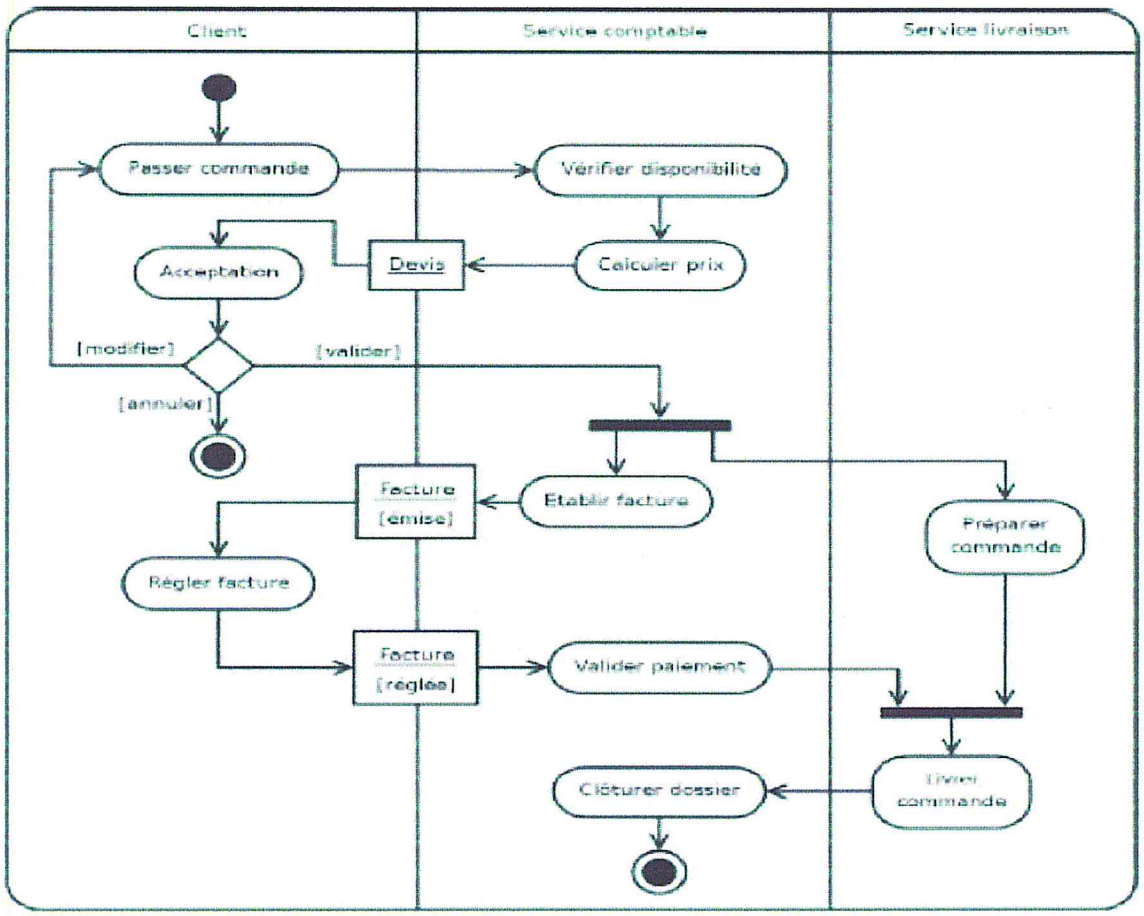



**Figure 32 :** Exemple illustratif (partitions d'activités) [19]

### C. Une région d'activité interruptible (interruptible activity region)

Une région interruptible d'activité est un groupe d'activité (regroupement de nœuds et d'arcs), pouvant contenir un arc jouant le rôle d'interrupteur pour cette région. L'activation de l'arc interrupteur implique l'arrêt de l'ensemble des flux dans la région [19].

### D. Une région d'expansion (expansion region)

Une région d'expansion est une région strictement emboitée dans une activité avec des entrées et des sorties sous forme de nœuds d'expansion. Ces derniers représentent une collection

d'éléments. La région d'expansion est exécutée pour chaque élément de la collection d'entrée. Dans une région d'expansion, les sorties sont aussi modélisées par des nœuds d'expansion [19].

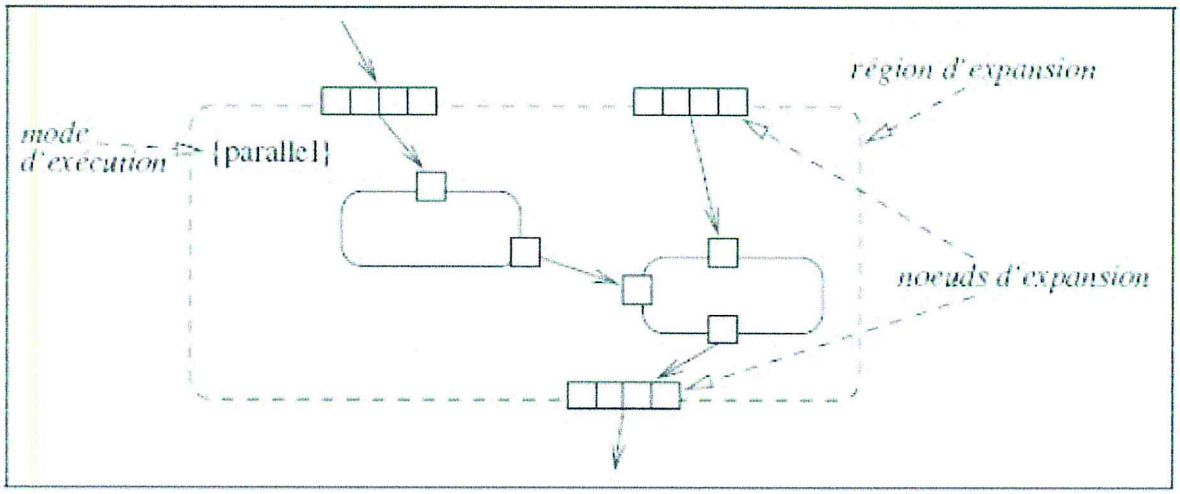



**Figure 33 :** Région d'expansion [19]

**E. Une précondition ou post-condition locale**

Une précondition est un ensemble facultatif de contraintes, en précisant ce qui doit être rempli lorsque le comportement est invoqué. Alors qu'une post-condition est un ensemble de facultatif de contraintes, en précisant ce qui doit être accompli après la fin de l'exécution du comportement [19].

**F. Un ensemble de paramètres (parameter set)**

Un ensemble de paramètres est définit comme un élément qui fournit des ensembles alternatifs d'entrées et de sorties nécessaires à un comportement. Chaque ensemble est exclusif des autres ensembles de paramètres du comportement [19].

- **Les arcs (edges)**

- **Arc d'activité (ActivityEdge)**

Un arc d'activité est une connexion dirigée entre deux nœuds d'activité. Si l'arc a un nom, il est noté près de la flèche [19].

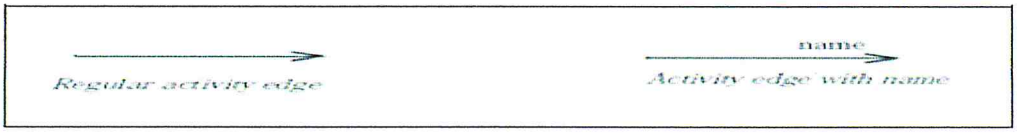



**Figure 34 :** Arc d'activité [19]

### A. Flux de contrôle (control flow)

Un flux de contrôle est un arc qui permet de décrire le séquencement de deux nœuds d'activité (un flux de contrôle démarre un nœud d'activité, après la terminaison d'une activité précédente). Il ne transmet pas des données [19].

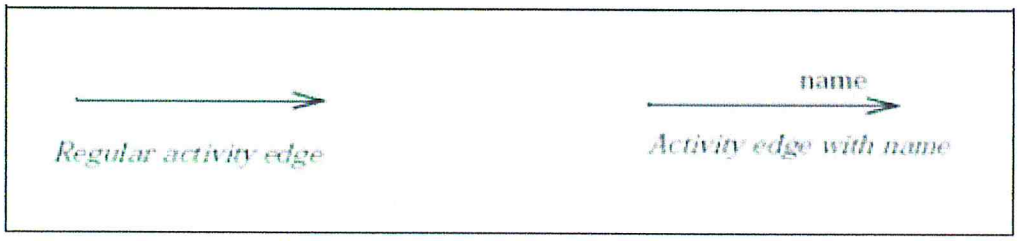



**Figure 35 :** Notation flux de contrôle [19]

### B. Flux d'objet (object flow)

Un arc de flux d'objets est un arc qui permet de transmettre des données entre des nœuds d'objet [19].

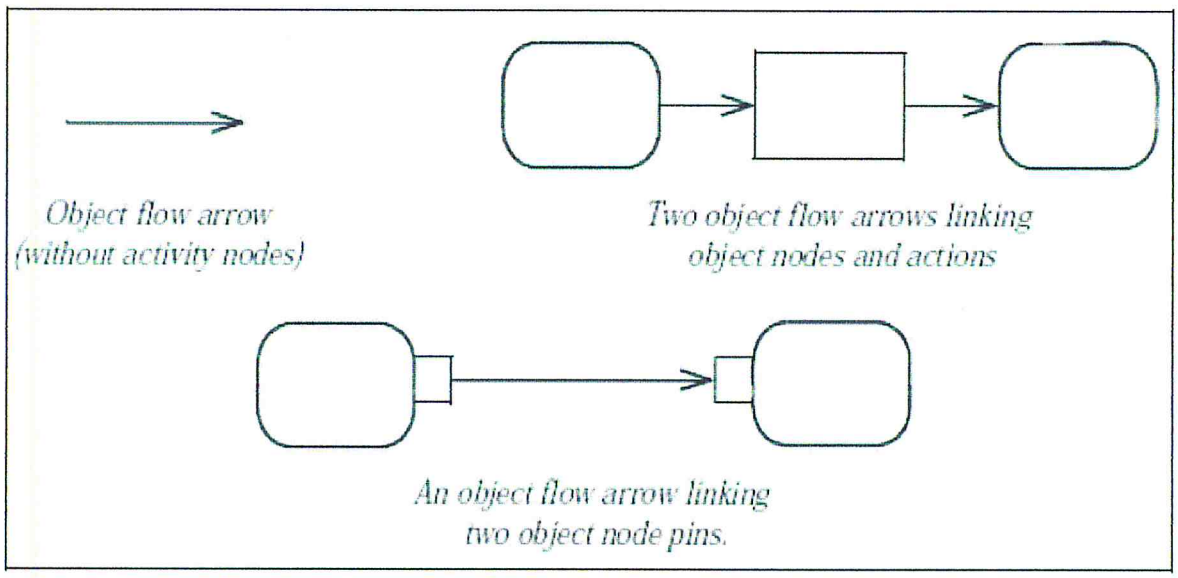



**Figure 36 :** Notation flux d'objet [19]

## 2.2 Diagramme de définition de blocs

Le Diagramme de définition de blocs [20] est un diagramme qui permet de définir le système et sa hiérarchie. La sémantique utilisée est très proche du diagramme de classe d'UML et d'un organigramme technique. Un bloc peut représenter des sous-systèmes (composants)

organiques ou fonctionnels. Des liaisons définissent les relations entre les blocs. L'entité spécification de flux permet de spécifier la nature d'un flux pour un port.

La figure 37 présente un exemple de diagramme de définition de blocs avec un bloc propulsion qui contient un moteur et un réducteur. Une spécification du flux énergie de rotation est définie et des ports sont déclarés en sortie du moteur et en entrée du réducteur. La cardinalité au niveau des liaisons (1…*) indique que plusieurs moteurs peuvent être associé à plusieurs blocs propulsion et de même pour le réducteur. Cette description graphique permet donc de définir les composants d'un système, d'y spécifier leur entrées/sorties et de préciser leur lien de composition dans le système.

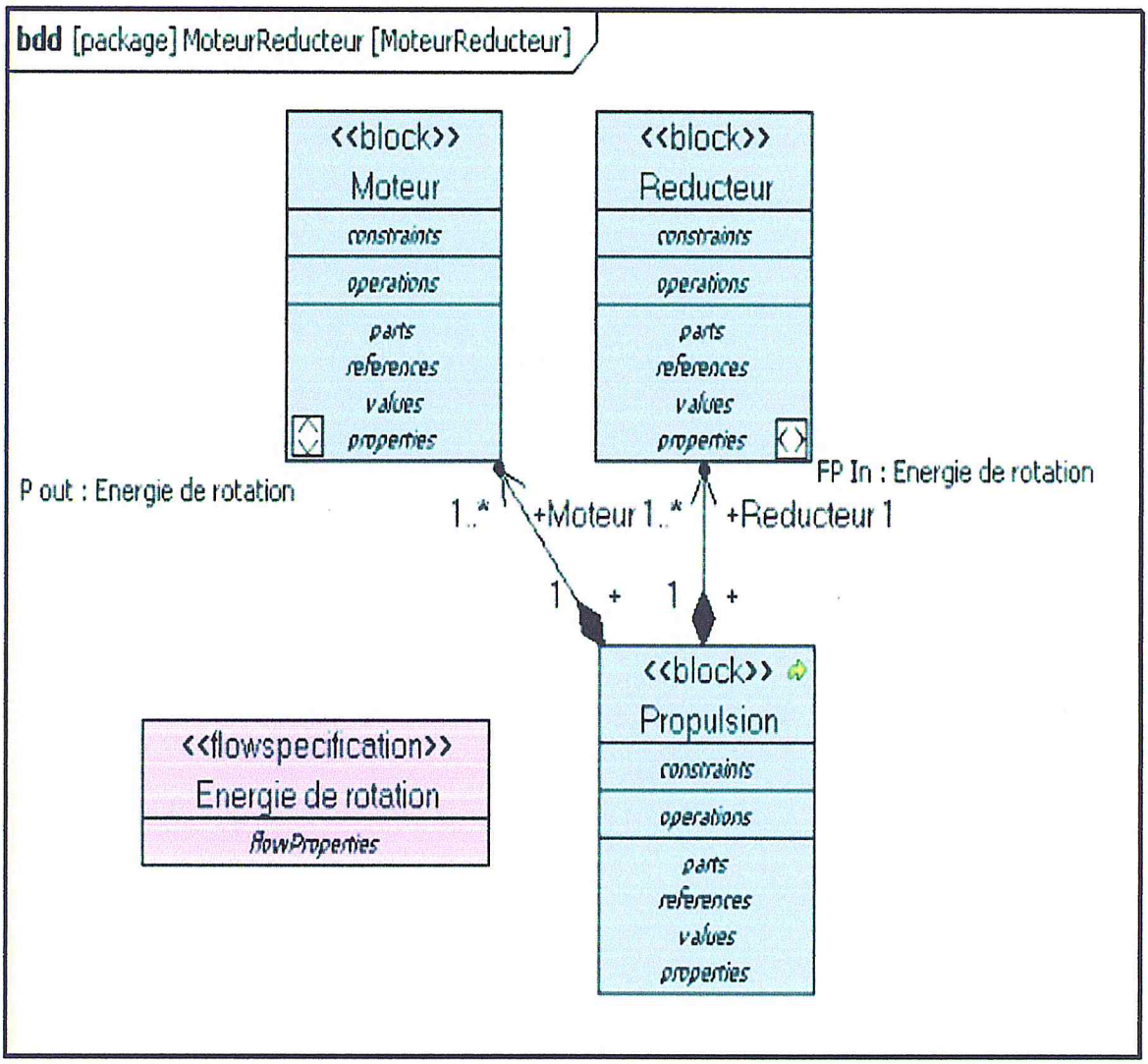



**Figure 37 :** Exemple d'un diagramme de définition de blocs [20]

### 2.2.1 Intérêts du diagramme de définition de blocs

Le but principal du diagramme de définition de bloc, est la clarté et la simplicité. Les diagrammes de définition de bloc devraient pouvoir être lus facilement et ils devraient avoir un sens. Un diagramme qui est difficile à lire peut simplement indiquer qu'il y a trop sur lui et qu'il doit être divisé en un certain nombre d'autres diagrammes. Il peut également être une indication que la modélisation n'est pas correcte et qu'elle doit être revisitée. Une autre possibilité est que le diagramme révèle une complexité fondamentale inhérente au système, à partir de laquelle des leçons peuvent être apprises [20].

### 2.2.2 Composition du diagramme de définition de blocs

Les diagrammes de définition de bloc sont constitués de deux éléments de base : les blocs et les relations.

Les deux blocs et les relations peuvent avoir différents types et ont une syntaxe plus détaillée qui peut être utilisé pour ajouter plus d'informations à leur sujet. Cependant, au plus haut niveau d'abstraction, il n'y a que les deux éléments très simples qui doivent exister dans le diagramme. Un diagramme de définition de bloc peut également contenir différents types de ports et d'interfaces, ainsi que des flux d'éléments, mais à leur plus simple, il suffit de contenir des blocs et des relations. Les blocs décrivent les types de choses qui existent dans un système, tandis que les relations décrivent les relations entre les différents blocs [20].

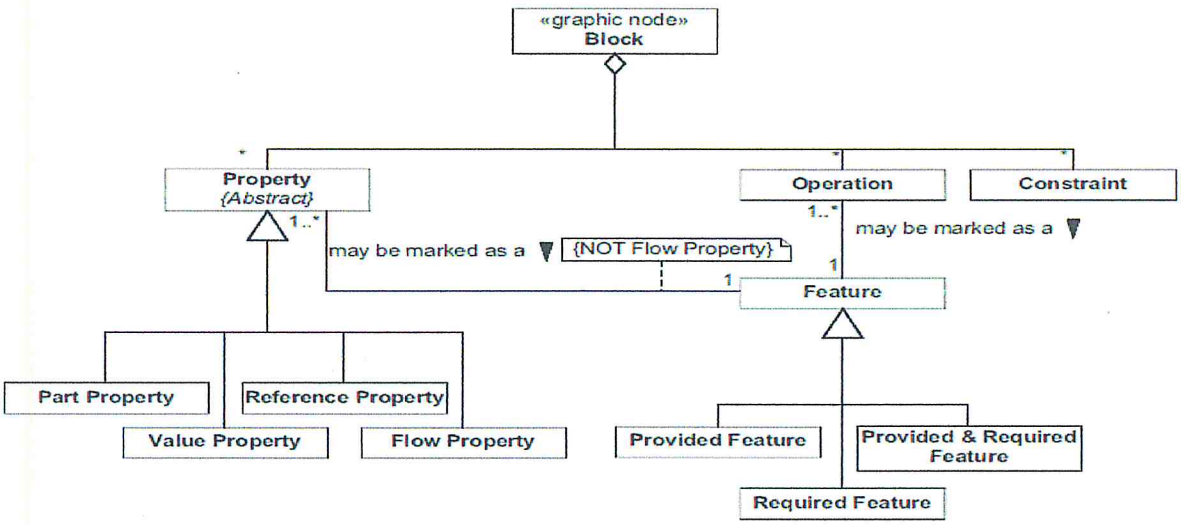



**Figure 38 :** un métamodèle partiel du diagramme de définition de blocs montrant les blocs [20]

- **Les blocs :**

Un bloc [21] a des points d'interaction définis par zéro ou plus de ports. Chaque 'port' est tapé par un block et peut être imbriqué avec zéro ou plusieurs autres ports. Un «port» peut être plus spécialisé dans deux sous-types principaux :

- 'Full port', utilisé pour représenter un point d'interaction qui est un élément distinct du modèle. C'est-à-dire qu'un port complet peut avoir ses propres parties internes et son comportement.
- 'Port proxy', utilisé pour représenter un point d'interaction qui identifie les fonctionnalités de son bloc propriétaire qui sont disponibles pour d'autres blocs externes. Ils ne constituent pas un élément distinct du modèle et ne précisent donc pas leurs propres parties et comportements internes. Ces caractéristiques et le comportement qu'ils rendent disponibles sont en fait ceux de son bloc propriétaire. Un 'Port proxy' ne doit être tapé que par un 'bloc d'interface'.

Chaque «bloc» est composé de zéro ou plus de «propriétés», zéro ou plus «opérations» et zéro ou plus «contraintes», comme illustré à la figure 33.

Le diagramme de la figure 33 montre le méta-modèle partiel pour le diagramme de définition de bloc montrant les éléments d'un bloc. Il existe quatre types de propriétés :

- 'Part Property', qui appartient au bloc. C'est-à-dire, une propriété intrinsèque au bloc, mais qui aura sa propre identité. Une propriété de pièce peut être entièrement appartenant à son bloc parent ou peuvent être partagés entre plusieurs blocs parents.
- 'Reference Property', qui est référencée par le bloc, mais qui ne lui appartient pas.
- 'Value Property', qui représente une propriété qui ne peut pas être identifiée excepté par la valeur elle-même, par exemple des nombres ou des couleurs.
- 'Flow Property', qui définit les éléments qui peuvent s'écouler vers ou depuis (ou les deux) un bloc. Ils sont principalement utilisés pour définir les éléments qui peuvent circuler dans et hors des ports et tous les flux d'éléments qui circulent entre les ports sont tapés par les propriétés de flux.

Une opération est une propriété (à l'exception d'une «propriété de flux») qui peut être marquée comme étant une fonctionnalité « Feature ». Une fonctionnalité est une propriété ou une opération qu'un bloc prend en charge pour d'autres blocs à utiliser une fonction fournie

« Provided Feature » ou qu'il exige d'autres blocs pour prendre en charge pour son propre usage une fonctionnalité requise « Required Feature », ou les deux.

- **Les relations :**

Il existe trois types principaux de «relation», comme illustré à la figure 34 :

- 'Association', qui définit une relation simple entre un ou plusieurs blocs. Il existe également deux spécialisations de l'association, appelées «agrégation» et «composition», qui montrent les parties partagées et les pièces appartenant respectivement.

- «Generalisation», qui montre une relation utilisée pour afficher les blocs parents et enfants.

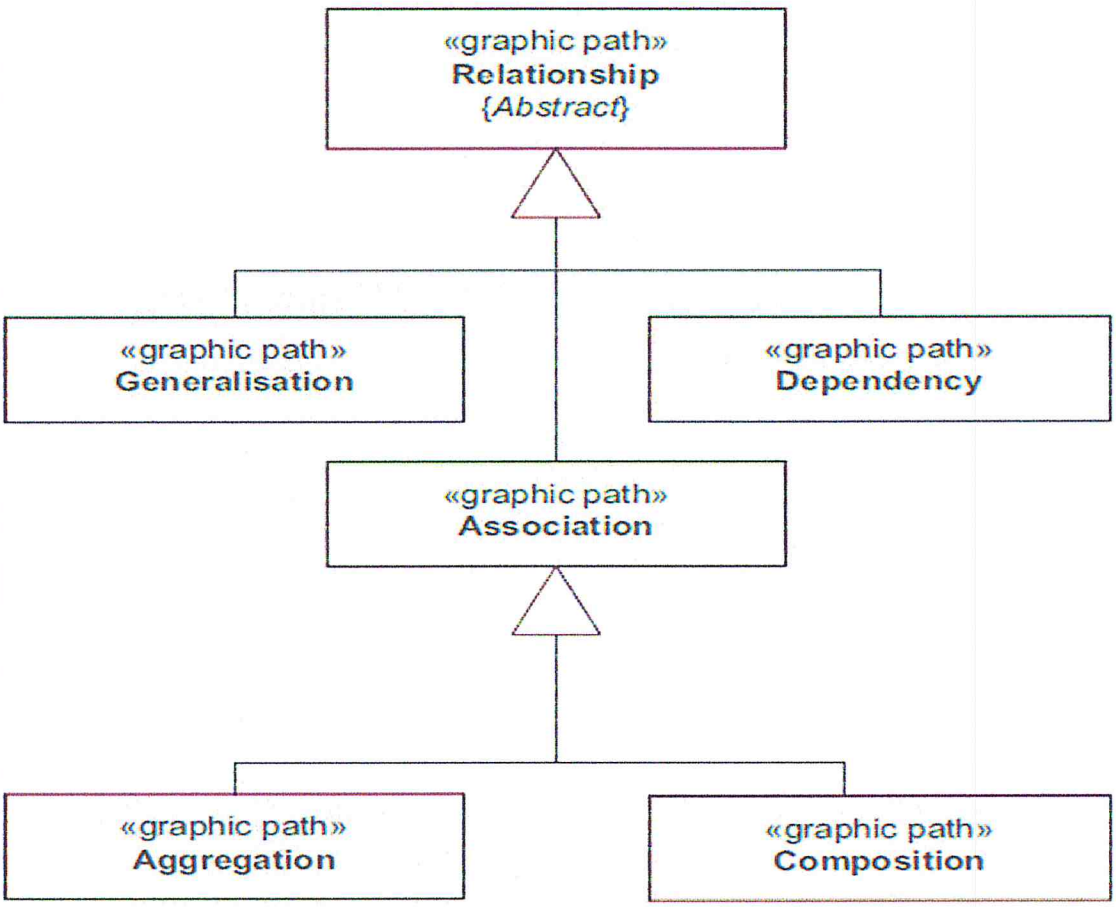



**Figure 39 :** un métamodèle partiel du diagramme de définition de blocs montrant les types de relations [21]

## 3 Source de transformation « Spécification de mission SOS »

La spécification de mission SOS permet de décrire l'objectif principal du SOS (par exemple, sauver le plus de gens possible).

L'approche de [22] est basée sur deux modèles complémentaires, le premier est utilisé pour modéliser la mission du SOS, tandis que le second permet de spécifier les rôles impliqués dans la mission et leurs capacités. Ces modèles sont décrits ci-dessous.

### 3.1 Modélisation de la mission

Une mission SOS est considérée comme un ensemble de processus, chacun composé d'un ensemble ordonné d'activités ou d'actions.

Pour spécifier une mission SOS, les auteurs de [22] ont proposé d'utiliser le diagramme d'activité SysML. En effet, le langage SysML est le langage référence pour les architectes système, il a été défini afin de spécifier des systèmes complexes.

Selon [22], une mission est caractérisée par les concepts suivants :

- La situation qui inclut tout élément d'information ayant une incidence sur la planification de la mission.

- L'énoncé de mission qui représente une déclaration concise de la mission à être accompli.

- L'exécution : qui est décrite par les opérations ordonnées et coordonnées (rôles / tâches).
- Les signaux et commande qui sont le moyen de donner des instructions de communication.

Selon [66], le diagramme d'activité est très adapté pour exprimer les concepts de la mission. Les données et les flux sont utilisés pour exprimer la collaboration entre les rôles (systèmes constitutifs). Une action est réalisée par un système constitutif. Avec les paramètres d'entrée d'activité, il est possible d'exprimer la situation d'un SoS. Le nom de l'activité définit l'énoncé de mission. En utilisant des signaux et des événements, nous modélisons les signaux et les commandes de la mission SOS. Enfin, le planning des actions peut être exprimé à l'aide d'activités, d'actions, de données et de flux de contrôle. Les contraintes peuvent être définies sur les actions pour spécifier le métier sémantique.

### 3.2 Modélisation des rôles et capacités

Pour être en mesure d'identifier une correspondance entre un rôle utilisé dans un diagramme, représentant une mission, et un système constitutif possible, l'architecte du système s'appuie sur les capacités requises pour jouer le rôle. Ainsi, ce dernier doit être correctement spécifié par l'expert du domaine d'application afin d'éviter toute erreur dans le choix des systèmes constitutifs par l'architecte du système.

Dans ce but, les auteurs de [22] ont proposé un nouveau type de diagramme qui permet de modéliser les capacités et les rôles. La figure 40 montre le modèle (appelé CMM) du diagramme des capacités qui met en évidence les concepts impliqués.

En se fondant sur sa connaissance du domaine d'application, l'expert doit maintenir un certain niveau d'abstraction afin qu'il permette de concevoir la mission sans préjuger des systèmes constitutifs qui existent réellement lors du lancement du SOS. Cela se fait à travers le concept de rôle défini par le rôle de la classe de la figure 40, qui est le concept principal dans le modèle de CMM. Il rassemble les compétences nécessaires (capacités de la classe) pour jouer un rôle nécessaire pour accomplir la mission. Une capacité d'un rôle est définie comme la capacité de fournir une certaine expertise aux besoins plus larges dans le contexte de SOS.

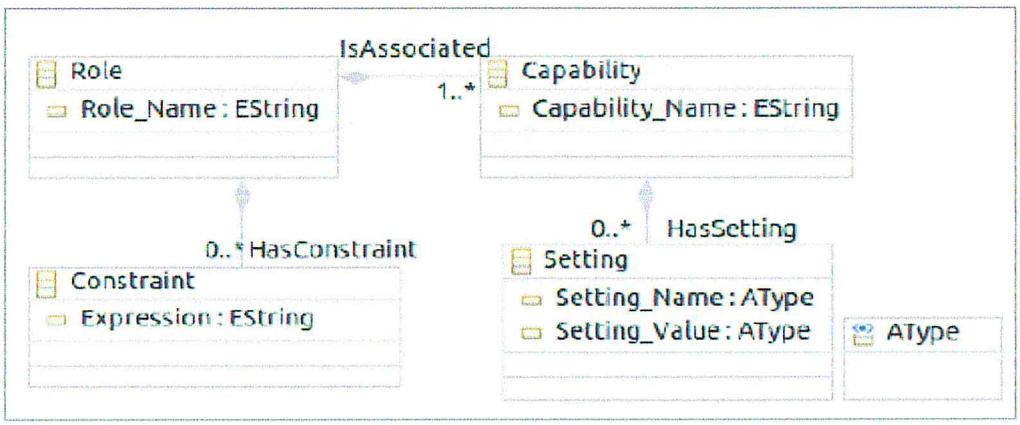



**Figure 40 :** Métamodèle du modèle des capacités CMM [22]

## 4 Cible de transformation « Architecture abstraite »

La définition de la mission contient toute la connaissance du domaine d'application liée à la mission. Le formulaire de description de la mission n'est pas adéquat pour un architecte système. En effet, le but du langage de description de la mission est de permettre aux experts

non informaticiens de mieux exprimer leurs besoins. Cette étape consiste à transformer la description de la mission en une architecture abstraite. L'architecture est appelée abstraite parce qu'elle utilise des rôles plutôt que des systèmes concrets. Pour la description de l'architecture abstraite le diagramme de blocs de SysML est utilisé. A ce niveau, l'architecte possède une architecture représentant une première solution pour le SOS ciblé.

## 5 Conclusion

Dans ce chapitre, nous avons présenté les deux diagrammes du langage de modélisation des systèmes SysML : le diagramme d'activité et le diagramme de définition de blocs. On a choisi ces deux modèles spécifiquement car leurs métamodèles font partie du processus de la transformation prévue pour notre projet, le premier pour la modélisation de la mission en le combinant avec le modèle de capacités spécifique à la modélisation des missions SoS, et le deuxième pour pourvoir générer l'architecture abstraite, et avec ça on a clôturé la partie état de l'art.

Pour la deuxième partie, on va présenter notre contribution et son implémentation, ainsi qu'une étude de cas permettant de tester le processus généré.

# Partie II : Contribution

- Chapitre 4 : Conception
- Chapitre 5 : Implémentation

# Chapitre 4

## CONCEPTION

## 1. Introduction

La partie état de l'art a comporté toutes les informations nécessaires pour effectuer une transformation de modèle. Dans ce chapitre, nous présentons les détails de la correspondance entre le métamodèle de mission, et le métamodèle de l'architecture abstraite. Nous illustrons le processus général des règles de transformation.

## 2. Processus de transformation

Comme mentionné dans la partie état de l'art, la transformation des modèles nécessite de mettre en disposition les métamodèles sources et cible pour l'outil de transformation, qui aura pour but de générer le modèle cible à partir du modèle source donné par l'utilisateur.

Pour notre cas, on a dû adapter les métamodèles selon nos besoins et selon les besoins de spécification, pour pouvoir générer les règles de transformations et effectuer les correspondances entre les deux métamodèles.

### 2.1 Choix de l'outil

La transformation est faite à travers un outil spécifique, et les règles de transformation sont élaborées via un langage dédié pour ça, pour notre travail on a choisi le langage ATL en se basant sur plusieurs critères d'évaluation et caractéristiques de qualités pour choisir la meilleure approche pour notre travail.

Nous avons constaté que ATL est le meilleur en plusieurs critères, comme le fait qu'il possède son propre jeu d'instructions, il factorise son code, ainsi que la correspondance que nous trouvons entre les éléments du modèle source et les éléments de navigation, pour pouvoir créer les éléments du modèle cible. Le critère important pour lequel nous avons pris notre décision est la possibilité de générer des solutions pour les cas complexes, et c'est le cas dans notre travail, d'où nous choisissons ATL pour réaliser la transformation d'une spécification de mission vers une architecture abstraite.

La figure ci-dessous présente le processus global de la transformation. En premier lieu, nous avons combiné les deux métamodèles présentés dans l'état de l'art, celui du diagramme d'activité et celui du diagramme de rôle. En effet, la spécification de mission repose sur ces deux métamodèles. Les rôles pour qui les actions sont attribuées sont considérés comme

partitions dans le diagramme d'activité. Ceci permet de décrire les couples rôle/action qui vont satisfaire des missions.

Après avoir combiné les deux métamodèles, nous avons décrit les règles de transformation, ceci va nous permettre de générer automatiquement l'architecture abstraite du SoS.

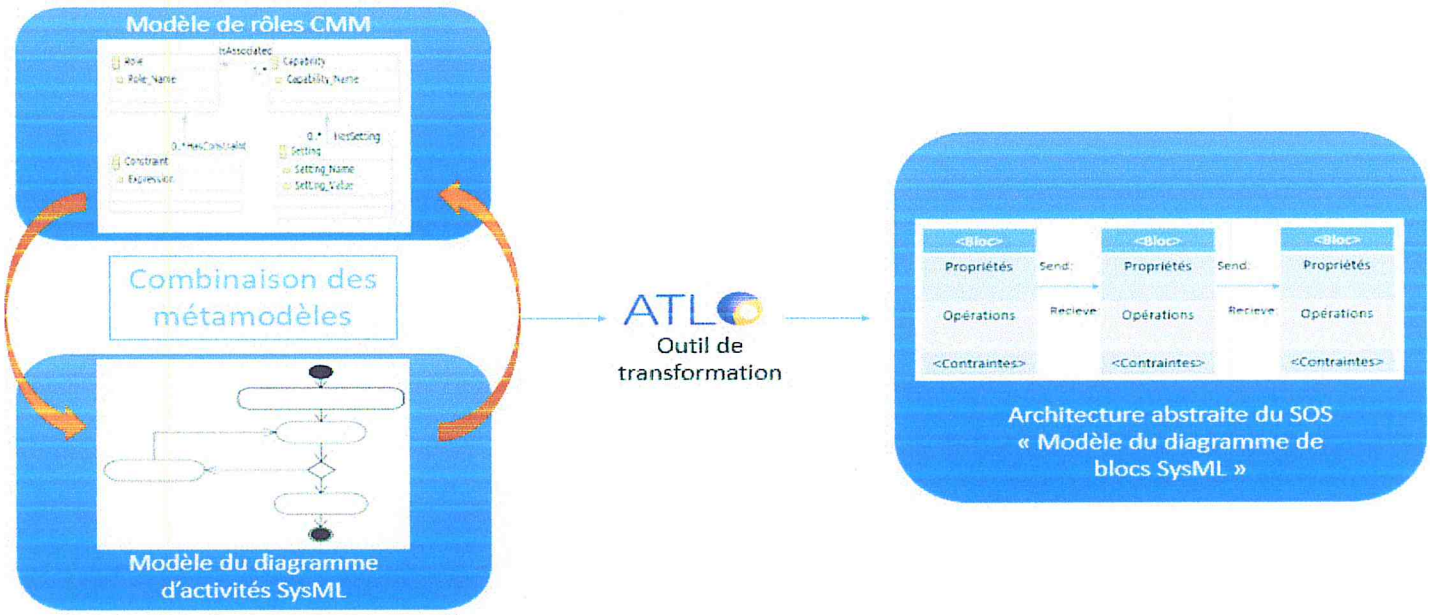



**Figure 41 :** Processus de la transformation

### 2.2 Métamodèles proposés

#### ❖ Métamodèle source

Pour pouvoir générer le modèle de spécification de mission qui est notre source de transformation, nous avons créé un nouveau métamodèle à partir des deux autres métamodèles, le métamodèle du diagramme d'activité SysML, et le métamodèle du digramme CMM proposé par [22], nous avons donné cette proposition car

- Dans le CMM, un chargé de rôle à la capacité d'effectuer les actions qui peuvent être présentées par le digramme d'activité SysML.
- Les interactions entre les actions de la mission SoS se font à partir du diagramme d'activité SysML, ce dernier n'est pas capable de représenter les contraintes des rôles effectués par les chargés de la mission, par contre, le CMM s'en occupe.

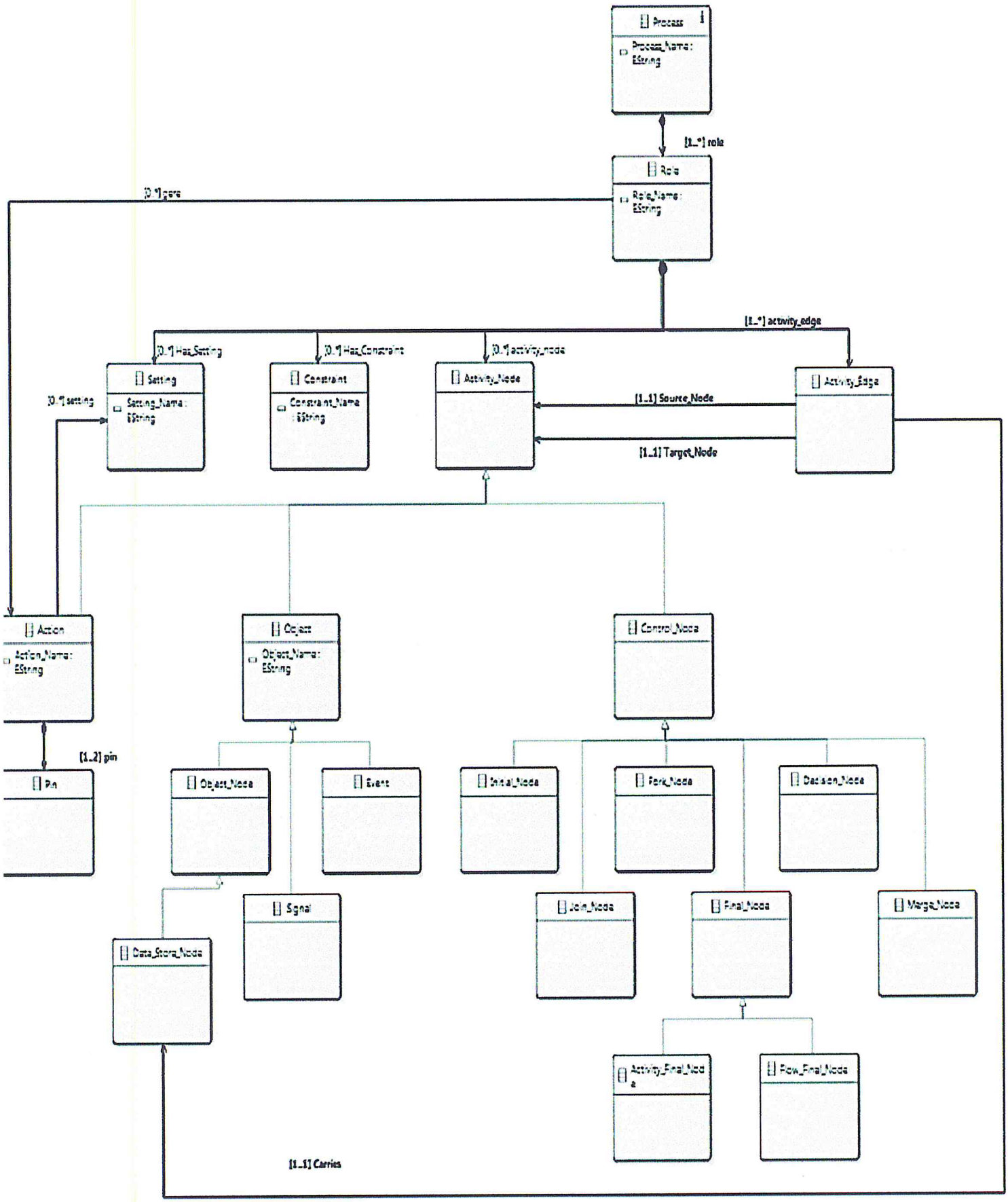



**Figure 42 :** Métamodèle proposé pour la spécification de mission SoS

Le diagramme d'activité est défini pour exprimer les données et les contrôles de flux entre les nœuds. Dans le contexte des SoS, les données et les flux sont utilisés pour exprimer les collaborations entre les rôles.

Une mission SoS est considérée comme un processus, chaque processus est composé d'activités ou d'actions.

Les nœuds d'activités ont certaines caractéristiques en commun. La partition représente la métaclasse qui a permis d'effectuer la combinaison entre les deux métamodèles vu la grande similarité entre le concept de partition et le concept de rôle. En effet, une partition se compose des actions et des flux entre ces actions. Chaque partition va comporter dans le modèle de mission les actions qui lui sont attribuées, ainsi que les flux avec d'autres rôles (partitions).

❖ **Métamodèle cible**

Pour la cible de transformation, nous avons décidé que notre présentation soit faite à travers le diagramme de définition de blocs SysML car :

- Le diagramme de bloc SysML est le meilleur représentant l'architecture abstraite, pour sa clarté, sa simplicité et sa capacité de représenter des systèmes.
- L'architecture abstraite est appelée abstraite car elle utilise les rôles au lieu des systèmes concrets qui peuvent être représentés dans les blocs.

Le diagramme de définition de blocs est composé d'un bloc, qui se compose à son tour des contraintes, propriétés, opérations et flux de données.

Dans le contexte de mission SOS, les systèmes constitutifs sont représentés par des blocs.

Les blocs d'architecture abstraite représentent les rôles responsables des actions. Les aspects de communication sont déduits du diagramme d'activité.

Item Flow permet de faire passer les informations entre les blocs, les opérations représentent les actions faites par le système constitutif, la propriété représente les paramètres concernant le système et ses capacités, ainsi que les contraintes qui représentes tout ce qui peut rencontrer le système avant d'effectuer ses opérations.

Avec cette définition, on peut assurer une base de travail pour l'architecte du système avec un certain un niveau d'abstraction pour qu'il puisse concevoir la mission SoS respectant toutes les exigences définies au début, par l'expert du domaine d'application.

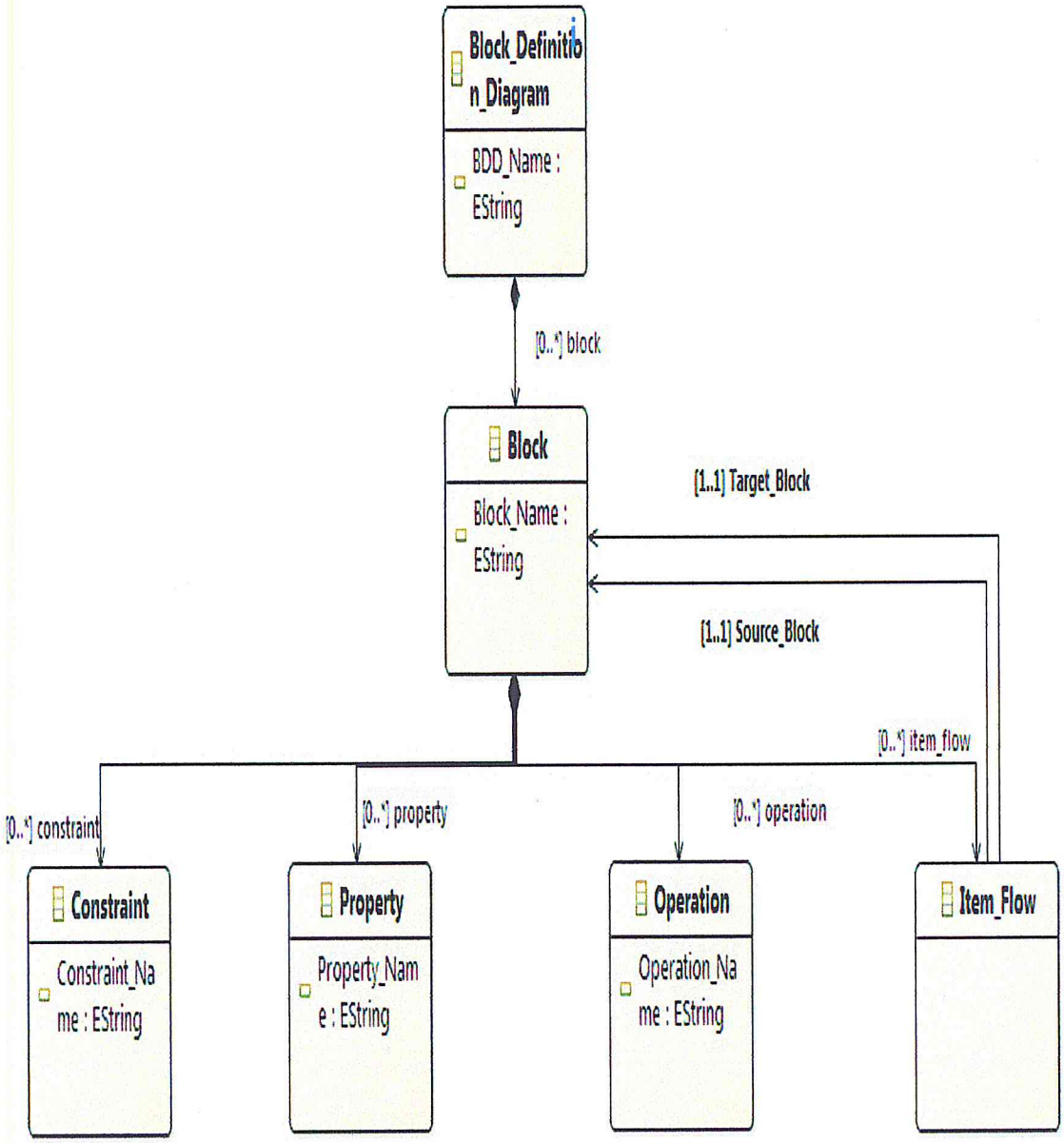



**Figure 43 :** Métamodèle proposé pour l'architecture abstraite d'une mission SOS

### 2.3 Règles de transformation

Pour pouvoir effectuer la transformation depuis un modèle de spécification de mission vers une architecture abstraite, il est nécessaire d'élaborer les règles correspondantes entre les métamodèles source et cible et les adapter selon les besoins, afin de les implémenter en utilisant le langage ATL.

Le tableau suivant représente les principales correspondances sur lesquelles nous nous sommes basés pour décrire les règles de transformation entre les deux métamodèles, suivi des justifications concernant le choix des éléments source et cible.

| SOURCE | CIBLE |
|---|---|
| Rôle | Block |
| Action | Operation |
| Setting | Property |
| Activity Edge | Item Flow |
| Data Store | Item flow |
| Process | Bloc Diagram |
| Contrainte OCL | Contrainte OCL |

**Tableau 02 :** Mapping représentant les règles de transformation entre les métamodèles

- Role To Block : Nous avons fait la correspondance entre la métaclasse « Role » du diagramme CMM et la métaclasse « block » du diagramme de définition de blocs, car le bloc a la capacité d'un système constitutif indépendant, et c'est la même chose pour le rôle dans notre cas. De plus, nous sommes convaincus que dans un rôle nous avons besoin de décrire les opérations qui vont être fournies par ce rôle, le block est le seul élément qui assure ça.

- Action To Operation : Une action doit être effectuée par un système constitutif « Role ». Chaque rôle doit vérifier ses capacités et ses contraintes avant d'effectuer des actions ce qui est le cas pour « Operation » dans le diagramme de bloc.

- Setting To Property : Les « Setting » du modèle CMM permettent d'identifier les paramètres pouvant affecter la capacité d'un rôle, ce sont des données qui doivent être définies au cours de la spécification des exigences d'un SoS. Le méta-attribut

« Property » du diagramme de définition de blocs définit les propriétés qui concernent le bloc. Donc nous pouvons utiliser la sémantique de la propriété pour mettre dedans les Settings. Par convention, nous avons décidé de donner un nom significatif à une propriété pour que l'architecte système sache à quelle capacité est liée cette propriété. Nous avons décidé d'utiliser la notation suivante : NomCapacité_NomPropriété.

- Activity Edge To Item Flow : Item Flow permet de transporter les informations entre deux blocs, comme Activité Edge qui fait la connexion entre deux nœuds d'activité, on remarque la grande similarité entre les deux composants.

- DataStore To Item Flow : Un dataStore qui représente un échange d'information entre deux actions appartenant à des rôles différents doit être véhiculé par un item flow. Pour tester si le datastore est échangé entre deux actions de rôles différents, nous avons utilisés des contraintes associées à ce mapping car c'est un mapping qui ne va pas avoir lieu si les deux actions appartiennent au même rôle.

- Process To Block diagram : Tout process dans le diagramme d'activité va être représenté par son propre diagramme de définition de bloc comportant les différents rôles et les différents flux entre les rôles qui contribuent à l'achèvement de l'activité (process).

- La contrainte OCL : Les contraintes OCL associées aux actions qui vont être réalisé par un rôle peuvent être mappées directement dans la partie contrainte d'un bloc.

### 3. Architecture générale de l'application

La figure suivante représente l'architecture globale de la transformation à partir de la source vers la cible.

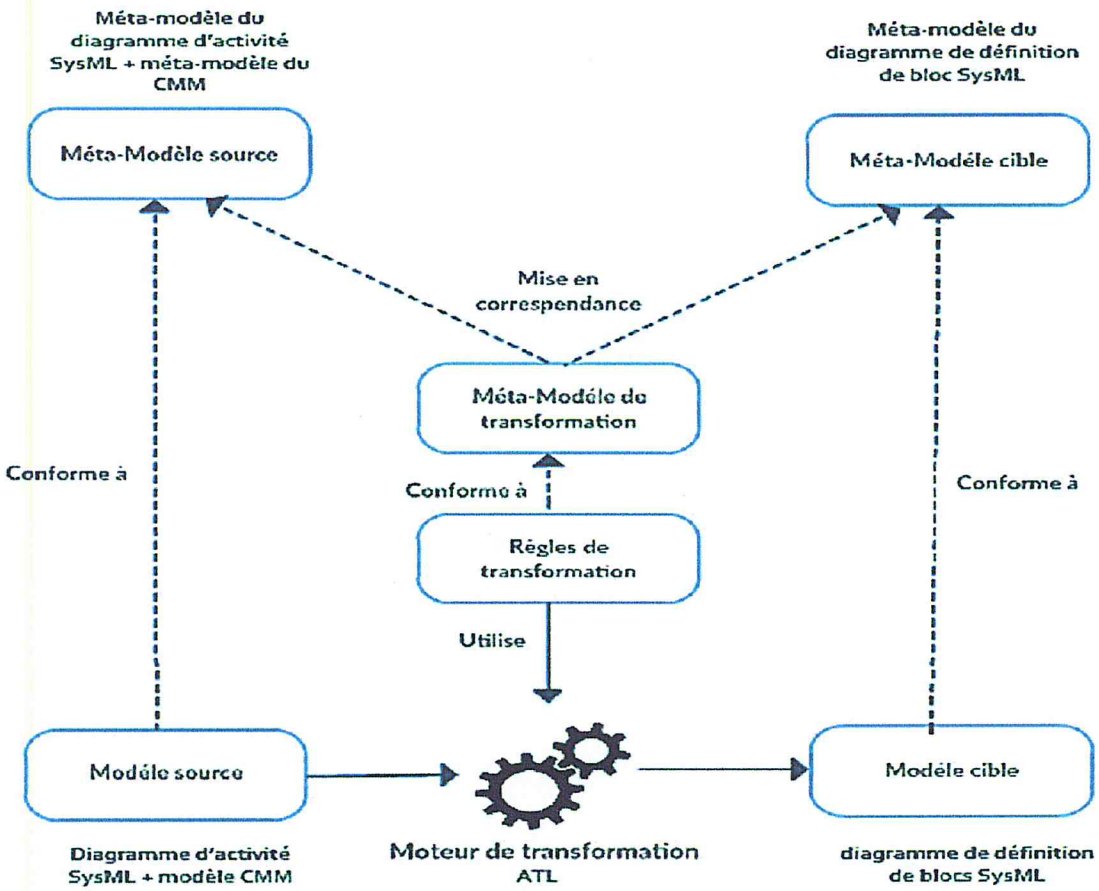



**Figure 44 :** Architecture générale de la transformation

## 4. Conclusion

La transformation de la spécification d'une mission SoS vers une architecture abstraite est une opération qui relie des concepts différents. Les chargés de la mission qui ont des rôles spécifiques sont capables d'effectuer certaines actions, c'est pour ça nous avons mis la correspondance entre le diagramme de rôle CMM et le diagramme d'activité SysML, et ceci est pour pouvoir élaborer une spécification complète de la mission. Depuis cette dernière on est passé au diagramme de définition de blocs qui représente à son tour l'architecture abstraite de la mission, ce passage a eu lieu grâce aux règles de transformation qui ont été créées.

# Chapitre 5

## *IMPLEMENTATION*

## 1. Introduction

Après l'étape conception, nous allons dans ce chapitre donner les détails d'implémentation de la transformation. Nous allons présenter les différents outils qui nous ont permis de concrétiser la transformation, et nous allons montrer les résultats à travers un exemple.

## 2. Outils d'implémentation

Les outils logiciels suivants sont nécessaires pour pouvoir effectuer la transformation de modèles.

### 2.1 Eclipse Modeling Framework (EMF)

Le projet EMF est un cadre de modélisation et une installation de génération de code pour les outils de construction et d'autres applications basées sur un modèle de données structuré. À partir d'une spécification de modèle décrite dans XMI, EMF fournit des outils et une prise en charge d'exécution pour produire un ensemble de classes Java pour le modèle, ainsi qu'un ensemble de classes d'adaptateur qui permettent l'affichage et la modification basée sur la commande du modèle, et un éditeur de base.

### 2.2 Ecore

Eclipse Core est une norme commune pour les modèles de données, de nombreuses technologies et des cadres sont basés sur. Cela inclut les solutions serveur, les frameworks de persistance, les frameworks d'interface utilisateur et la prise en charge des transformations. Veuillez consulter le projet de modélisation pour une vue d'ensemble des technologies EMF.

### 2.3 ADT

ATL est accompagné d'un ensemble d'outils construits au-dessus de la plate-forme Eclipse. La figure 14 représente l'architecture globale des outils de développement ATL (ADT). ADT est composé du moteur de transformation ATL (bloc moteur) et de l'environnement de développement intégré ATL [16].

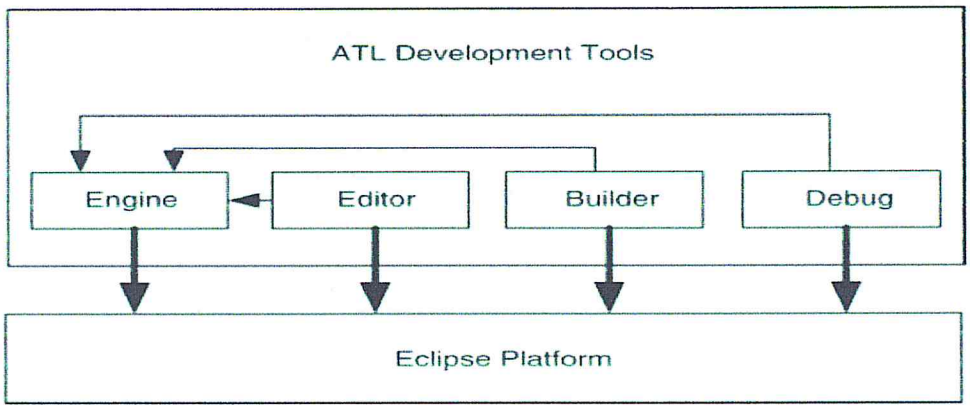



**Figure 45 :** Architecture ADT [16]

## 3. Etude de cas

Dans cette section, nous présentons l'implémentation de transformation de modèle de spécification de mission SoS à une architecture abstraite à travers l'étude de cas «Crowd Management» qui concerne le développement d'un SoS de gestion de la foule (CMSoS) spécialement pour les événements sportifs.

### 3.1 Présentation de l'étude de cas « Crowd Management »

Cette étude de cas fait partie du système de réponse aux catastrophes qui est un exemple largement utilisé. Il vise à développer un système intégré de contrôle des foules des événements temporaires, tels que les événements sportifs, réunions politiques…

Le processus de haut niveau lié à la mission principale (Crowd) est composé des sept activités proposées, et qui sont nécessaires pour contrôler toute situation d'urgence.

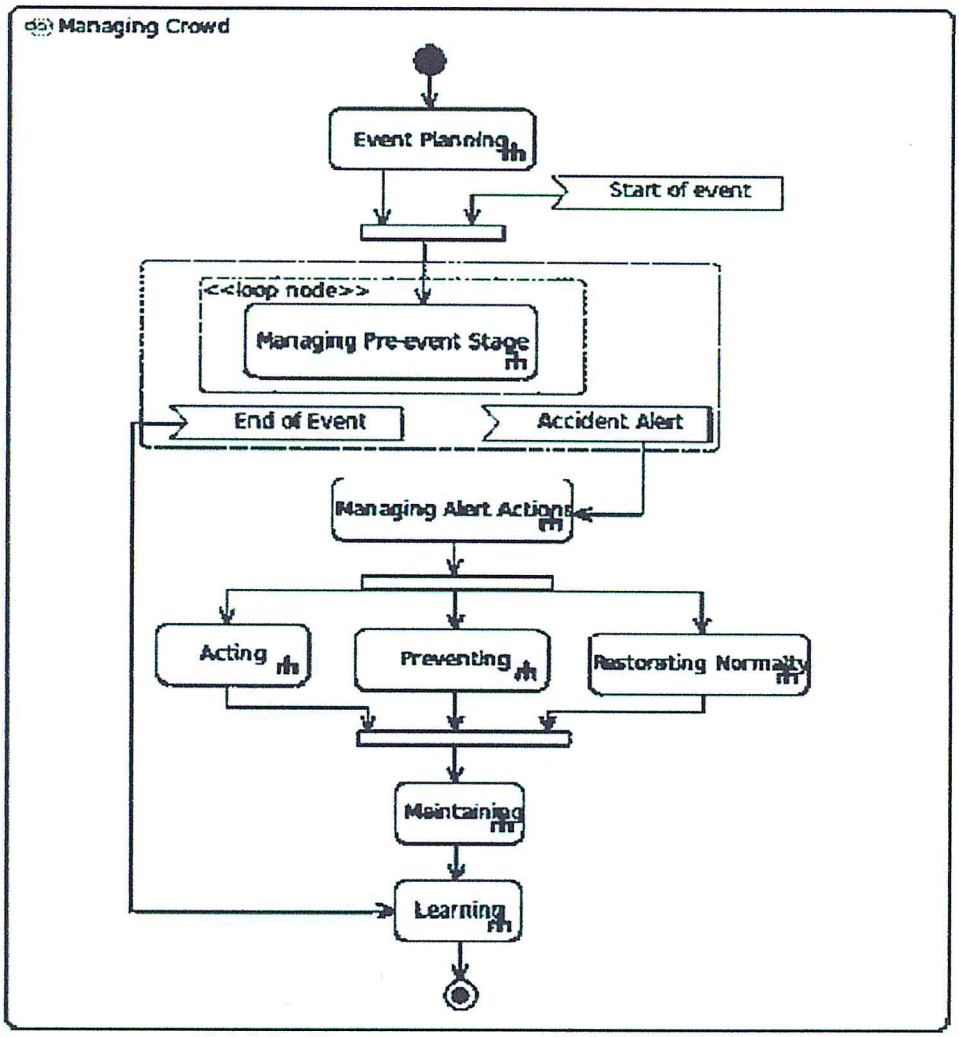



**Figure 46 :** Diagramme d'activité SysML de l'étape « Managing Crowd » [22]

L'activité « Managing Pre-event Stage » sert à contrôler le comportement de la foule et d'éviter tout incident, cela commence quand l'événement démarre. C'est une activité répétitive et interruptible au même temps, elle peut être interrompue par la fin de l'événement ou quand une alerte d'accident se déclenche.

La figure suivante, représente le diagramme d'activité de l'étape « Managing Pre-event stage incluant trois autres activités « Assessing Risks », « Training Staff » et « Putting precautions ».

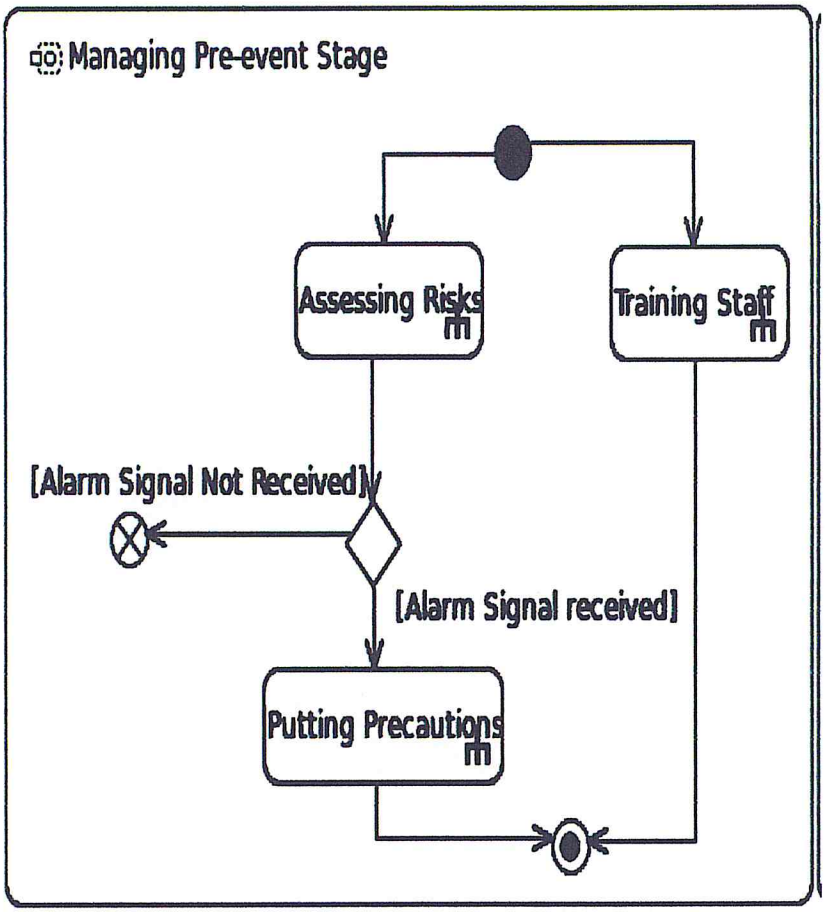



**Figure 47 :** Diagramme d'activité SysML de l'étape « Managing Pre-event Stage » [22]

Les actions de la mission <<Assessing Risks>> sont planifiées et représentées dans le diagramme d'activité. Par la suite, les rôles sont attribués aux différentes actions. La figure suivante illustre le résultat.

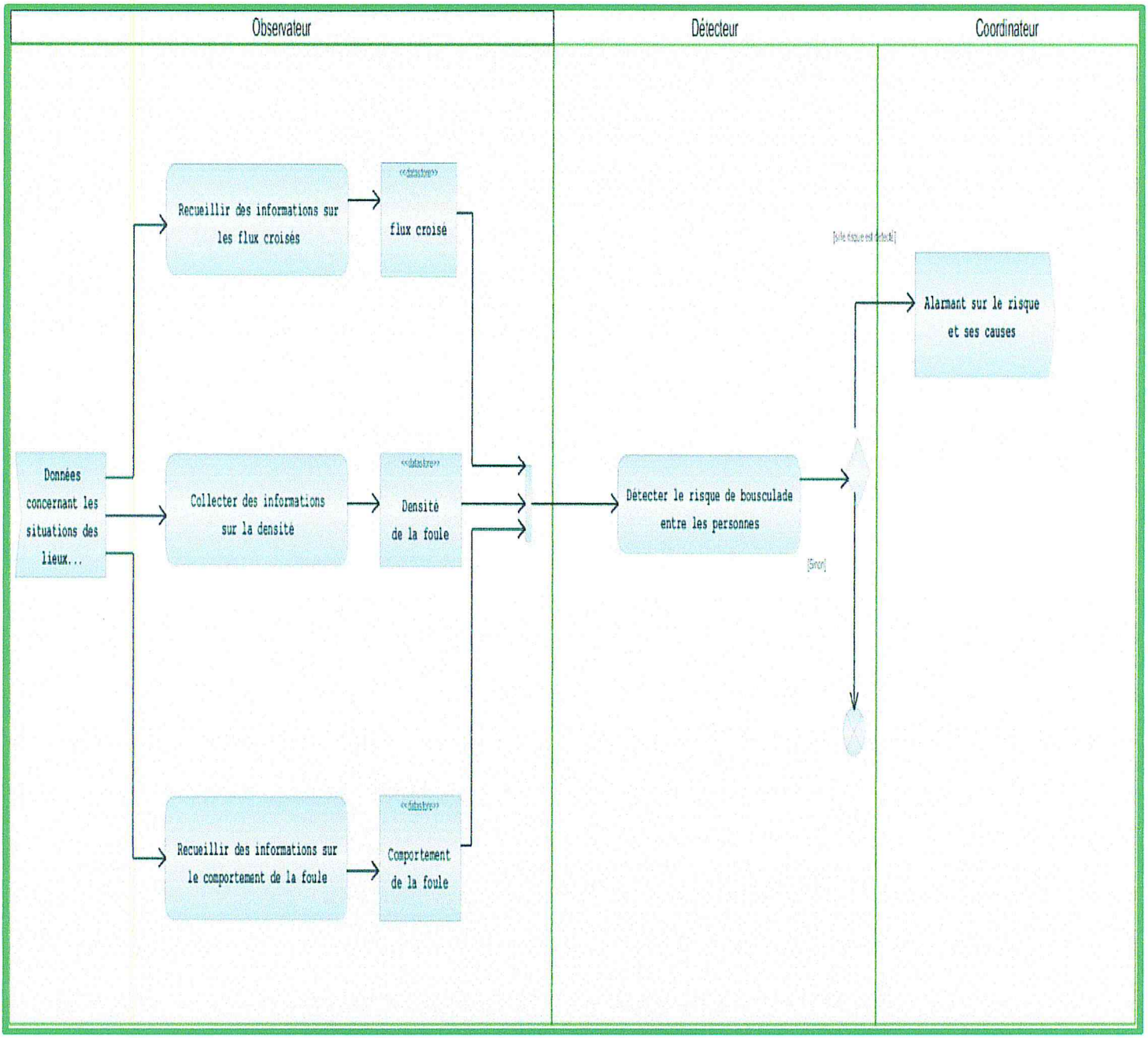



**Figure 48 :** Modèle de spécification de la mission SoS « Assessing Risks »

Pour valider la transformation, nous allons prendre le « Assessing Risks » comme exemple de modèle de mission, et générer automatiquement son architecture abstraite.

### 3.2 Implémentation de la transformation du modèle « Assessing Risks »

Pour obtenir une architecture abstraite de cette mission à l'aide de l'outil ATL on doit suivre les étapes suivantes :

- Créer un projet ATL :

File -> New -> Other -> Choisir ATL Project

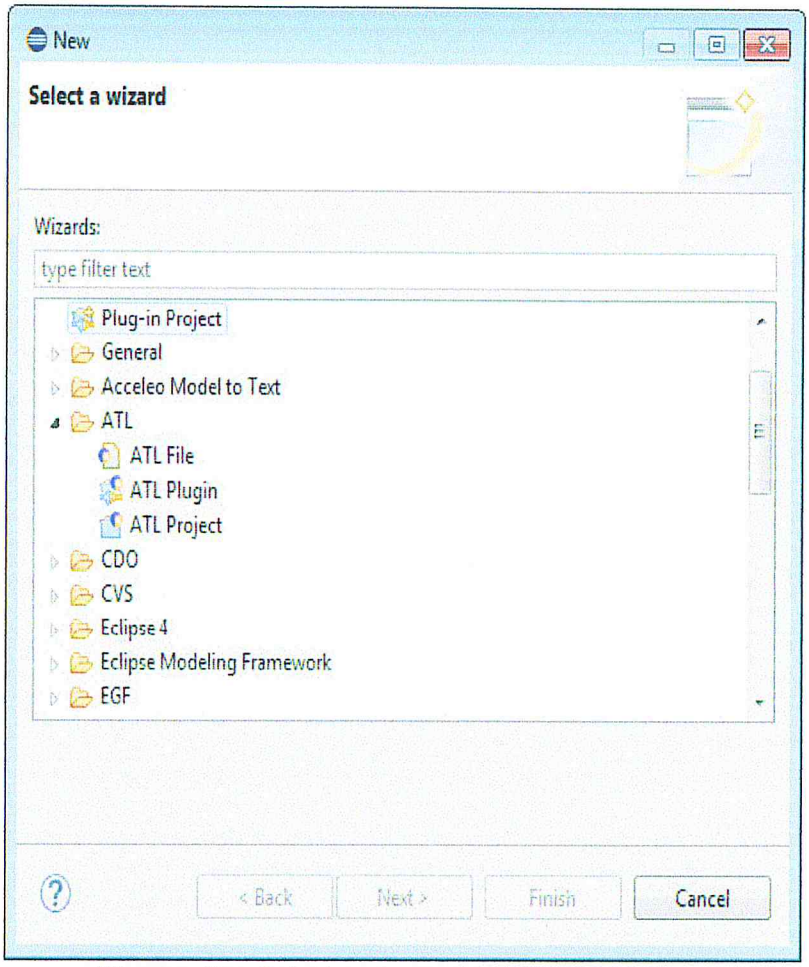



**Figure 49 :** Création du projet ATL sur Eclipse

- Créer 3 dossies (Folders)
  - Un dossier pour les métas modèles
  - Un autre pour les modèles
  - Et le dernier pour le fichier ATL.

Ensuite, en créant un modèle Ecore, nous aurons la possibilité d'introduire les métamodèles source et destination graphiquement en utilisant la palette, ainsi que le modèle source de transformation. Les résultats obtenus sont représentés comme suit.

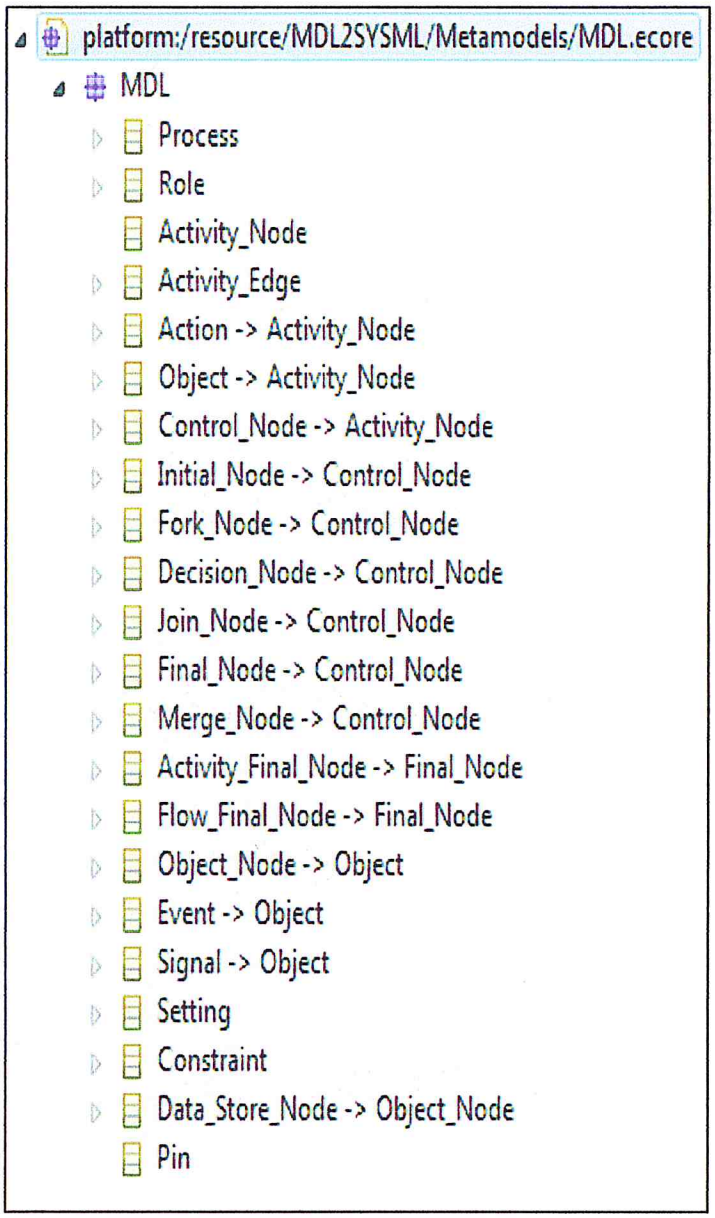



**Figure 50 :** Eléments du métamodèle source représentés sous Ecore

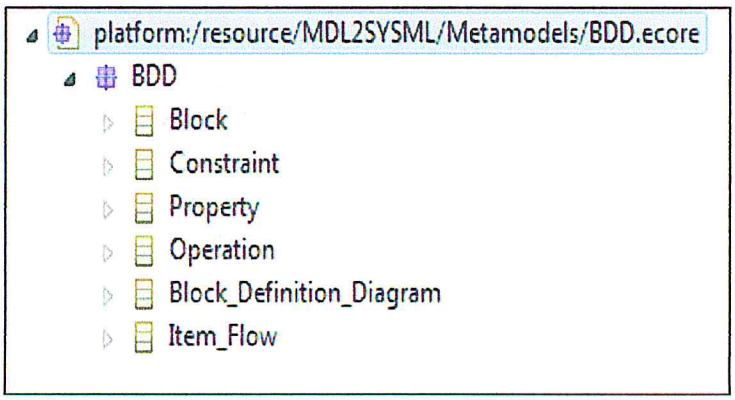



**Figure 51 :** Eléments du métamodèle cible représentés sous Ecore

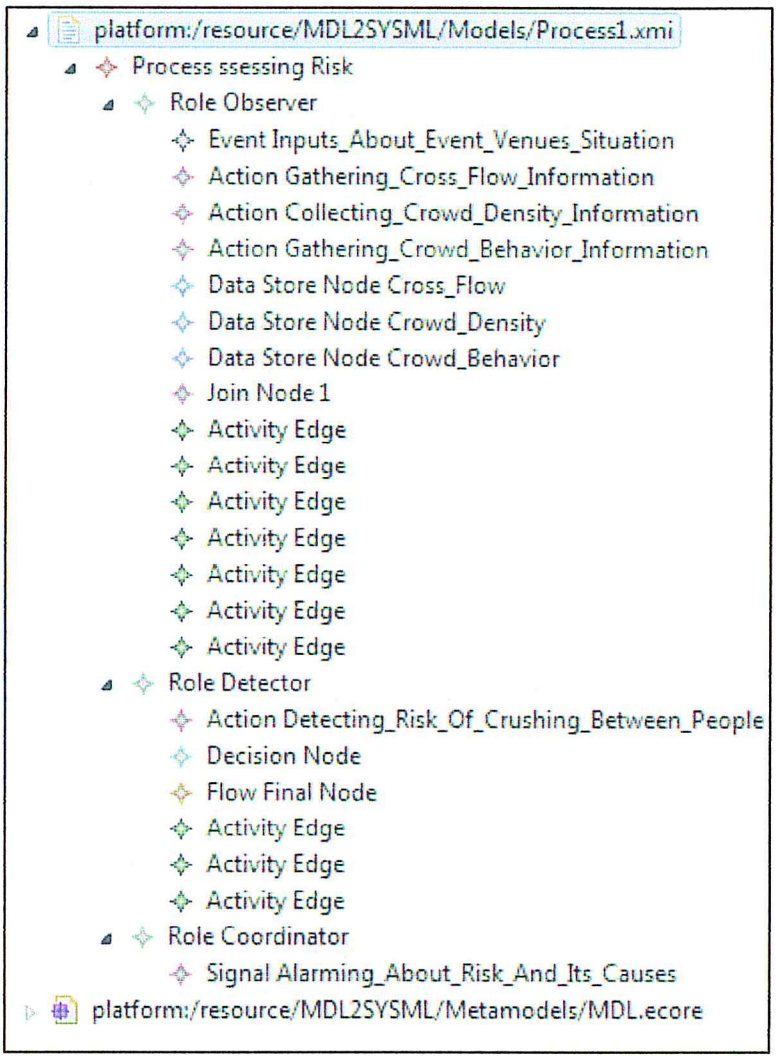



**Figure 52 :** Eléments du modèle source représentés sous Ecore

Après avoir mettre les modèles et leurs éléments en entrée, nous aurons besoin de spécifier les règles de transformations entre les métamodèles sous ATL, ces règles ont été représentés graphiquement dans le chapitre précédent.

- Les règles implémentées sous ATL

```
rule ProcessToBDD {
    from
        proc : MM!Process
    to
        bdd : MM1!Block_Definition_Diagram (
            BDD_Name <- proc.Process_Name,
            block <- proc.role
        )
}
```

**Figure 53 :** La règle « ProcessToBDD » sous ATL

```
rule RoleToBlock {
    from
        rol : MM!Role
    to
        block : MM1!Block (
            Block_Name <- rol.Role_Name,
            constraint <- rol.Has_Constraint,
            property <- rol.Has_Setting,
            operation <- rol.gere,
            item_flow<- rol.activity_edge
            )
}
```

**Figure 54 :** La règle « RoleToBlock » sous ATL

```
rule ContraintaToContraintb {
    from
        conta : MM!Constraint
    to
        contb : MM1!Constraint (
            Constraint_Name <- conta.Constraint_Name
        )
}
```

**Figure 55 :** La règle « ContraintaToContraintb » sous ATL

```
rule SettingToProperty {
    from
        set : MM!Setting
    to
        pro : MM1!Property (
            Property_Name <- set.Setting_Name
        )
}
```

**Figure 56 :** La règle « SettingToProperty » sous ATL

```
rule ActionToOperation {
    from
        act : MM!Action
    to
        op : MM1!Operation (
            Operation_Name <- act.Action_Name
        )
}
```

**Figure 57 :** La règle « ActionToOperation » sous ATL

```
rule Edge2Item_Flow {
   from
      flow : MM1!Activity_Edge( ((flow.Source_Node).refImmediateComposite()) <> ((flow.Target_Node).refImmediateComposite())
       )
      -- verifier si la source de l'arc est different de la destination ( l'arc est un flut entre deux roles )

   to

      item : MM1!Item_Flow  (
      Source_Block <-  (flow.Source_Node).refImmediateComposite(),
          Target_Block <-  (flow.Target_Node).refImmediateComposite()

      )
}
```

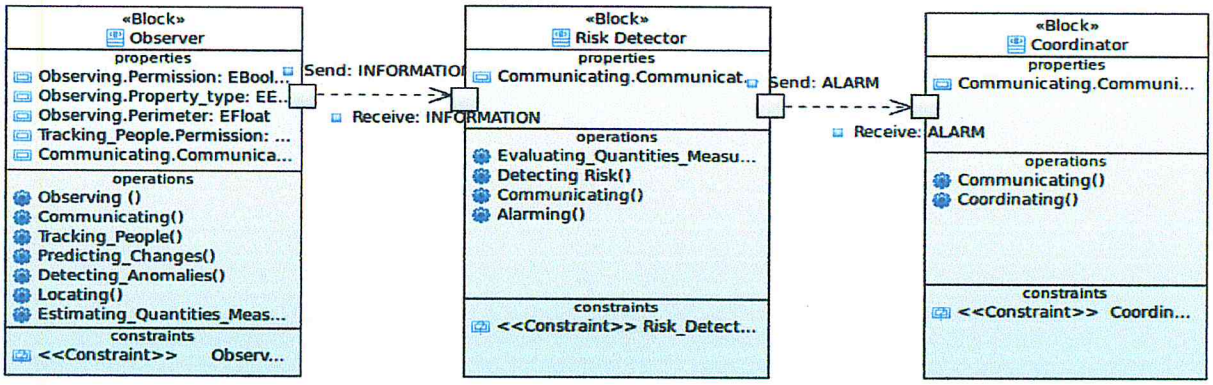
**Figure 58 :** La règle « Edge2Item_Flow» sous ATL

Après l'élaboration des règles de transformation, le fichier ATL devient prêt à être exécuté, le résultat obtenu est sous forme d'un fichier XML représentant le modèle cible de transformation, comme suit

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <BDD:Block_Definition_Diagram xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:BDD="www.BDD.com">
3   <block Block_Name="Observer">
4     <operation Operation_Name="Gathering_Cross_Flow_Information"/>
5     <operation Operation_Name="Collecting_Crowd_Density_Information"/>
6     <operation Operation_Name="Gathering_Crowd_Behavior_Information"/>
7     <item_flow Source_Block="//@block.0" Target_Block="//@block.1"/>
8   </block>
9   <block Block_Name="Detector">
10    <operation Operation_Name="Detecting_Risk_Of_Crushing_Between_People"/>
11    <item_flow Source_Block="//@block.1" Target_Block="//@block.2"/>
12  </block>
13  <block Block_Name="Coordinator"/>
14 </BDD:Block_Definition_Diagram>
15
```

**Figure 59 :** Fichier XML démontrant le résultat de transformation

Le résultat obtenu en XML est équivalent au diagramme de définition de blocs suivant :




**Fichier 60 :** Diagramme de définition de blocs démontrant le résultat de transformation

## 4. Conclusion

Nous avons présenté dans ce chapitre l'implémentation de notre transformation. Nous avons commencé par définir l'environnement de travail, ainsi que les outils du développement utilisés. Nous avons par la suite présenté l'application que nous avons développée suivie d'une étude de cas.

# Conclusion générale

L'objectif de ce travail, consiste en l'automatisation de la transformation d'une spécification de mission vers l'architecture abstraite. Le but de cette transformation est de créer un lien entre la définition des exigences faite par l'expert du domaine d'application et la phase de conception qui est faite par l'architecte système. Ceci a pour but d'éviter la déviation de la conception des objectifs initiaux de la mission. Ainsi, l'architecte système aura à sa disposition la définition de la mission du SoS sous forme d'une architecture abstraite, qui lui servira en tant que guide et contrôleur pendant l'étape d'évolution.

Le travail qui nous a été demandé consiste en premier lieu, à définir la spécification de mission déduite depuis les recherches de [22], en suivant leur approche de conception. Ensuite, nous devions étudier les métamodèles source et destination de la transformation, faire une étude comparative des approches et des langages existants de transformation de modèles, définir les règles de transformation entre les métamodèles selon les besoins de spécification et les possibilités d'adaptation et de correspondance, et à la fin, nous devions implémenter la solution, en automatisant l'opération de transformation.

Nous pouvons humblement affirmer que les objectifs du projet ont été intégralement atteints avec satisfaction des buts fixés au début de l'étude.

Ce travail nous a permis d'acquérir des connaissances sur :

- La construction des SoS, la conception et la spécification des missions d'un SoS, ainsi que les approches de MDA concernant la métamodélisation et la transformation de modèles.
- La programmation des règles de transformation de modèles en utilisant le langage ATL.

Les perspectives envisagées pour améliorer ce travail sont multiples, nous identifions d'entre elles :

- L'implémentation d''une autre transformation vers l'architecture concrète.

# Bibliographie

[1] Skander Turki. Ingénierie système guidée par les modèles : Application du standard IEEE 15288, de l'architecture MDA et du langage SysML à la conception des systèmes mécatroniques. Thèse de Doctorat, Université du Sud Toulon Var France, 2008.

[2] Department of Defense Office. The Deputy Under Secretary of Defense for Acquisition and Technology, USA (august 2008) Systems Engineering Guide for Systems of Systems. https://www.acq.osd.mil/se/docs/se-guide-for-sos.pdf

[3] Hitchins, D. 2009. "System of Systems - The Ultimate Tautology." http://www.hitchins.net/profs-stuff/profs-blog/system-of-systems---the.html. Consulté le 20 mars 2018

[4] M. W. Maier, "Architecting principles for systems-of-systems", Systems Engineering, vol. 1, no. 4, pp. 267-284, Feb. 1998.

[5] E. Silva, T. Batista, and E. Cavalcante, "A mission-oriented tool for system-of-systems modeling", Proceedings of the 3rd International Workshop on Software Engineering for Systems-of-Systems. USA : IEEE, 2015, pp. 31-36.

[6] E. Silva, T. Batista, and F. Oquendo, "A mission-oriented approach for designing system-of-systems", Proceedings of the 10th System of Systems Engineering Conference. USA : IEEE, 2015, pp. 346-351.

[7] E. Silva, E. Cavalcante, T. Batista, F. Oquendo, F. C. Delicato, and P. F. Pires, "On the characterization of missions of systems-of-systems", Proceedings of the 2014 European Conference on Software Architecture Workshops. USA : ACM, 2014.

[8] Paul C. Clements « A Survey of Architecture Description Languages », Software Engineering Institute. Carnegie Mellon University. Pittsburgh, PA 1521

[9] E. Silva, E. Cavalcante, T. Batista, « Refining missions to architectures in software-intensive systems-of-systems », Proceedings of the Joint 5th International Workshop on Software Engineering for Systems-of-Systems and 11th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems, May 20-28, 2017, Buenos Aires, Argentina

[10] About OMG, consulté le 14 mai 2018, https://www.omg.org/about/index.htm

[11] Mouna AOUAG, Des diagrammes UML 2.0 vers les diagrammes orientés aspect à l'aide de transformation de graphes, Thèses de Doctorat LMD, Université Constantine2, Algérie (2014)

[12] Samba Diaw, Redouane Lbath, Bernard Coulette, Etat de l'art sur le développement logiciel basé sur les transformations de modèles, Université de Toulouse

[13] Houda HAMROUCHE, Une Approche de transformation des diagrammes d'activité d'UML vers CSP basée sur la transformation de graphes, Thèses de Magister, Ecole Doctorale de l'Est – Pôle ANNABA, Algérie (2017)

[14] Sen, S., Moha, N., Mahé, V., Barais, O., Baudry, B., Jézéquel, J.-M. : Reusable model transformations. SoSyM 11(1) (2010)

[15] Jouault, F, Contribution à l'étude des langages de transformation de modèles. PhD Thesis, Université de Nantes (2006)

[16] Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I. : ATL : A model transformation tool. Science of Computer Programming 72(1/2), 31–39 (2008)

[17] S. Diaw, R. Lbath, B. Coulette, "Etat de l'art sur le developpement logiciel base sur les transformations de modéles", TSI Herms Sciences Publications, vol. 29, pp. 4–5/2010-536, juin 2010.

[18] Object Management Group. "OMG Systems Modeling Language (OMG SysML™)," Version 1.1. November 2008. OMG Document Number: formal/2008-11-01, Standard document URL: http://www.omg.org/spec/SysML/1.1.

[19] Damien Foures. Transformation des diagrammes d'activités SysML1.2 vers les réseaux de Petri dans un cadre MDE. Rapport LAAS n° 11864. Rapport de stage. 2012, 55p. <hal-00761053>

[20] Hoffmann, Hans-Peter, "SysML-Based Systems Engineering Using a Model-Driven Development Approach," Proceedings of INCOSE 2006 International Symposium, Orlando, FL, Jul. 12, 2006

[21] Holt, Jon ; Perry, Simon, Jan 01, 1753, SysML for Systems Engineering. 2nd Edition : A Model-Based Approach. The Institution of Engineering and Technology, Stevenage, ISBN : 9781849196529

[22] I. Cherfa, S. Sadou, N. Belloiry, R. Fleurquin « Involving the Application Domain Expert in the Construction of Systems of Systems » SoSE System of Systems Engineering Conference, Paris, June 2018, pp. 19–22.