REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre : /M2/2022

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : Génie Logiciel et Systèmes Distribués

GLSD

Reconstruction d'architecture pour les systèmes IOT

Par :

DJOGHMA Younes

Soutenu le 26 /06/2022, devant le jury composé de :

Bennoui Hammadi	Professeur	Président
Kerdoudi Mohammed Lamine	MCA	Rapporteur
Djaber Khaled	MAA	Examinateur

Année Universitaire : 2021 – 2022

Dedicace

Je dédie ce travail à mes chers parents qui m'ont aidé tout au long de mes années d'études et dans ma vie. J'espère qu'ils seront fiers de moi et de mes frères et sœurs. J'adresse également mes sincères remerciements au Dr

Mohamed Lamine Kerdoudi, qui m'a accompagné et m'a beaucoup aidé dans la réalisation de cette mémoire, car il en était l'auteur, car son bureau était toujours ouvert pour résoudre toutes les questions et tous les problèmes que j'ai rencontrés dans ce projet.

Résumé :

La reconstruction de l'architecture des systèmes IoT est un mécanisme de réextraction des modèles CAPS via des codes de programmation comme ThingML en suivant certaines règles de conversion entre les deux langages L'architecture de logiciel joue un rôle important lors de la maintenance et du développement du système, Cela permet d'économiser du temps, de l'argent et des efforts.

L'objectif de ce travail est de fournir un mécanisme de mappage et de conversion entre deux langages de modélisation IOT différents : CAPS-SAML et ThingML.

Cette transformation est nécessaire pour obtenir des modelés générés automatiquement à partir de code. Cela nécessite une grande connaissance des deux langages et d'avoir un outil ou un mécanisme entre faire une telle conversion qui sera construit sur des bases scientifiques et logiques. Notre mémoire présente notre travail couvrant toutes les étapes de génération de modèles SAML à l'aide du Framework CAPS.

Mots-clés : Architecture logicielle, de l'architecture des systèmes IoT, systèmes IoT

RECONSTRUCTION D'ARCHITECTURE POUR LES SYSTEMES IOT

DJOGHMA YOUNES
Juin 26, 2022

Table des matières

Introduction générale

L'Internet des objets a été introduit pour la première fois par Kevin Ashton. Nommé pour une variété des objets, qu'on peut contrôler par internet telles que des graphiques uniques et des systèmes d'adressage, qui sont capables d'interagir et de fonctionner avec les autres objets dans le système IOT. La plate-forme principale de l'IOT est des objets quotidiens tel que (réfrigérateur, télévision, ordinateurs portables, téléphones portables, etc.). Ces objets sont équipés de composants électroniques tels que des supports de correspondance radio, des processeurs de traitement, des capteurs et/ou des actionneurs, etc. La grande force de l'Internet des Objets réside dans le fait que ses objets communiquent, analysent, nécessitent et gèrent les données de manière autonome.

L'architecture logicielle de l'Internet des objets (IoT) fait référence à l'infrastructure d'un système logiciel. Chaque architecture comprend des éléments logiciels et leurs relations, propriétés. L'architecture du système IoT est une métaphore similaire à l'architecture d'un bâtiment. Sert de planification de projet de système et de développement, identifiant des tâches nécessaires à accomplir par les équipes de conception. L'architecture logicielle IoT consiste à faire des choix structurels de base qui sont coûteux à modifier une fois mis en œuvre. Les options d'architecture logicielle incluent des choix structurels spécifiques parmi les capacités de conception de logiciels.

Lorsque le code des systèmes IoT est hétérogène et volumineux en termes de nombre de lignes de code et de complexité, la compréhension de ces systèmes IoT devient très difficile et perd le temps aux développeurs pour comprendre, augmente le coût de la maintenance. Pour résoudre ce problème dans ce travail on s'intéresse à l'extraction des architectures des logiciel.

Nous avons proposé un processus pour extraire les modèles CAPS à partir des codes écrit en langage de programmation ThingML. Ces modèles permettent d'aider les développeurs à améliorer l'évolution et la maintenance des logiciels.

Nous avons fourni un mécanisme de mappage et de conversion entre deux langages IoT différents : CAPS-SAML et ThingML. En convertissant le modèle ThingML en modèle SAML en suivant certaines règles de conversion. Nous avons utilisé dans notre transformation deux méta-modèles existants des langages ThingML et SAML. En d'autres termes, les règles utilisées consistent en un mappage entre les éléments des deux méta-modèles. Le modèle SAML généré peut être visualisé graphiquement en utilisant le Framework CAPS.

Ce mémoire est organisé comme suit :

- **Introduction générale :** présente le contexte, la problématique et l'objectif de ce travail.
  - **Chapitre 1 :** nous donnons d'abord un aperçu de l'Internet des objets, de son histoire, de son importance, de ses composants, de ses fonctions, de ses domaines d'application et des technologies alternatives.

  - **Chapitre 2 :** fournit un aperçu du l'architecture logiciel en termes de définition du l'architecture logiciel et de l'importance du l'architecture logiciel et du développement pour et à travers la réutilisation et les modèles architecturaux ainsi que le développement de logiciels.

  - **Chapitre 3 :** présente les derniers développements en matière d'extraction d'architecture d'application par le code ainsi que d'extraction de code par l'ingénierie d'application. Présentez-nous les technologies actuelles et les travaux de développement dans ce domaine.

  - **Chapitre 4 :** présente l'approche proposé et la solution pour convertir le code ThingML en un schéma CAPS,

  - **Chapitre 5 :** présente les détails sur la mise en œuvre de notre approche proposée et le cadre et les outils que nous avons utilisés.

  - **Conclusion générale** : nous concluons et présentons quelques perspectives d'avenir dans le domaine de l'extraction architecture de logicielle.

Chapitre 1 : Concepts fondamentaux de l'Internet des objets

1.1. Introduction

Dans ce chapitre, nous présentons la définition de l'Internet des Objets (Internet of Things), son histoire et son importance, ainsi que ses composantes et domaines d'application, nous expliquons également ses avantages et ses inconvénients.

1.2. L'internet des objets (IOT)

1.2.1. Définition de l'internet des objets

Le terme IoT est apparu la première fois en 1999 dans un discours de l'ingénieur britannique Kevin ASHTON. Il servait à désigner un système ou les objets physiques sont connectés à internet. Il s'agit également de systèmes capables de créer et transmettre des données afin de créer de la valeur pour ses utilisateurs à travers divers services (agrégation, analytique, etc.). Selon l'UIT (Union Internationale des Télécommunications), l'Internet des Objets est d'défini comme (une infrastructure mondiale pour la société de l'information, qui permet de disposer de services ´évolues en interconnectant des objets (physique ou virtuels) grâce aux technologies de l'information et de la communication interopérables existantes ou en évolution). Au fil du temps, le terme a évolué et il englobe maintenant tout l'écosystème des objets connectés. Cet écosystème englobe, des fabricants de capteurs, des éditeurs de logiciels, des opérateurs historiques ou nouveaux sur le marché, des intégrateurs, etc. Cet éclectisme en fait sa richesse. Inspire de [1], la figure (**Figure 1.1**) montre l'architecture passée, présente et future de l'IOT. A l'avenir, les appareils ne devraient pas seulement être connectés à Internet et a d'autres appareils locaux, mais devraient également communiquer directement avec d'autres appareils sur Internet. Outre les appareils ou les objets connectés, le concept d'IOT social (SIoT1) émerge également.

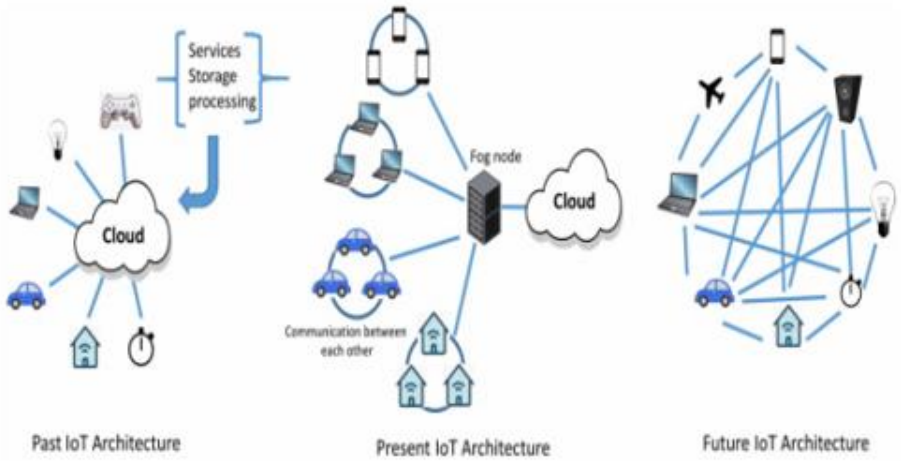


Figure 1.1 : Architecture actuelle et future de l'IOT [1]

1.2.2. Historique de l'Internet des Objets

Le terme « Internet of Things » (en Français Internet des Objets) est né en 1999 au centre MIT (Massachusetts Institute of Technologie), grâce à Kevin Ashton, un chercheur britannique, pionnier dans son domaine (IDO). Son équipe lança la promotion d'une connectivité ouverte de tous les objets en utilisant les étiquettes RFID (Radio Frequency Identification). Grâce à l'apparition du nouveau protocole IPv6, des secteurs comme l'aéronautique s'emparent rapidement du concept de l'Internet des objets, et participent aux recherches. Ce concept de l'Internet des Objets commence à connaitre une popularité en 2007. On a envisagé alors de mettre en place un Internet des Objets Global, Ubiquitaire [3].

Les différentes applications des technologies de l'Internet au monde des objets ont ainsi dépassé aujourd'hui le stade de concept. L'ère de la généralisation s'ouvre, avec des applications ciblées sur les entreprises, des produits. Les relations de machine à machine est un domaine pionnier mais des applications plus globales se développent comme celles visant à assurer, par exemple, la traçabilité des produits de luxe. Le déploiement d'applications est favorisé par les performances et le faible coût, d'une part des technologies d'échanges d'information issues de l'Internet ainsi que celles d'accès rapide aux informations de masse : informatique des nuages et Big-Data (Cloud Computing) et d'autre part des technologies optiques. Les experts et les utilisateurs, notamment au sein des entreprises, envisagent, non pas un Internet prolongé au monde physique, mais plutôt des applications et des systèmes s'intéressant aux objets en utilisant les technologies de l'Internet [3].

1.2.3. Importances et enjeux de l'Internet des Objets

La montée en puissance des applications de l'Internet des Objets peut s'observer dans plusieurs secteurs ou registres des activités sociales : des personnelles aux plus industrielles. Le large spectre des applications d'ores et déjà observables indique que nous sommes aujourd'hui face à une tendance bien ancrée.

Le point et l'intérêt économiques de certaines applications contribuent à stimuler les investissements de recherche et développement et à installer durablement les utilisations de l'Internet des Objets. Ensuite, par son caractère très global, l'Internet des Objets est porté par des mouvements profonds de la société : la convergence grandissante entre la communication en réseau et les systèmes d'information, le développement de la mobilité et la constitution d'environnements sociotechniques autonomes et centrés sur l'individu, le renforcement de la traçabilité et des processus de contrôle des activités et des personnes.

Ainsi, l'Internet des objets sous-tend à la fois un renforcement des outils de stimulation et de modélisation et amélioration des performances dans la réalité physique, grâce aux possibilités offertes dans la manipulation, le traitement et l'enrichissement des objets identifiés [5].

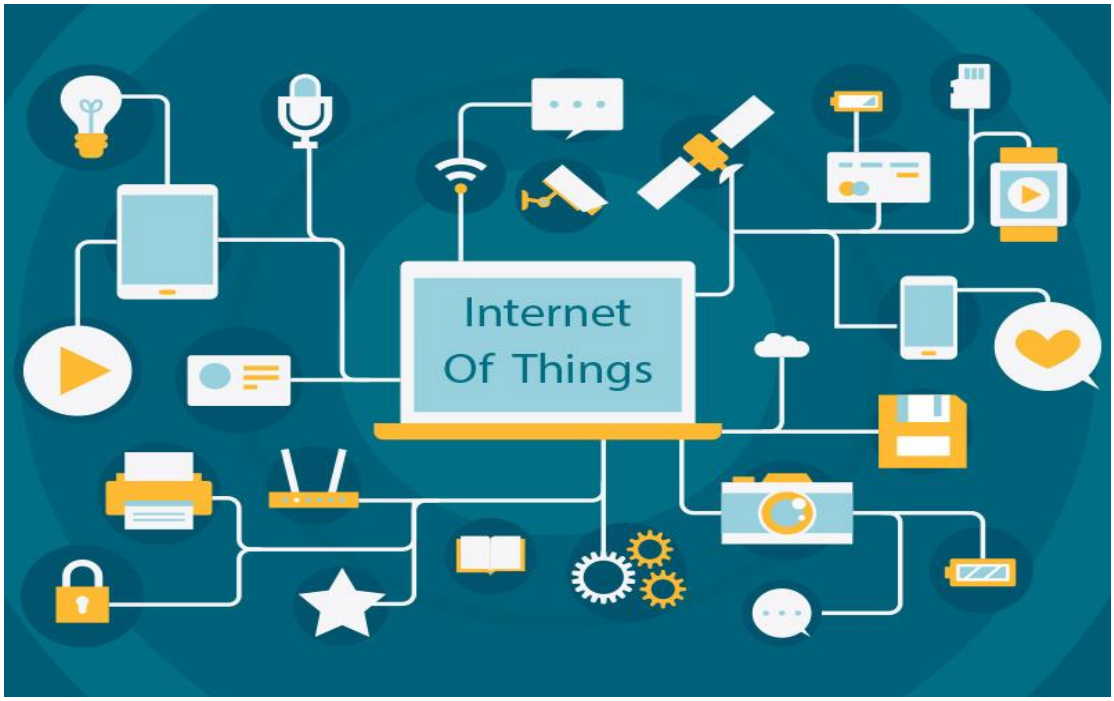


Figure 1.2 : Importances et enjeux de l'IOT [34]

Dans un cas, il construit des passerelles entre le monde de l'Internet et le monde réel, en connectant les objets et les informations qui les concernent. Dans le second cas, il prolonge les promesses de l'Internet et les systèmes d'informations existants en remplaçant l'observation et la saisie d'information par l'intégration même des objets dans le réseau. Cette convergence s'exprime aujourd'hui sous des appellations différentes (réalité augmentée, machines communicantes ou réseaux ubiquitaires) qui expriment la variété des registres dans lesquels se déploie l'Internet futur [5].

Pour comprendre l'importance et les enjeux associés à l'Internet des objets, il paraît utile de revenir sur certains traits saillants qui marquent ce mouvement vers l'Internet du futur. Il est important de garder à l'esprit que l'Internet du futur s'inscrit dans une trajectoire sociotechnique déjà ancienne. Les nouvelles directions où il se déploie restent marquées par certaines orientations initiales de l'Internet qui pèsent sur l'infrastructure et les configurations actuelles. Les choix d'aujourd'hui guideront, pour plusieurs années encore les trajectoires de développement, les structures de gouvernance ainsi que les usages de l'Internet des Objets [5].

1.2.4. Composant de l'Internet des Objets

Les composant IOT est cinq. L'objet connecte est d'abord un objet qui a une fonction mécanique et/ou électrique propre, il peut soit être conçu directement connectable, soit il est déjà existant et la connectivité est rajoutée a posteriori. L'objet connecte a pour fonction de collecter des données de capteurs, de traiter ces données et de les communiquer à l'aide de d'une fonction de connectivité et de recevoir des

instructions pour exécuter une action. Généralement ces fonctions de l'objet connecté n´nécessitent une source d'énergies, surtout quand les données sont prétraitées directement dans l'objet [2].

Figure 1.3 : Les Composantes de l'IOT [35]

- **Capteurs**

Les capteurs sont des dispositifs permettant de transformer une grandeur physique observée (température, luminosité, mouvement etc.) En une grandeur digitale utilisable par des logiciels. Il existe une très grande variété de capteurs de tous types, les objets connectés ont souvent la fonction de captation de ces grandeurs physiques sur leurs lieux d'utilisation .

Exemple de capteurs : lumière, présence, proximité, position, d´emplacement, accélération, rotation, température, humidité, son, vibration, ´électrique, magnétique, chimique, gaz, flux, force, pression, niveau [2].

- **Reseaux de capteurs**

Afin de satisfaire les besoins de communication entre eux, les capteurs sont « qui-prés de dispositifs sans fil pour l'émission et la réception de données. Cela ne suffit cependant pas à rendre un ensemble de capteurs accessibles ou du moins de m ânière inter-opérable, trans- parente et simplifiée Pour cela, les capteurs doivent aussi

s’organiser Ce qui c caractérise un réseau de capteurs, c’est que ses ´éléments sont de très petits appareils, dotés de capacités de transmission sans fil [6].

- **Énergie**

La plus importante contrainte à laquelle sont soumis les réseaux capteurs concernant l’Energie. L’autonomie temporelle des nœuds s’´évalué en termes d’années [7].

- **Actionneurs**

Les actionneurs sont des dispositifs qui transforment une donnée digitale en phénomène physique pour créer une action, ils sont en quelque sorte l’inverse du capteur. Exemple d’actionneurs : Afficheurs, Alarmes, Caméras, Hautparleurs, Interrupteurs, Lampes, Mo- tours, Pompes, Serrures, Vannes, Ventilateur, Vérins [2].

- **Connectivité**

La connectivité de l’objet est assurée par une petite antenne Radio Fréquence qui va permet- tré la communication de l’objet vers un ou plusieurs réseaux (qui sont détaillés dans la section réseaux IOT). Les objets pourront d’une part remonter des informations telles que leur identité, leur ´état, une alerte ou les données de capteurs, et d’autre part recevoir des informations telles que des commandes d’action et des données. Le module de connectivité permet aussi de gérer le cycle de vie de l’objet, c’est-`a-dire, l’authentification et l’enregistrement dans le réseau, la mise en service, la mise a` jour et la suppression de l’objet du réseau [2].

1.2.5. Technologies permutantes pour l'internet des objets

Technologies permutantes pour l'internet des objets Est une infrastructure mondiale société de l'information, permettant des avancées interconnecter des choses (physiques et virtuelles) basées sur et l'évolution de l'information et de la communication interopérables les technologies. Avec l'Internet des objets, la communication est étendue via Internet à toutes les choses qui nous entourent. L'Internet Les choses sont bien plus que de machine à machine communication, réseaux de capteurs sans fil, réseaux de capteurs, 2G/3G/4G, GSM, GPRS, RFID, WI-FI, GPS, microcontrôleur, microprocesseur etc. Ceux-ci sont considérés comme étant les technologies qui font des applications « Internet des objets » possible. Les technologies habilitantes pour l'Internet des objets sont prises en compte dans [1] et peuvent être regroupés en trois catégories : (1) technologies qui permettent aux « choses » d'acquérir l'information, (2) les technologies qui permettent des informations contextuelles, et (3) des technologies pour améliorer sécurité et confidentialité. Les deux premières catégories peuvent compris comme des blocs de construction fonctionnels nécessaires à la construction "l'intelligence" en "choses", qui sont en effet les caractéristiques qui différencier l'IOT de l'Internet habituel.

La troisième catégorie n'est pas une exigence fonctionnelle mais plutôt une exigence de fait, laquelle la pénétration de l'IoT serait [2] L'Internet des objets n'est pas une technologie unique, mais c'est mélange de différentes technologies matérielles et logicielles. le L'Internet des objets fournit des solutions basées sur de la technologie

de l'information, qui fait référence au matériel et logiciel utilisé pour stocker, récupérer et traiter des données et la technologie des communications qui comprend les systèmes électroniques utilisé pour la communication entre les individus Informatique, Mai 2016 6123 technologies permettantes pour L'IoT infrastructures pour le services avancés par et virtuel) des choses basées sur l'existant et l'évolution de l'information et de la communication interopérables Choses par lesquelles la communication est étendue Internet à toutes les choses qui nous entourent. Internet de Les choses sont bien plus que de machine à machine communication, réseaux de capteurs sans fil, réseaux de capteurs, FI, GPS, microcontrôleur, microprocesseur etc. Ceux-ci sont considérés comme étant le Applications "choses" Les choses sont considérées regroupés en trois catégories : (1) technologies qui permettent aux « choses » d'acquérir l'information, (2) les technologies qui permettent aux « choses » de traiter des informations contextuelles, et (3) des technologies pour améliorer Les deux premières catégories peuvent être conjointement compris comme des blocs de construction fonctionnels nécessaires à la construction "l'intelligence" en "choses", qui sont en effet les caractéristiques qui l'Internet habituel.

La troisième catégorie n'est pas une exigence fonctionnelle mais plutôt une exigence de fait, sans serait fortement réduit. Things n'est pas une seule technologie, mais c'est une différente technologie matérielle et logicielle. Le Things fournit des solutions basées sur l'intégration de la technologie de l'information, qui fait référence au matériel et logiciel utilisé pour stocker, récupérer et traiter des données et gy qui comprend les systèmes électroniques individus ou groupes. Il existe un mélange hétérogène de technologies de communication, qui doivent être adaptés afin de répondre aux besoins de l'IoT applications telles que l'efficacité énergétique, spé fiabilité. Dans ce contexte, il est possible que le niveau de la diversité sera réduit à un certain nombre une connectivité gérable les technologies qui répondent aux besoins des applications IoT, sont adoptés par le marché, ils ont déjà prouvé utilisable, soutenu par une solide alliance technologique. Des exemples de normes dans ces catégories incluent filaire et technologies sans fil comme Ethernet, WI GSM et GPRS. [1, 2] Les principales technologies génériques pour Internet présenté à la figure 3.

1.2.6. Domaines d'application de l’Internet des Objets

Il existe une panoplie de domaines d'applications pour les secteurs de l'internet des objets, du machine to machine et des objets connectés que ce soit dans le monde industriel ou dans la vie de tous les jours. Dans cet article seront présentés quelques applications des objets connectés, du machine to machine et de l'internet des objets [8].

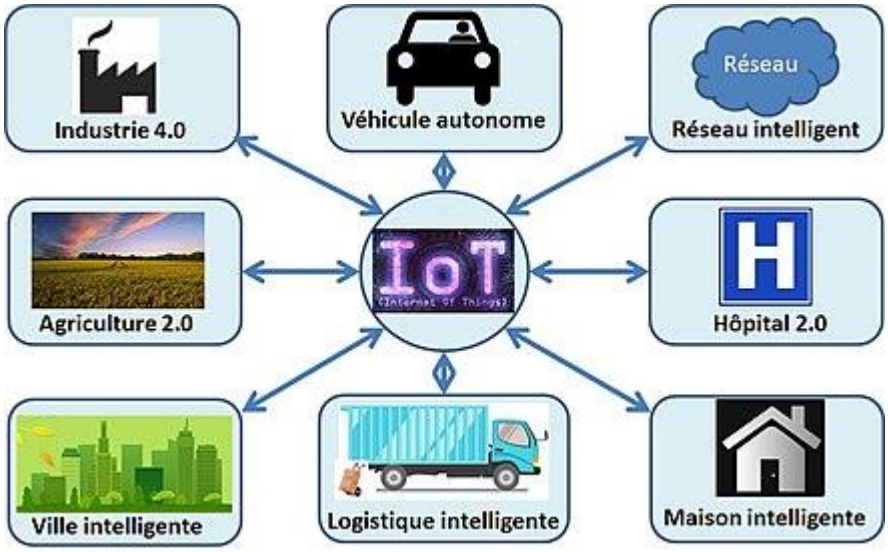


Figure 1.4 : Les domaines d'application de l'IOT [36]

- **Les applications médicales**

Dans le domaine santé, les objets connectés peuvent servir à plusieurs choses. On peut par exemple les utiliser comme dispositif de surveillance ou de monitoring sous forme de bracelet connecté ou de montre connectée qui permettront de suivre à tout moment nos activités physiques en nous informant de signes irréguliers.

Si vous êtes une personne qui prend des médicaments ces objets connectés peuvent également vous informer quand il est temps de prendre vos médicaments En outre, cela pourrait éventuellement informer votre médecin d'une situation d'urgence lui permettant ainsi de vous localiser grâce à votre appareil.

Je pense que pour les personnes âgées l'automatisation des procédures de prises en charge médicales pourrait être précieuse. Supposons par exemple qu'une personne âgée soit tombée hasardeusement et qu'elle soit incapable de se lever. Elle peut avoir de sérieux problèmes si elle vit seule sans assistance.

Maintenant, imaginez qu'elle ait porté un dispositif qui pourrait alerter son entourage ou ses proches quand la personne reste inactive après X temps pendant la journée. Le dispositif connecté pourra avertir automatiquement le soignant ou médecin désigné par la suite. Ainsi, les applications des objets connectés dans le monde médical sont infinies [8].

- **Bien-être et le confort**

La domotique ou la maison intelligente est un classique. Imaginez un instant que votre thermostat soit capable de se mettre en marche tout seul en fonction de l'emplacement de votre voiture vous permettant de vous réchauffer une fois rentrer à la maison. Aussi, imaginez que votre réfrigérateur vous informe lorsque vous aurez besoin

d'acheter du lait ou qu'il soit capable de créer une liste d'achats personnalisée en fonction de vos articles les plus achetés. Ou encore vous dire quand votre nourriture est sur le point de périmer !

Futuriste ? Oui. Imaginatif ? Oui. Faisable ? C'est seulement une question de temps, le futur nous le dira. Ce qui est important ici c'est la faculté de ces appareils à communiquer entre eux pour fournir à l'utilisateur tout le confort et le bien être dont il a besoin [8].

- **Applications industrielles**

Dans le monde industriel le machine to machine ou M2M peut accroitre considérablement la performance et la productivité d'une usine. Par exemple supposons que l'on ait un compteur intelligent et connecté en amont de notre ligne de production et que ce compteur recense et répertorie en temps réel l'ensemble des pièces disponibles en stock dans une base de données, en cas de pénurie, il pourra avertir et envoyer automatiquement un message au système de GPAO qui se chargera à son tour d'effectuer par exemple une opération d'approvisionnement. Dans ce cas-ci, la problématique sera de faire communiquer tous les dispositifs de sorte que le processus soit la plus autonome possible [8].

1.2.7. Avantages et inconvénients de l'Internet des Objets

- **Avantages de l'IoT**

- Il peut aider à contrôler plus intelligemment les maisons et les villes via les téléphones portables. Il améliore la sécurité et offre une protection personnelle.
- En automatisant les activités, cela nous fait gagner beaucoup de temps

L'information est facilement accessible, même si nous sommes loin de notre emplacement réel, et elle est mise à jour fréquemment en temps réel.

- Les appareils électriques sont directement connectés et communiquent avec un ordinateur de contrôle, tel qu'un téléphone portable, ce qui permet une utilisation efficace de l'électricité. En conséquence, il n'y aura pas d'utilisation inutile d'équipements électriques.
- Une assistance personnelle peut être fournie par les applications IoT, qui peuvent vous alerter sur vos plans réguliers.
- Il est utile pour la sécurité car il détecte tout danger potentiel et avertit les utilisateurs. Par exemple, GM OnStar est un dispositif intégré à ce système qui identifie un accident de voiture ou un accident sur la route. Il passe immédiatement un appel en cas d'accident ou de crash.
- Il minimise l'effort humain car les appareils IoT se connectent et communiquent entre eux et effectuent une variété de tâches sans intervention humaine.
- Les soins aux patients peuvent être effectués plus efficacement en temps réel sans avoir besoin d'une visite chez le médecin. Cela leur donne la possibilité de faire des choix et de fournir des soins fondés sur des données probantes.

- Le suivi des actifs, le suivi du trafic ou du transport, le contrôle des stocks, la livraison, la surveillance, le suivi des commandes individuelles et la gestion des clients peuvent tous être rendus plus rentables avec le bon système de suivi.

Les inconvenient de l'IoT

- Les pirates peuvent accéder au système et voler des informations personnelles. Étant donné que nous ajoutons autant d'appareils à Internet, il existe un risque que nos informations soient utilisées à mauvais escient.
- Ils dépendent fortement d'Internet et sont incapables de fonctionner efficacement sans lui.
- Avec la complexité des systèmes, il existe de nombreuses façons pour eux d'échouer.
- Nous perdons le contrôle de nos vies – nos vies seront entièrement contrôlées et dépendantes de la technologie.
- La surutilisation d'Internet et de la technologie rend les gens inintelligents, car ils dépendent d'appareils intelligents au lieu de faire un travail physique, ce qui les rend paresseux.
- Les travailleurs non qualifiés courent un risque élevé de perdre leur emploi, ce qui pourrait conduire au chômage. Des caméras de surveillance intelligentes, des robots, des systèmes de repassage intelligents, des machines à laver intelligentes et d'autres installations remplacent les agents de sécurité, les femmes de chambre, les ferrailleurs et les services de nettoyage à sec, etc.
- Il est très difficile de planifier, de créer, de gérer et d'activer une large technologie dans le cadre de l'IOT [9].

Figure 1.5 : Avantages et inconvenient de IOT [37]

1.2.8. Conclusion

Ce chapitre nous a permis d'avoir un aperçu de l'Internet des objets, de son histoire et de son importance, de ses composants, de ses fonctions, de ses domaines d'application, ainsi que de ses technologies permutantes. En notant les avantages et les inconvénients de l'Internet des objets. Dans le chapitre suivant, nous expliquerons Architecture logicielle et évolution logicielle.

Chapitre 2 : Architecture logicielle et évolution logicielle

2.1 Introduction

Dans ce chapitre, nous expliquons la définition, l'importance de l'architecture lors d'un développement de logiciel, développement pour et par la réutilisation. En plus nous expliquons la définition, la nécessité de l'évolution logiciel.

2.2 Architecture logicielle

L'architecture logicielle décrit d'une manière symbolique et schématique les différents éléments d'un ou de plusieurs systèmes informatiques, leurs interrelations et leurs interactions. Contrairement aux spécifications produites par l'analyse fonctionnelle, le modèle d'architecture, produit lors de la phase de conception, ne décrit pas ce que doit réaliser un système informatique mais plutôt comment il doit être conçu de manière à répondre aux spécifications. L'analyse décrit le « quoi faire » alors que l'architecture décrit le « comment le faire ».

2.2.1 Définition des architectures de logicielle

L'architecture logicielle détermine la manière dont les programmes sont conçus, c'est-à-dire la manière dont les différents éléments qui le composent sont agencés, leur permettant de fonctionner. Tout comme l'architecture, le système mécanique ou le système informatique d'un bâtiment, il est essentiel que chaque pièce soit en place et puisse jouer son rôle afin que l'ensemble fonctionne de manière efficace et efficiente. La structure du programme varie selon son utilisation.

En plus des performances du programme, de nombreux autres éléments doivent être pris en compte dans la conception de l'architecture. En fait, le produit final doit être gérable, facile à utiliser, suffisamment flexible et évolutif pour permettre les futures versions. Il doit également être fiable et compatible avec d'autres outils potentiellement plus anciens ou plus récents. Enfin, les coûts de construction et d'exploitation doivent également être abordables.

2.2.2 Importance de l'architecture lors d'un développement de logiciel

La conception de l'architecture est une phase particulièrement importante du développement d'un logiciel. Elle conditionne sa **s**tabilité**,** son efficacité et sa pérennité. Au contraire, certaines applications peuvent connaitre des faiblesses dues à une architecture mal pensée, pas ou plus adaptée au contexte.

Si la pression du time-to-market pèse sur le développement d'un logiciel, elle pèse donc également sur la conception de son architecture. Sachez tout de même qu'une fois le projet commencé, s'agissant d'un élément aussi structurel, il est dangereux voire impossible d'en changer**.**

Cela étant dit, il n'est pas si fréquent de trouver de « mauvaises » architectures dans l'absolu, mais on observe souvent des architectures qui ne sont pas parfaitement adaptées au contexte du projet de développement. Car l'architecture logicielle est avant tout issue d'un compromis entre les exigences techniques, opérationnelles et fonctionnelles qui entourent l'application. Et c'est là où l'architecte logiciel devra exercer son savoir-faire et disposer d'expériences suffisantes [38].

2.2.3 Développement pour et par la réutilisation

La réutilisation de composants logiciels est l'activité permettant de réaliser les économies les plus substantielles, encore faut-il posséder des composants à réutiliser. De plus, la réutilisation de composants nécessite de créer une architecture logicielle permettant une intégration harmonieuse de ces composants. Le développement par la réutilisation logicielle impose donc un cycle de production-réutilisation perpétuel et une architecture logicielle normalisée.

Une réutilisation bien orchestrée nécessite la création et le maintien d'une bibliothèque logicielle et un changement de focus ; créer une application revient à créer les composants de bibliothèque nécessaires puis à construire l'application à l'aide de ces composants. Une telle bibliothèque, facilitant le développement d'application est un Framework (cadriciel) d'entreprise et son architecture, ainsi que sa documentation sont les pierres angulaires de la réutilisation logicielle en entreprise.

Le rôle de l'architecte informatique se déplace donc vers celui de bibliothécaire. L'architecte informatique doit explorer la bibliothèque pour trouver les composants logiciels appropriés puis créer les composants manquants, les documenter et les intégrer à la bibliothèque. Dans une grande entreprise, ce rôle de bibliothécaire est rempli par l'architecte informatique en chef qui est responsable du développement harmonieux de la bibliothèque et de la conservation de l'intégrité de son architecture [10].

2.2.3 Les modèles de conception dans l'architecture logicielle

Les modèles de conception sont des solutions éprouvées et réutilisables par rapport à un problème fréquent, considérées comme bonnes pratiques et pas spécifiques à un langage de programmation en particulier. Imaginons une application ayant besoin d'utiliser un ensemble complexe d'API, avec des appels à de nombreuses procédures différentes, l'utilisation d'un modèle 'façade' fournit une interface plus simple et plus uniforme. C'est une couche de code intermédiaire qui appelle les API appropriées**.**

Au lieu de créer vous-même l'architecture logicielle de bout en bout, il est possible de lier plusieurs modèles de conception existants ou de vous en inspirer.

Ces modèles, pourtant ni savants ni compliqués, mais issus des bonnes pratiques au fil du temps, élargissent les possibilités de votre application et améliorent la qualité des développements [39].

Généralement associés à une programmation orientée objet, il existe trois modèles de conception distincts :

- **Modèles de création**

Les modèles de création permettent une représentation simplifiée du processus pour certaines instances, en d'autres termes, ils séparent le développement des objets complexes de leur représentation. Par exemple, le modèle « Singleton » est utilisé pour créer une classe de base qui n'aura qu'une seule instance [39].

- **Modèles de structure**

Les modèles de structure sont prêts à l'emploi pour les relations entre les classes. Par exemple, le modèle « Données de classe privées » est utilisé pour limiter l'accès à une classe spécifique et ainsi prévenir la modification non souhaitée d'un objet. Le modèle « Décorateur », permet quant à lui, d'ajouter des fonctionnalités aux classes existantes.

Le modèle 'Façade' évoqué plus haut entre dans cette catégorie [39].

- **Modèles de comportements**

Les modèles de comportement permettent de modéliser les comportements d'un logiciel, notamment la manière dont ils communiquent entre eux. Les comportements sont définis via des abstractions avec des responsabilités bien spécifiques.

Les modèles de conception sont des outils utiles pour la programmation, il est aussi possible de s'en inspirer pour créer une application au plus près du besoin. Il ne faut pas pour autant s'interdire de chercher de nouvelles solutions qui peuvent être plus efficaces [39].

2.3 Évolution logicielle

2.3.1 Définition Évolution logicielle

L'évolution du logiciel est le développement continu d'un logiciel après sa sortie initiale pour répondre aux exigences changeantes des parties prenantes et/ou du marché. L'évolution des logiciels est importante car les organisations investissent de grosses sommes d'argent dans leurs logiciels et sont complètement dépendantes de ces logiciels. L'évolution des logiciels aide les logiciels à s'adapter aux exigences changeantes des entreprises, à corriger les défauts et à s'intégrer à d'autres systèmes changeants dans un environnement de système logiciel.

2.3.2 Nécessité de l'Évolution logiciel :

L'évaluation du logiciel n'est nécessaire que pour les raisons suivantes [11] :

a) Changement des exigences au fil du temps : avec le temps, les besoins de l'organisation et la façon dont l'entreprise fait des affaires peuvent changer radicalement, donc dans cette période en constante évolution, les outils (logiciels) qu'ils utilisent doivent changer pour maximiser les performances.

b) Changement d'environnement : À mesure que l'environnement de travail modifie les choses (outils) qui nous permettent de travailler dans cet environnement, il change également en termes relatifs : Le nouvel environnement.

c) Erreurs: À mesure que l'âge des logiciels déployés au sein d'une organisation augmente sa précision ou sa perfection et diminue également l'efficacité pour supporter la charge de travail de plus en plus complexe. Par conséquent, dans ce cas, il devient nécessaire d'éviter d'utiliser des programmes obsolètes et obsolètes. Tous ces logiciels hérités doivent subir un processus d'évolution afin de devenir robustes en fonction de la complexité de la charge de travail dans l'environnement actuel.

d) Risques de sécurité : L'utilisation de logiciels obsolètes au sein d'une organisation peut conduire à une variété de cyberattaques logicielles et peut exposer vos données confidentielles illégalement associées au logiciel utilisé. Par conséquent, il devient nécessaire d'éviter de telles failles de sécurité grâce à une évaluation régulière des correctifs/modules de sécurité utilisés dans le logiciel. Si le logiciel n'est pas assez puissant pour résister aux cyberattaques en cours, il doit être changé (mis à jour).

e) Pour les nouvelles fonctionnalités et caractéristiques : afin d'augmenter les performances, le traitement rapide des données et d'autres fonctions, l'organisation doit constamment développer le logiciel tout au long de son cycle de vie afin que les parties prenantes et les clients du produit puissent fonctionner efficacement.

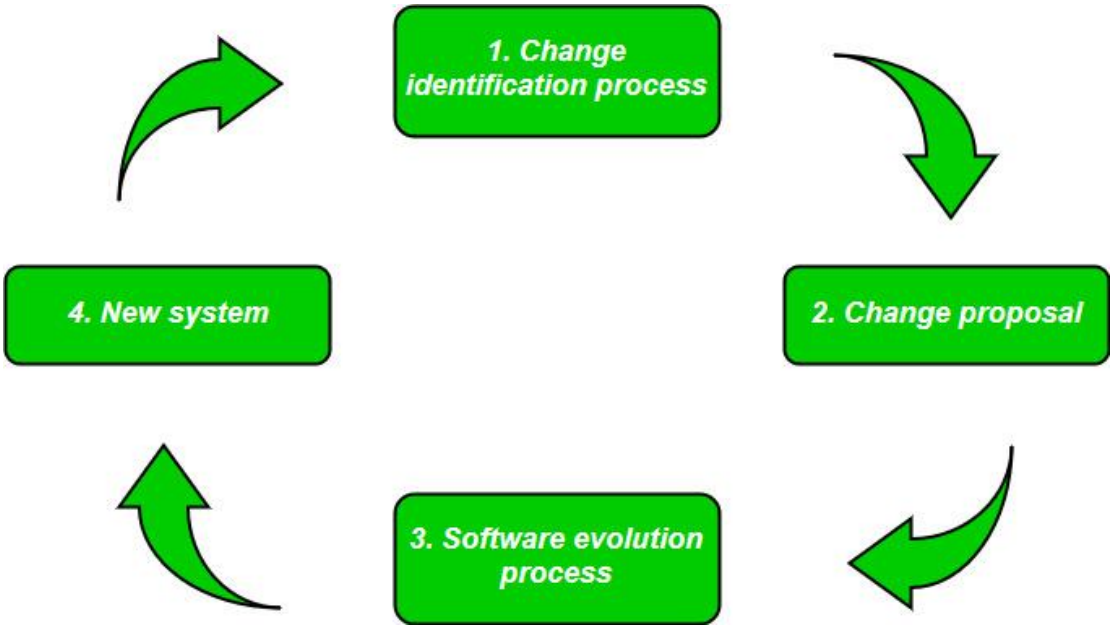


Fig. 2.2 Lois utilisées pour l'évolution du logiciel [11]

1) Loi du changement continu :

Cette loi stipule que tout système logiciel qui représente une réalité du monde réel subit des changements continus ou devient progressivement moins utile dans cet environnement.

2) Loi de complexité croissante :

Au fur et à mesure qu'un programme évolue, sa structure devient plus complexe à moins que des efforts efficaces ne soient faits pour éviter ce phénomène.

3) Loi de conservation de la stabilité de l'organisation :

Au cours de la durée de vie d'un programme, le taux de développement de ce programme est à peu près constant et indépendant des ressources consacrées au développement du système.

4) Loi de conservation de la familiarité :

Cette loi stipule que pendant la durée de vie active du programme, les modifications apportées dans la version successive sont presque constantes.

2.3.3 Évolution de l'architecture logicielle

L'évolution ne s'est peut-être pas seulement produite dans la section informatique. L'évolution se produit dans tous les autres domaines, même l'humain a une évolution. Donc, dans cet article, je vais vous expliquer l'évolution de l'architecture logicielle et je vous emmènerai des systèmes autonomes aux micro services.

- **Architecture à un niveau**

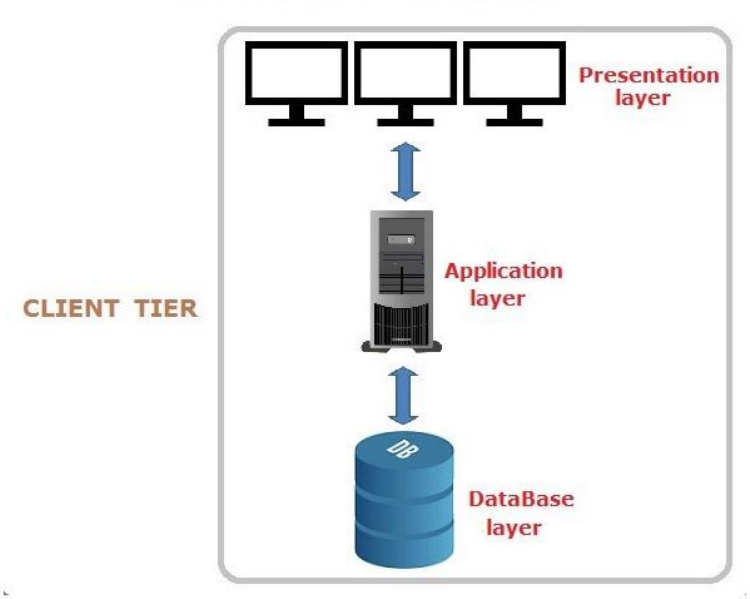


Fig. 2.3 Architecture à un niveau [40]

L'application One Tier est également connue sous le nom d'application autonome, qui est l'architecture la plus simple. Cela équivaut à exécuter l'application sur un ordinateur personnel. Ainsi, tous les composants nécessaires à l'exécution d'une application se trouvent sur une seule application ou un seul serveur. Ainsi, la couche de présentation, la couche de logique métier (application) et la couche de données sont toutes situées sur une seule machine ou un progiciel. Certains des exemples d'architecture à un niveau sont le lecteur MP3 et MS Office. Ainsi, dans une architecture à un niveau, les données peuvent être stockées dans le système local ou sur un lecteur partagé. En parlant de certains des avantages de ce type de systèmes,

1) C'est simple et facile à mettre en oeuvre.
2) Très efficace.
3) Aucune compatibilité et aucun problème de changement de contexte

Quand on pense à certains des inconvénients,

1. Puisqu'il ne s'agit que d'une seule machine, il ne prend pas en charge l'accès distant/distribué aux ressources de données.

❖ **Architecture à deux niveaux**

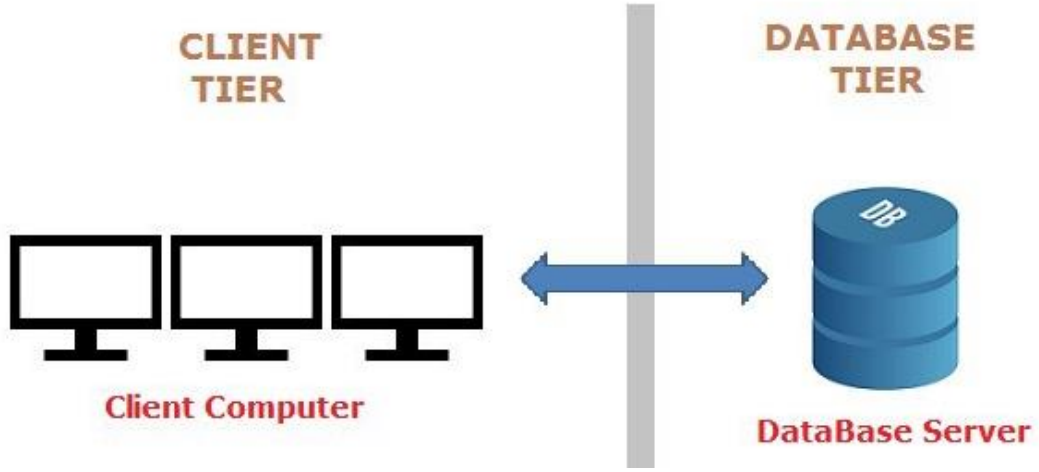


Fig. 2.4 Architecture à deux niveaux [40]

Après l'architecture à un niveau, nous sommes passés à l'architecture à deux niveaux. L'architecture à deux niveaux est également connue sous le nom d'application client-serveur. Cette architecture est donc divisée en deux parties. Ce sont,

1. Application client (niveau client)

2. Base de données (niveau de données)

Le système client gère à la fois la couche présentation et la couche métier (couche application) et le système serveur gère la couche base de données. Donc,

fondamentalement, ce qui se passe, c'est que le système client envoie la demande au système serveur et que le système serveur traite la demande et renvoie les données au système client. La communication a donc lieu entre le client et le serveur. Quand on considère les avantages,

Facile à entretenir et la modification est un peu facile.

Communication plus rapide.

Certains des inconvénients sont,

Les performances de l'application seront réduites lors de l'augmentation du nombre d'utilisateurs.

- **Architecture à trois niveaux**

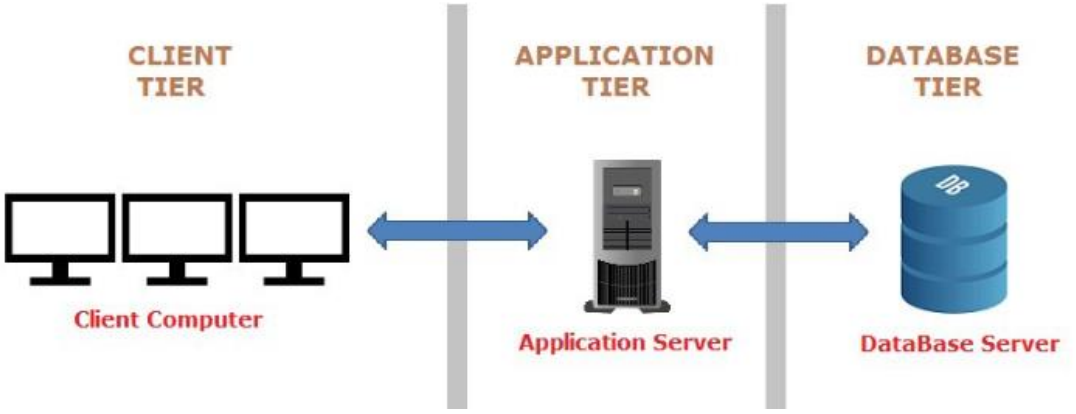


Fig. 2.5 Architecture à trois niveaux [40]

Après l'architecture à deux niveaux, nous sommes passés à l'architecture à trois niveaux. L'application à trois niveaux est également connue sous le nom d'application Web. L'architecture à trois niveaux est divisée en trois parties :

1. Couche de présentation (niveau client)

2. Couche application (niveau entreprise)

3. Couche de base de données (niveau de données)

Dans une architecture à trois niveaux, le système client gère la couche de présentation, le serveur d'application gère la couche d'application et le système serveur gère la couche de base de données. Voyons maintenant quels sont les avantages d'une architecture à trois niveaux,

1.La sécurité est élevée car le client n'a pas d'accès direct à la base de données.

2. Évolutivité élevée car chaque couche s'exécute sur un serveur différent

Certains des inconvénients sont,

1.Structure complexe

2.4 Conclusion

Ce chapitre nous a permis d'avoir un aperçu de l’architecture logiciel, de sa définition et de son importance de l’architecture lors d’un développement de logiciel, et le développement de logiciel et développer pour et par la réutilisation en plus nous avons fait un aperçu sur évolution de logiciel de quotté définition et nécessité.

Chapitre 3 : Travaux antérieurs dans le domaine de la modélisation et de réingénierie des IOT

3.1 Introduction

Dans ce chapitre, nous présentons quelques travaux effectués par certains développeurs dans le domaine de la modélisation de l'ingénierie des systèmes de l'Internet des objets et de la reconversion des schémas d'ingénierie au code du système de l'Internet des objets.

3.2 Modélisation et génération de code pour l'IOT

3.2.1 Framework de modélisation et de génération de code pour l'IOT

La plupart des systèmes dans leur développement et leur développement dépendent de réutilisation de Framework open source. En développant le système là où il peut vérifier le processus d'analyse obtenu et renforcer la communication entre le système les intervenants dans un environnement communicatif.

CAPS réalisés pour modéliser et analyser les architectures des choses. Adoption du Framework ThingML l'idée de faciliter la génération de code pour les systèmes IOT. Objectifs de notre approche lorsque vous couvrez toute la chaîne de modélisation et d'analyse à l'aide de CAPS, la mise en œuvre est la puissance du code d'objet. Cet article propose un cadre de génération de code cadre de modélisation CAPS. Convertissez un modèle SAML-CAPS en modèle ThingML. Le modèle SAML représente CAPS.

Le passage de SAML à ThingML en utilisant le Framework de génération de code CAPSML. L'objectif de CAPSML est de permettre aux développeurs IOT d'apaiser leurs inquiétudes apprenez les langages de programmation qui implémentent leurs systèmes IOT.

Conditions. Les composants peuvent échanger des données en échangeant des messages via ports de messagerie. La connectivité détermine la communication entre les composants. Identifie la connexion source et ports cibles. Les ports d'entrée et de sortie sont l'interface pour composant. Pour plus d'informations sur SAML, consultez le fichier travail complet.

3.2.2 Contexte de Caps

CAPS est un Framework de modélisation, dispose d'un outil pour l'ingénierie des bouchons des cyber systèmes physiques (CPS) [12], [13], [14] Offre un cadre de modélisation riche qui mène à une classe Intérêts entre différentes perspectives de modélisation. Conçu et Il a été mis en œuvre en tenant compte de trois points de

Vue architecturale : Présentation structurée et comportementale de l'architecture logicielle (SAML), Vue matérielle (HWML), vue de l'espace physique (SPML) [15].

Dans ce travail, nous nous concentrerons sur le rendu SAML pour Performances de génération de code. Le modèle SAML se compose principalement d'un ensemble de composants logiciels et d'interfaces. Un composant est une unité arithmétique avec un état interne Une interface spécifique. Chaque composant peut contenir plusieurs Modèles qui décrivent son comportement. Comportement des modes Désigné par un ensemble d'événements, de procédures et de conditions. Les composants peuvent échanger des données en échangeant des messages via ports de messagerie. La connectivité détermine la communication entre les composants. Identifie la connexion source et ports cibles. Les ports d'entrée et de sortie sont l'interface pour composant. Pour plus d'informations sur SAML, consultez le fichier travail complet [15].

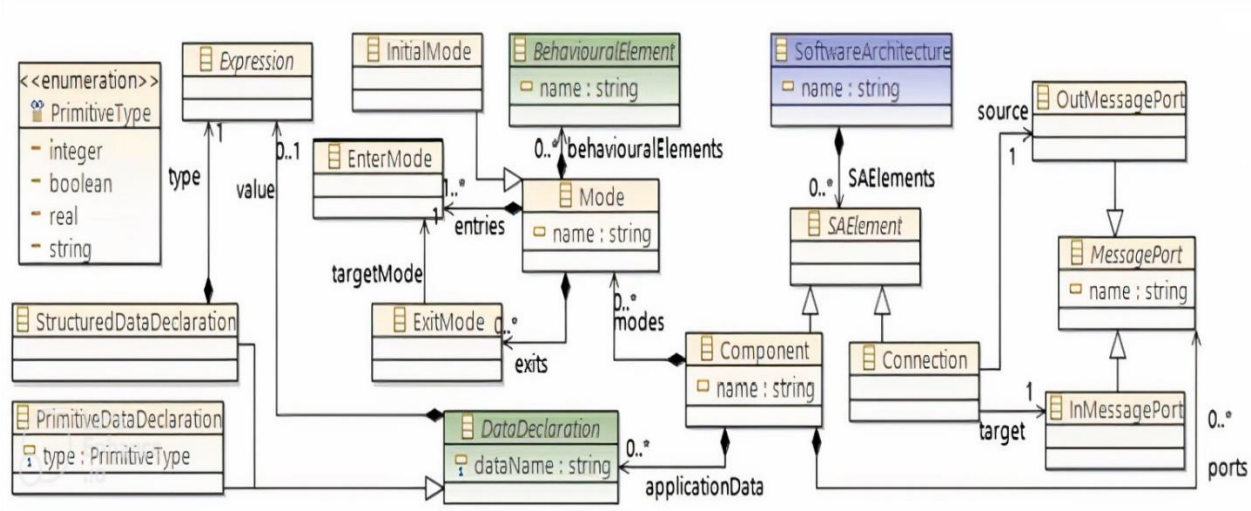


Fig. 3.1 SAML metamodel: structural concepts [15]

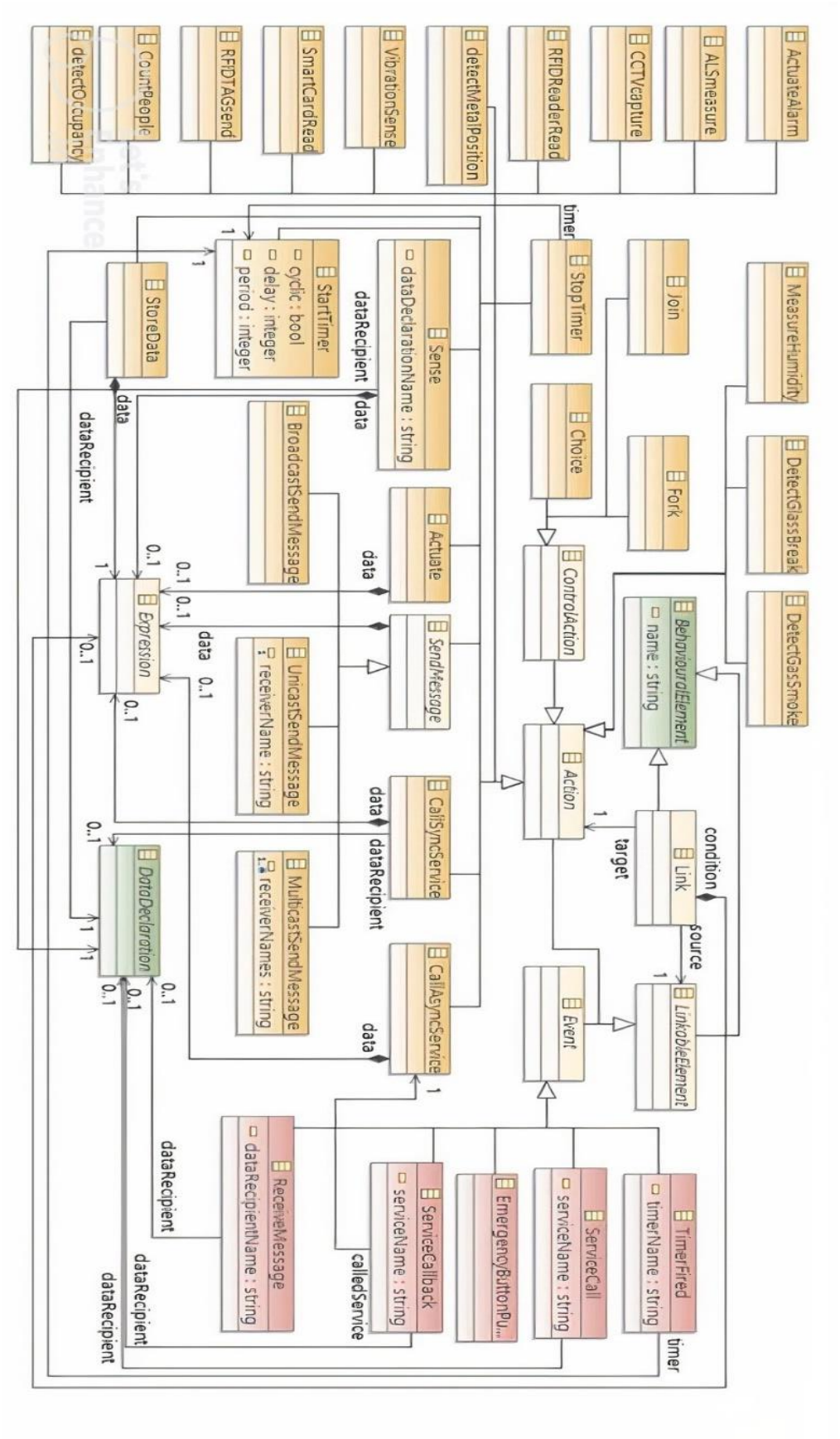


Fig. 3.2 SAML metamodel: behavioral concepts [15]

3.2.3 Contexte de ThingML

Dans ce langage de modélisation d'assemblage ThingML Construire des modèles – pour la conception et la mise en œuvre systèmes de IOT [16]. Objectifs de ThingML compris un groupe de traducteurs capables de Convertissez le modèle ThingML en code entièrement fonctionnel, exécutez Divers langages (tels que C, Java et Javascript), prêts à être construits

ThingML est basé sur deux structures de base : la chose qui représente le composant du programme et la configuration qui décrit l'interconnexion des choses. La chose est Une unité d'exécution, également appelée processus ou composant. Définit les propriétés (variables), les fonctions, les messages, les ports et groupe de pays. De plus, le comportement de la chose est illustré Utiliser la synchronisation des états composés qui sont exprimés Utilisez le mode action D'état d'événement (ECA).

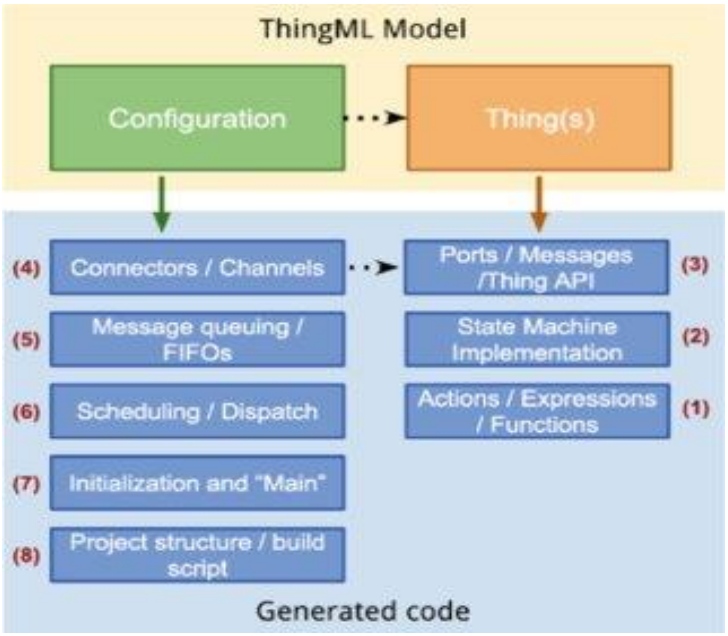


Fig. 3.3 ThingML model [16]

3.2.4 Framework de génération de code CAPSml

CAPSml qui vise à convertir le modèle SAML en CAPS vers le langage ThingML. Le processus de conversion démarre en CAPSml à partir des fichiers ecore et xmi SAML. Puis à travers Plusieurs modèles de transformation de texte, made in Acceleo1, résumé Le formulaire SAML est mappé au contenu du formulaire ThingML. Finalement, Le fichier ThingML est généré automatiquement et peut être importé directement Dans le cadre de ThingML.Le cadre de génération de code CAPSml passe par quatre étapes de transformation de modèle.

Ces étapes sont décrites dans les sections suivantes :

- **Phase de préparation**

À ce stade, nous définissons l'URI du méta-modèle SAML, voir la ligne 2 de la figure. 4.3 Ensuite, Nous fixons le point de départ de la transformation à la couche supérieure dans SAML, qui est la classe de génie logiciel indiquée à la ligne 4 de la figure 4.3. De cette classe de maître

Nous pouvons parcourir progressivement tous les éléments du programme décrits dans le profil SAML Modèle. De plus, nous spécifions le nom cible du fichier à convertir en Résultats (CAPS. Thingml), voir ligne 7 Figure. 4.3 Enfin, nous importons une bibliothèque privée dans Langage ThingML requis pour les définitions de type de données, voir ligne 8 Figure 4.3

```
1 [comment encoding = UTF-8 /]
2 [module main('http://ualberta.ssrg.components')/]
3
4 [template public main(element : SoftwareArchitecture)]
5  [comment @main /]
6
7 [file ('CAPS.thingml', false, 'UTF-8')]
8 import "datatypes.thingml" from stl
9
```

Figure. 3.4 Code génération préparation [41]

- **Phases de conversion des composants**

À ce stade, nous mappons le composant avec ses éléments en SAML à la chose et leur correspondance dans ThingML. La conversion commence à partir du mappage Le composant dans l'objet, voir Figure 6.3. Fondamentalement, tout composant dans SAML a Déclarations de données primitives, ports et modèles. Il est converti comme suit :

- **Déclarations d'instructions primitives**

Il s'agit de variables utilisées localement lors d'opérations au sein du composant. Voir figure. 7.3. Il est important de mentionner Que le type de données réel dans CAPS est converti en un type de données flottant dans ThingML.

Nous avons créé un type spécial de chose, appelé la pièce d'occasion
Définit toutes les variables qui seront utilisées dans l'échange des messages configurés. Ainsi, chaque composant contenant un message qui sera envoyé ou reçu contiendra un fichier Partie pour vous d'utiliser les valeurs de message. Conversion de messages Les variables sont présentées dans la figure. 3.5

```
25  // ***** Variables *****
26  [for (var : PrimitiveDataDeclaration | applicationData->
       filter(PrimitiveDataDeclaration)->asSequence())]
27  property [var.dataName/] : [if (var.type.toString()
          .equalsIgnoreCase('real'))]Float[else]
           [var.type.toString().toUpperFirst()/][/if]
28  [/for]
29
```

Figure 3.5 Lettres de décalage variables [41]

```
22 [for (comp : Component | Components)][comment building Things /]
23 thing [comp.name/] includes Messages {
24
```

Figure. 3.6 Component transformation [41]

```
10 thing fragment Messages {
11  [for (messComp : Component | Components)]
12   [for (dataDec : PrimitiveDataDeclaration | applicationData->
        filter(PrimitiveDataDeclaration)->asSequence())]
13    [if (dataDec.type.toString().equalsIgnoreCase('real'))]
14  message [dataDec.dataName/] (value : Float)
15    [else]
16  message [dataDec.dataName/] (value : [dataDec.type.toString()
                                          .toUpperFirst()/])
17    [/if]
18   [/for]
19  [/for]
20 }
```

Figure. 3.7 Data déclarations variables transformation [41]

- **Ports**

Ils sont utilisés comme interface avec le composant. Chaque ingrédient *de SAML* à zéro ou plusieurs ports de message. Ces ports de message peuvent être *InMessagePort* ou OutMessagePort. Connectez *OutMessagePort* en tant que fichier Source à InMessagePort en tant que cible. Ces ports peuvent recevoir quatre ports différents Les types de message sont *UnicastSendMessage*, *BroadcastSendMessage*, *MulticastSendMessage* ou *ReceiveMessage*. InMessagePort est défini dans SAML au « port demandé » dans ThingML, et OutMessagePort est défini sur le « port disponible » dans ThingML. En cas Diffusion, nous spécifions uniquement les données qui seront transmises à partir de "Fournir le port "in" envoie et atteindra tous

les ports connectés. Sinon, dans un message unicast, nous spécifions les données à envoyer dans "*send*" et un destinataire Les données dans le "reçoit".

Enfin, si le port reçoit un message, alors Nous définissons le nom du port dans "Required Port" et spécifions les données requises Reçu dans "reçoit". La figure 8 montre les conversions de ports saisies Considérez les différents types de messages.

```
41  // ***** Ports *****
42  [for (modePorts : Mode | modes)]
43   [for (it : BehaviouralElement | behaviouralElements)]
44    [if (oclIsKindOf(UnicastSendMessage))]
45     [for (mess : MessagePort | it->
          filter(UnicastSendMessage).toMessagePorts)]
46  provided port [mess.name/] {
47   sends [it->filter(UnicastSendMessage).dataRecipient.dataName/]
48   receives [it->filter(UnicastSendMessage).receiverName/]
49  }
50     [/for]
51    [elseif (oclIsKindOf(BroadcastSendMessage))]
52     [for (mess : MessagePort | it->
          filter(BroadcastSendMessage).toMessagePorts)]
53  provided port [mess.name/] {
54   sends [it->filter(BroadcastSendMessage).dataRecipient.dataName/]
55  }
56     [/for]
57    [elseif (oclIsKindOf(MulticastSendMessage))]
58     [for (mess : MessagePort | it->
          filter(MulticastSendMessage).toMessagePorts)]
59  provided port [mess.name/] {
60   sends [it->filter(MulticastSendMessage).dataRecipient.dataName/]
61      [for (receivers : String | it->
           filter(MulticastSendMessage).receiverNames->asSequence())]
62   receives [receivers/]
63      [/for]
64  }
65     [/for]
66    [elseif (oclIsKindOf(ReceiveMessage))]
67     [for (mess : MessagePort | it->
          filter(ReceiveMessage).fromMessagePorts)]
68  required port [mess.name/] {
69   receives [it->filter(ReceiveMessage).dataRecipient.dataName/]
70  }
71     [/for]
72    [/if]
73   [/for]
74  [/for]
```

Figure 3.8. Ports transformation [41]

- **Patterns**

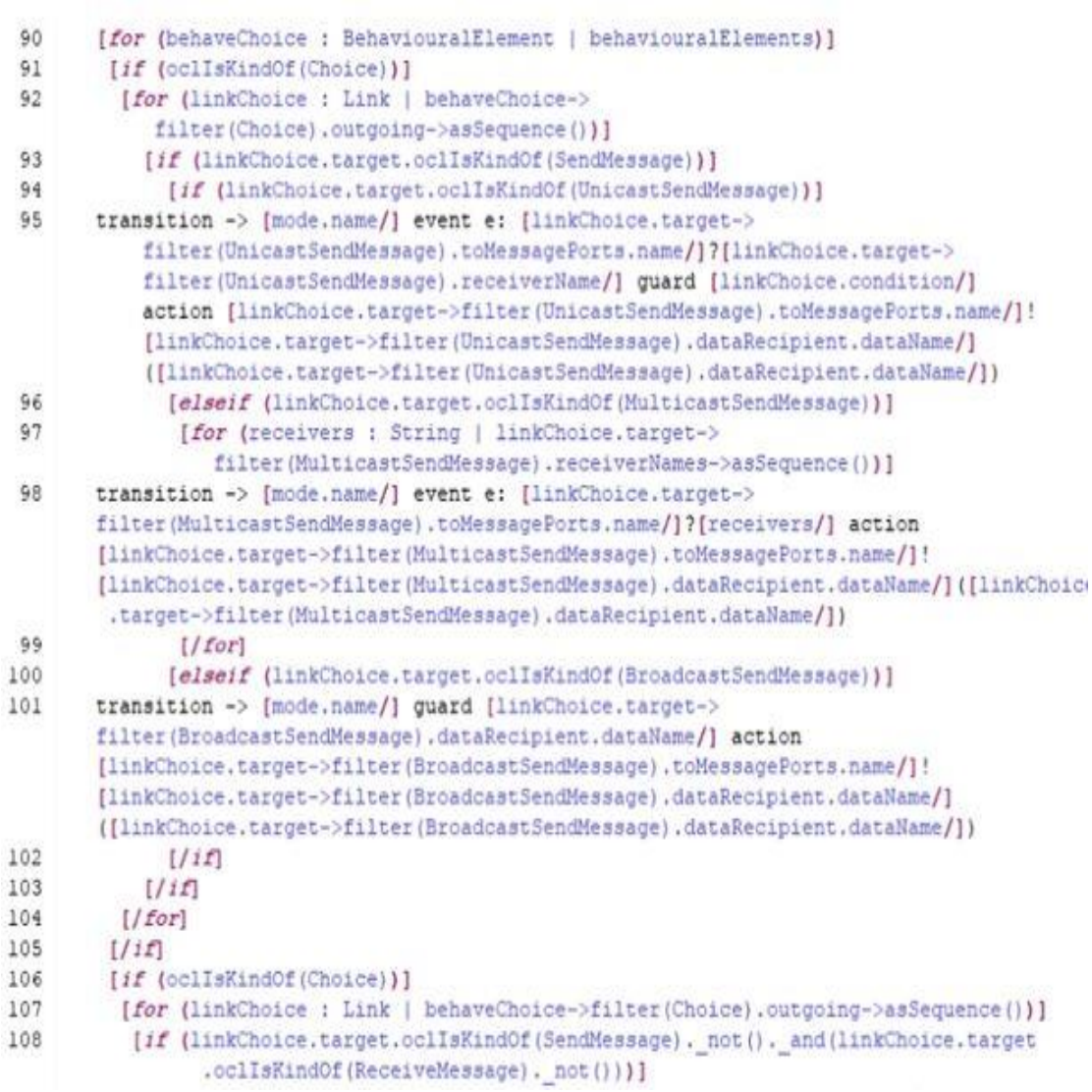
Représente la partie comportementale du composant. Composant Il peut avoir plusieurs modes dans lesquels le mode initial est sélectionné et la synchronisation

d'entrée et de sortie du reste des modes est sélectionnée. Contenir Un diagramme de cas qui comprend un ou plusieurs états illustrant le comportement réaliser la chose. Chaque diagramme de cas indique l'état initial dans CAPSSAML comme état initial et les autres modes comme états. Chaque état a une source d'entrée qui peut être identifiée par "On Entry" dans ThingML. La transition du mode à l'état est illustrée sur la **Figure. 3.10** Chaque mode a le sien Eléments comportementaux décrivant le concept de mise en œuvre de la situation. Les éléments comportementaux peuvent être principalement des événements, des circonstances et des actions, et peuvent également Les liens qui identifient la source et la cible sont des éléments comportementaux. Chaque lien Parmi les composantes comportementales, l'événement qui provoque le passage d'une composante comportementale à une autre doit être pris en compte. La procédure peut être une sélection imbriquée, Dans CAPS-SAML, il est affecté à l'événement Dans ThingML, l'état est défini sur Guard, le lien est défini pour aller et l'action dans CAPS-SAML peut être traduite en une action dans ThingML (si l'action envoie un message ou un choix), voir

Figure. 3.11, ou pour travailler dans ThingML. Un travail peut être responsable de nombreuses actions telles que la détection de données, le stockage de données, l'exploitation, etc. voir figure. 3.12 Voir la composante comportementale de la situation transformée en fonction. Voir "Pour connaître les règles de conversion SAML vers ThingML de voir image n°3 au chapitre 4" que Résume le mappage entre les éléments composant et objet.

```
76  // ***** States *****
77  [for (mode : Mode | modes)][comment display states /]
78  [if (mode.oclIsTypeOf(InitialMode))]statechart init [mode.name/] {[/if]
79   state [mode.name/] {
80    on entry do
81    [for (onEntry : BehaviouralElement | behaviouralElements)]
82     [if (onEntry.eClass().eSuperTypes->last().name.equalsIgnoreCase('Action'))]
83      [if (onEntry->filter(Action).incoming.source.oclIsKindOf(Choice)
          ->asSequence()->first().toString().equalsIgnoreCase('true')._not())]
84     [onEntry.name/]([if (onEntry.oclIsKindOf(StartTimer))]
                        [onEntry->filter(StartTimer).period/]
                       [else][onEntry.eCrossReferences()->
                         filter(PrimitiveDataDeclaration).value/][/if])
85      [/if]
86     [/if]
87    [/for]
88    end
```

Figure. 3.10 Modes transformation [41]



```
90  [for (behaveChoice : BehaviouralElement | behaviouralElements)]
91   [if (oclIsKindOf(Choice))]
92    [for (linkChoice : Link | behaveChoice->
        filter(Choice).outgoing->asSequence())]
93      [if (linkChoice.target.oclIsKindOf(SendMessage))]
94        [if (linkChoice.target.oclIsKindOf(UnicastSendMessage))]
95  transition -> [mode.name/] event e: [linkChoice.target->
      filter(UnicastSendMessage).toMessagePorts.name/]?[linkChoice.target->
      filter(UnicastSendMessage).receiverName/] guard [linkChoice.condition/]
      action [linkChoice.target->filter(UnicastSendMessage).toMessagePorts.name/]!
      [linkChoice.target->filter(UnicastSendMessage).dataRecipient.dataName/]
      ([linkChoice.target->filter(UnicastSendMessage).dataRecipient.dataName/])
96        [elseif (linkChoice.target.oclIsKindOf(MulticastSendMessage))]
97         [for (receivers : String | linkChoice.target->
              filter(MulticastSendMessage).receiverNames->asSequence())]
98  transition -> [mode.name/] event e: [linkChoice.target->
    filter(MulticastSendMessage).toMessagePorts.name/]?[receivers/] action
    [linkChoice.target->filter(MulticastSendMessage).toMessagePorts.name/]!
    [linkChoice.target->filter(MulticastSendMessage).dataRecipient.dataName/]([linkChoic
     .target->filter(MulticastSendMessage).dataRecipient.dataName/])
99         [/for]
100       [elseif (linkChoice.target.oclIsKindOf(BroadcastSendMessage))]
101 transition -> [mode.name/] guard [linkChoice.target->
    filter(BroadcastSendMessage).dataRecipient.dataName/] action
    [linkChoice.target->filter(BroadcastSendMessage).toMessagePorts.name/]!
    [linkChoice.target->filter(BroadcastSendMessage).dataRecipient.dataName/]
    ([linkChoice.target->filter(BroadcastSendMessage).dataRecipient.dataName/])
102       [/if]
103     [/if]
104   [/for]
105  [/if]
106  [if (oclIsKindOf(Choice))]
107   [for (linkChoice : Link | behaveChoice->filter(Choice).outgoing->asSequence())]
108    [if (linkChoice.target.oclIsKindOf(SendMessage)._not()._and(linkChoice.target
          .oclIsKindOf(ReceiveMessage)._not()))]
```

Figure 3.11 Choix et Traitement des messages [41]

- **Lancement de la phase du générateur de code**

À ce stade, nous exécutons le fichier Acceleo résultant des trois premières étapes. Nous avons donc construit un projet de lanceur Java qui importe la bibliothèque MTCLauncher. Le fichier écore du formulaire de base SAML est inclus sous dossier metamodel Le fichier CAPSml.mtl est inclus dans le dossier AcceleoTransformations. Pour démarrer la conversion de n'importe quel formulaire SAML, nous devons ouvrir Projet de lancement et intégration du fichier xmi du modèle SAML dans le dossier Modèles dans le projet de lanceur. Ensuite, nous exécutons le projet pour obtenir le fichier de sortie ThingML généré (CAPS. Thingml) sous le dossier public. Le fichier thingml. Contiendra ThingML a été transformé à partir du modèle SAML.

Le Framework CAPSml est capable de produire un fichier thingml complet. C'est la vérité En tant que lien entre les Framework CAPS-SAML et ThingML. Ainsi, il permet l'Internet des objets Les concepteurs qui utilisent SAML-CAPS pour modéliser et analyser leurs systèmes IoT, Exploiter la puissance de ThingML dans la

génération de code. La chose qui a été créée Le fichier peut être importé dans le Framework ThingML.

```
30  // ***** Behavioural Elements (Functions) *****
31  [for (modeFunc : Mode | modes)]
    [comment display Behavioural Elements in each mode as functions /]
32    [for (behaveFunc : BehaviouralElement | behaviouralElements)]
33      [if (oclIsKindOf(Link)._not()
          ._and(oclIsKindOf(UnicastSendMessage)._not())
          ._and(oclIsKindOf(MulticastSendMessage)._not())
          ._and(oclIsKindOf(BroadcastSendMessage)._not())
          ._and(oclIsKindOf(ReceiveMessage)._not())
          ._and(oclIsKindOf(Choice)._not())
          ._and(oclIsKindOf(TimerFired)._not()))]
34    function [behaveFunc.name/]([if (behaveFunc.oclIsKindOf(StartTimer))]
            val : Float[elseif (behaveFunc.eCrossReferences()->
            filter(PrimitiveDataDeclaration).type->notEmpty())]
            [if (behaveFunc.eCrossReferences()->
             filter(PrimitiveDataDeclaration).type->first()
              .toString().equalsIgnoreCase('real')) ]val : Float
            [else]val : [behaveFunc.eCrossReferences()->
              filter(PrimitiveDataDeclaration).type.toString()
               .toUpperFirst()/][/if][/if]) do
35    // Do something
36    end
37      [/if][comment end if testing /]
38    [/for][comment end Behavioural Elements /]
39  [/for][comment end Modes 1 /]
```

```
135 [/for][comment end Component /]
136 // ***** Configurations *****
137 configuration result {
138 [for (compConf : Component | Components)]
139  instance [compConf.name/]: [compConf.name/]
140 [/for]
141
142 [for (conn : Connection | element.SAElements->filter(Connection))]
143  connector
       [conn.target.eContainer(Component).name/].[conn.target.name/]
       => [conn.source.eContainer(Component).name/].[conn.source.name/]
144 [/for]
145 }
146 [/file]
147
148 [/template]
149
```

Figure 3.13 Transfer contact [41]

3.3 Méthodologie de modélisation et de génération de code

Il comprend trois étapes illustrées à la figure 3.14 :

1. Modélisation à l'aide du cadre CAPS : à cette étape, les concepteurs réalisent la conception Les systèmes IoT exploitent la puissance de CAPS dans la modélisation et Analyse des systèmes IoT [17] et graphiques Interface utilisateur optimisée par CAPS Modeling Framework.

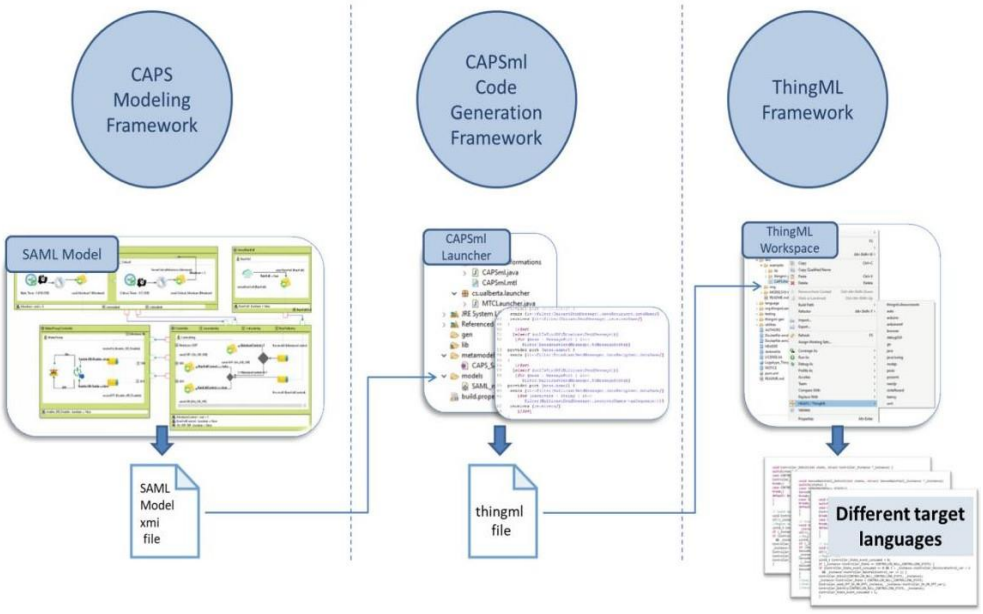


Figure. 3.14 Méthodologie de modélisation et de génération de code [41]

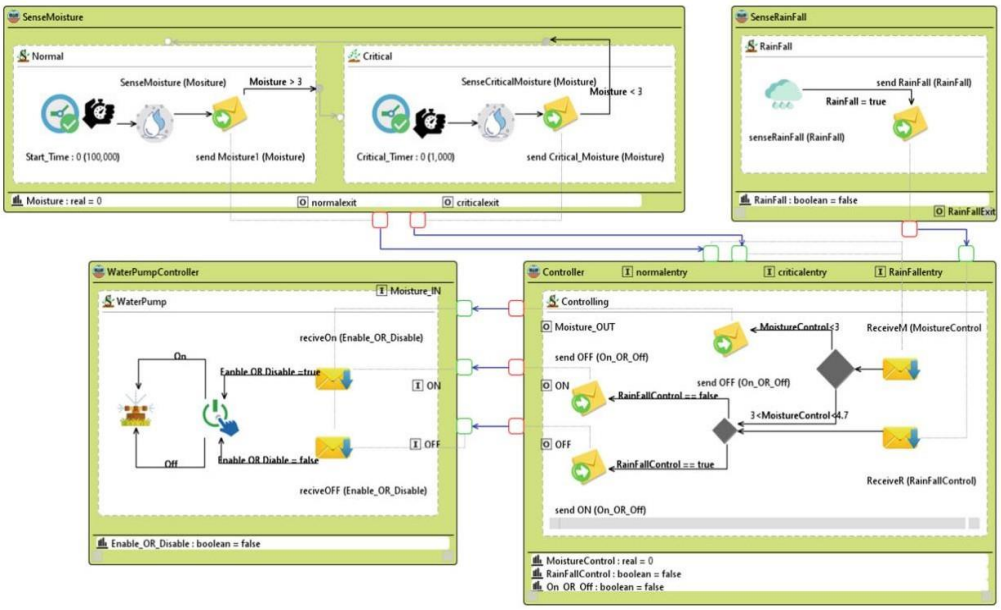
2. Exécution de l'infrastructure CAPSml : à ce stade, les concepteurs exécutent l'infrastructure CAPSml pour convertir le modèle CAPS-SAML en modèle ThingML. Le processus de conversion démarre automatiquement à partir des fichiers SAML xmi et ecore.
   Ensuite, le contenu du formulaire SAML est mappé au contenu de ThingML Modèle. Enfin, un langage ThingML complet est généré automatiquement au format.
3. Exécution du Framework ThingML : dans cette phase, les concepteurs utilisent le fichier thingML résultant de la phase deux de l'exécution du Framework ThingML. ThingML permet aux concepteurs de sélectionner parmi plusieurs compilateurs celui qui cible leur plate-forme souhaitable. Suivre cette méthodologie aide les développeurs à modéliser, analyser et produire Systèmes. Il atténue les problèmes des développeurs dans l'apprentissage du langage ThingML et donc les langages de programmation qui peuvent être générés à l'aide de ThingML Framework.

- **Une étude de cas d'irrigation intelligente**

L'agriculture est l'une des ressources les plus vitales pour l'économie et l'alimentation d'un pays Produire.

- **D’écrire un scénario à l'aide de CAPS**

Nous présentons un scénario simple décrit par la surveillance de l'humidité du sol. Un exemple de script SAML est illustré à la Fig. 15. Important à noter
Cette figure est une capture d'écran de la modélisation à l'aide de l'interface graphique
Propulsé par l'outil CAPS. Le modèle SAML se compose de quatre composants
1. Composant SenseMoisture : Il est responsable de la détection de la valeur de l'humidité du sol. Il comprend deux modes :
- Mode normal : Dans ce mode, le capteur d'humidité détecte la valeur d'humidité
De terre toutes les 100 secondes. Ensuite, il enregistre la valeur à l'humidité primaire.
- Mode critique : Dans ce mode, le capteur détecte la valeur d'humidité de chacun
En deuxième. Conserve la valeur de la variable primitive d'humidité.
2. Composant SenseRainfall : responsable de la détection des précipitations. Il elle
Il comprend un mode, RainFall.
- Mode précipitations : Dans ce mode, il y a un capteur d'interruption qui détecte sa présence
Est-ce qu'il pleut ou pas. La valeur extraite de ce capteur est conservée comme valeur primitive.



FIGUE. 16.3 Un exemple de composant de console SAML converti en console La chose dans ThingML [41].

3. Le Contrôleur Financier : Il est chargé de prendre les décisions d'équipe La pompe à eau est allumée ou éteinte. Comprend un mode, contrôle.

- Mode de contrôle : dans ce mode, les valeurs sont reçues de l'humidité et Les messages de précipitation sont stockés dans des variables primitives.
4. Composant WaterPumpController : Il est responsable du fonctionnement de la pompe Ou éteignez-le à la discrétion de l'observateur. Comprend un mode C'est la pompe à eau.
- Mode WaterPump : reçoit un message du composant de contrôle et Il le stocke dans une variable primitive. Cette valeur est envoyée au déclencheur. Si La valeur envoyée est correcte, l'actionneur met la pompe en marche. Si la valeur envoyée est Erreur, l'opérateur exécute une rampe.

- **Génération de code avec CAPSml**

Dans cette section, nous décrivons les résultats de l'exécution d'un modèle SAML sur CAPSml
Cadre d'action. Ensuite, nous affichons le code généré à l'aide de ThingML Code Génération Cadre d'action.
Avant d'exécuter le Framework CAPSml, nous devons définir le modèle décrit Puis, Pour exécuter le projet de lancement du Framework CAPSml, nous devons trouver le fichier xmi pour Le modèle créé à l'aide de CAPS-SAML [41].

```
180 // ***** Configurations *****
181 configuration result {
182     instance SenseMoisture: SenseMoisture
183     instance SenseRainFall: SenseRainFall
184     instance Controller: Controller
185     instance WaterPumpController: WaterPumpController
186
187     connector Controller.RainFallentry => SenseRainFall.RainFallExit
188     connector Controller.normalentry => SenseMoisture.criticalexit
189     connector Controller.criticalentry => SenseMoisture.normalexit
190     connector WaterPumpController.ON => Controller.ON
191     connector WaterPumpController.OFF => Controller.OFF
192     connector WaterPumpController.Moisture_IN => Controller.Moisture_OUT
193 }
```

Figure. 3.17 Partie de la configuration ThingML générée [41].

```
thing fragment Messages {
    message Moisture (value : Float)
    message RainFall (value : Boolean)
    message MoistureControl (value : Float)
    message RainFallControl (value : Boolean)
    message On_OR_Off (value : Boolean)
    message Enable_OR_Disable (value : Boolean)
}
```

```
// On Exit Actions:
void Controller_OnExit(int state, struct Controller_Instance *_instance) {
switch(state) {
case CONTROLLER_STATE:{
Controller_OnExit(_instance->Controller_State, _instance);
break;}
case CONTROLLER_NULL_CONTROLLING_STATE:{
break;}
default: break;
}
}

// Event Handlers for incoming messages:
void Controller_handle_OFF_On_OR_Off(struct Controller_Instance *_instance, bool value) {
if(!(_instance->active)) return;
//Region null
uint8_t Controller_State_event_consumed = 0;
if (_instance->Controller_State == CONTROLLER_NULL_CONTROLLING_STATE) {
if (Controller_State_event_consumed == 0 && 3 < _instance->Controller_MoistureControl_var < 4.7
   && _instance->Controller_RainFallControl_var == 1) {
Controller_OnExit(CONTROLLER_NULL_CONTROLLING_STATE, _instance);
_instance->Controller_State = CONTROLLER_NULL_CONTROLLING_STATE;
Controller_send_OFF_On_OR_Off(_instance, _instance->Controller_On_OR_Off_var);
Controller_OnEntry(CONTROLLER_NULL_CONTROLLING_STATE, _instance);
Controller_State_event_consumed = 1;
}
}
```

Figure. 3.18 Une partie du code général C++ pour Controller Thing/Component [41]

3.4 Une approche de transformation de modèles pour le code génération à partir du diagramme de la machine d'état

3.4.1 Aperçu de l'approche

Cet article présente une approche de génération de code par transformation de modèle prenant comme modèles source : le Domain Class Diagram (DCD) et le State machine Diagram (SMD) et au lieu de générer directement du texte brut pour la plate-forme choisie, un modèle structurel intermédiaire pour la plate-forme Java est généré. Un tel modèle intermédiaire permettra son extensibilité avec de nouvelles fonctionnalités. L'idée centrale de cet article est la génération de code par transformation de modèle à partir du diagramme d'état de transition pour les classes

complexes du système. Le code généré contient tous les détails de la classe (attributs) et les signatures complètes des méthodes.

L'approche permet la génération de spécifications d'opérations complètes déduites de la dynamique du système représentée dans le diagramme de machine d'état. De plus, nous générons des opérations supplémentaires nécessaires pour gérer le changement complet du cycle de vie des données (CRUD) en appliquant le profil EJB au modèle structurel généré de la plate-forme Java cible.

La méthode peut être décomposée en trois étapes principales (**Figure**.3.18) :

- *Identification des opérations* à partir des activités exécutées lors de la réception de différents événements.
- ***Spécification des signatures d'opération***. La signature de chaque opération est dérivée des actions et événements reçus par l'objet. Lors de la réception d'un événement, l'objet change d'état et exécute une activité. L'événement contient les paramètres nécessaires pour exécuter correctement cette activité. Par conséquent, nous pouvons déduire correctement la signature de l'opération complète.
- ***Génération de modèle structuré pour la plateforme java***. L'approche utilise un modèle structurel pour la plate-forme cible au lieu de générer directement le texte brut pour permettre son extension après génération avec des fonctionnalités supplémentaires.
- ***Application du profil EJB***. Pour étendre la plate-forme java avec des capacités EJB3, nous appliquons un profil EJB au modèle structurel généré permettant ainsi d'annoter les classes avec les informations nécessaires au mappage relationnel de l'entité pour garantir la génération de l'opération de base pour gérer les données (CRUD).
- ***Génération du code***. La dernière étape consiste à générer le code à partir du modèle structuré. Nous générons les classes completes avec la signature complète des opérations et les méthodes complètes avec l'implémentation complète du corps des opérations CRUD.

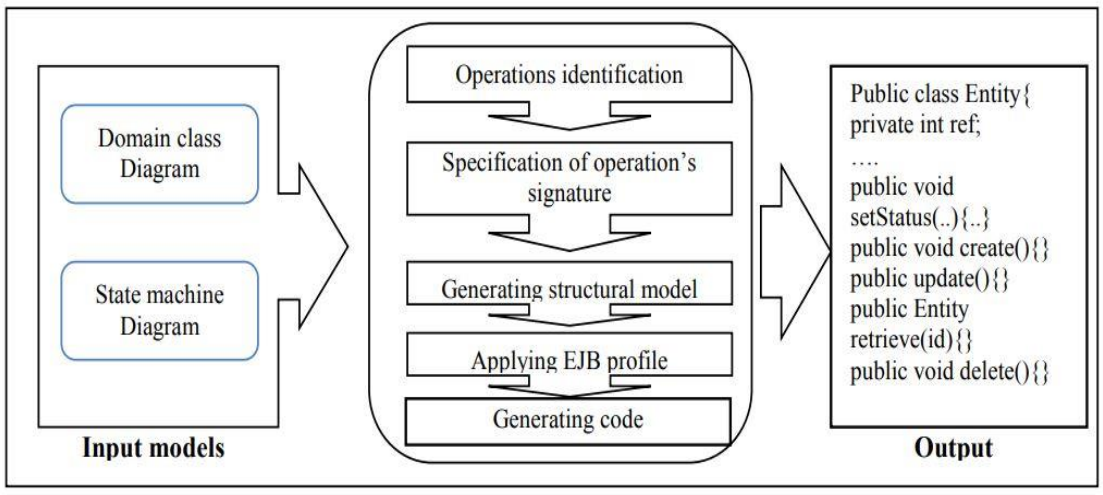


Figure 3.18 : Vue d'ensemble de la méthode.[42]

3.4.2 Métamodèles d’entrée

- **Métamodèle du diagramme de machine d’état**

Les diagrammes de machine d'état ont été utilisés dès le début dans la modélisation orientée objet. L'idée de base est de définir une machine qui a un certain nombre d'états (d'où le terme machine à états finis) qui définissent un ensemble de valeurs aux attributs de l'objet à un instant donné. La machine reçoit des événements du monde extérieur, et chaque événement peut faire passer la machine d'un état à un autre. Ainsi, cette machine représente le cycle de vie de la classe qui est essentiel pour représenter et façonner la dynamique du système et donner une définition formelle du comportement du système.

De plus, il est recommandé de tracer le diagramme d'état uniquement pour les classes complexes qui ont des états excessivement variables et sont souvent convoités par d'autres classes. Dans ce cas, il convient de créer un diagramme d'état de ces classes afin de limiter les transitions d'état pour une classe par rapport à ses interactions avec les autres.

Enfin analyser son comportement « instable » améliore notre compréhension du problème et implémente les méthodes de la classe. Ainsi, les diagrammes d'états permettent de compléter les diagrammes de classes de conception avec des méthodes qui correspondent aux différentes actions et activités du diagramme d'états.

L'exemple des machines à soda fait partie des exemples intéressants qui montrent l'importance d'utiliser le diagramme de transition d'état (Figure. 3.19).

La transition entre les états est activée par la réception d'un événement. Il est composé de paramètres, de conditions et exécute une action ou une activité. Un état a un temps limité et peut être simple ou complexe. Un état complexe est composé d'au moins deux autres états. Le passage d'un état à un autre est effectué par un événement extérieur. Une classe peut aussi avoir une transition interne sans changement d'état déclenchée par un événement interne. Les événements internes sont prédéfinis (entry, do, on event after et exit). L'événement exécute des actions ou des activités qui sont transformées en opérations ou en méthodes de classe.

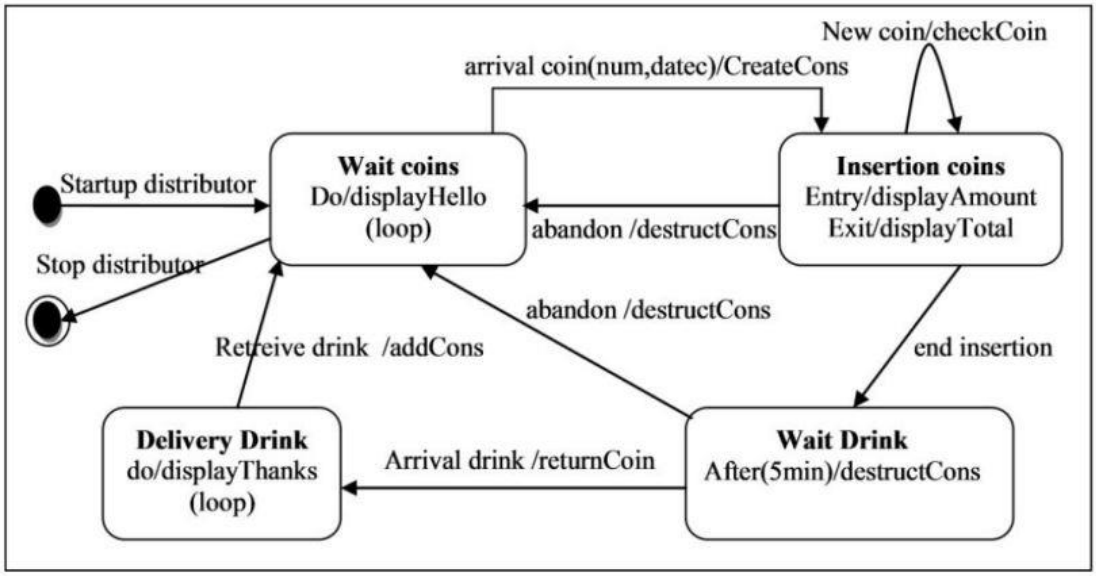


Figure 3.19 : diagramme de la machine à états de la machine à soda [42]

Dans certains cas, les systèmes informatiques ont plus de résultats d'opérations à partir d'événements que de résultats d'opérations à partir de l'interaction entre des objets. Dans ce cas, les diagrammes d'états de transition sont mieux placés pour trouver ces opérations que d'autres diagrammes d'interaction (diagramme de collaboration, diagramme de séquence de comportement interne...) tels que Cash machine, Drink's Distributor et Robot systèmes...

Par conséquent, dans la Fig. 3, nous présentons le métamodèle TSD basé sur les spécifications MOF de l'OMG [42]. Dans ce métamodèle, l'élément *StateMachine* commence par un état simple particulier le *initialState* et il inclut un ensemble de classes d'états abstraits qui sont de deux sortes : *SimpleState* et *ComposanteState*, ce dernier est composé d'au moins deux autres états imbriqués. La transition entre eux est représentée par la moyenne d'une méta-classe *Transition* qui représente la transition entre deux états *(oldState* et *newState).* Une transition peut être déclenchée lorsqu'une condition a été satisfaite par un événement qui peut contenir des paramètres, l'événement déclencheur peut être interne *InternalEvent*, représenté par les événements par défautÿ : *entrée, faire, sortie, après* ou *d'autres* événements ; ou *ExternalEvent externe.* Enfin, un événement effectue une action qui sera ensuite transformée en méthodes [42].

- **Métamodèle du diagramme de classes Domain**

Un diagramme de classes est l'un des principaux diagrammes d'une modélisation UML. Il permet de décortiquer le système en montrant ses composants (classes) permettant alors une véritable modélisation orientée objet. Il fournit une vue statique du système modélisé. Parfois, il est utilisé pour modéliser le vocabulaire du système. Cela implique une décision basée sur quels concepts ou entités font partie du système et quels concepts ou entités sont en dehors de ses limites. Le diagramme de classes est également utilisé pour créer des modèles de domaine, où tous les concepts de domaine d'application sont affichés dans le diagramme, y

compris les relations entre eux. Ils peuvent également être utilisés pour modéliser des collaborations entre un ensemble de classes, qui travaillent ensemble pour fournir un comportement collaboratif, ou même pour représenter un schéma de base de données, entre autres. Il est possible de construire un diagramme de classes à différents niveaux d'abstraction et avec différents degrés de détail. Par exemple, les modèles d'analyse, qui sont généralement utilisés dans la première phase du développement, n'ont pas de détails d'implémentation, tandis que les diagrammes de classes de conception auraient des détails d'implémentation.

Dans le métamodèle présenté dans la Figure 3.21 ci-dessous, nous avons présenté les différents éléments qui définissent les composants orientés objet des systèmes tels que les classes, leurs propriétés et leurs méthodes. Une propriété peut représenter un simple attribut ou une fin d'association liée à une relation entre classes. Cependant, une propriété peut identifier l'entité ou sera unique ou dérivable. Les différentes propriétés et méthodes ont un genre de visibilité présenté dans l'énumération *VisibilityKind [42]*.

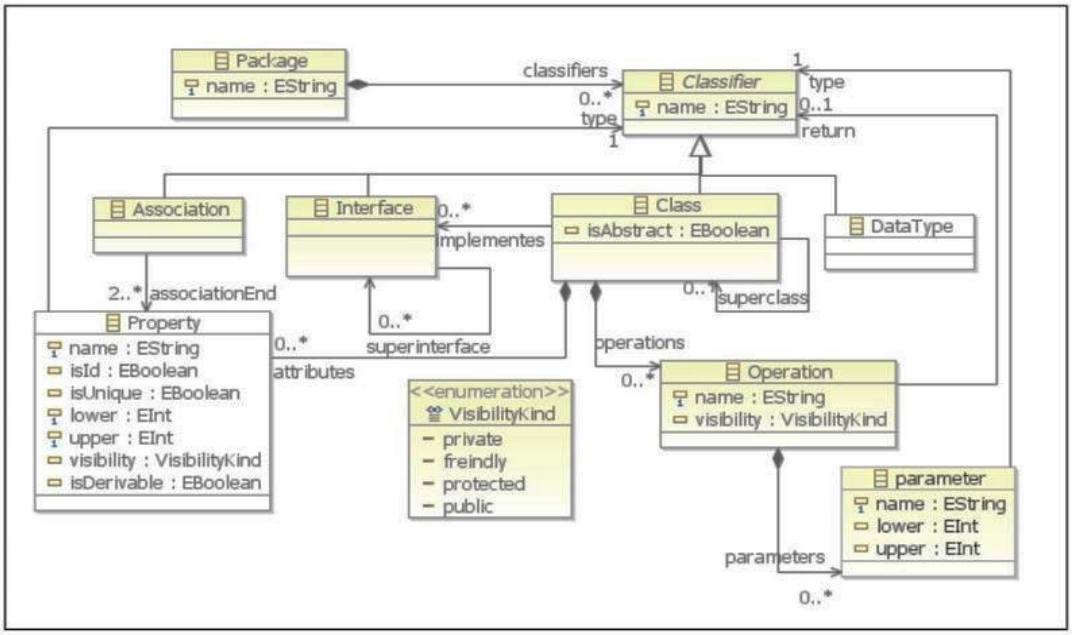


Figure 3.20 : Métamodèle de la machine d'état [42].

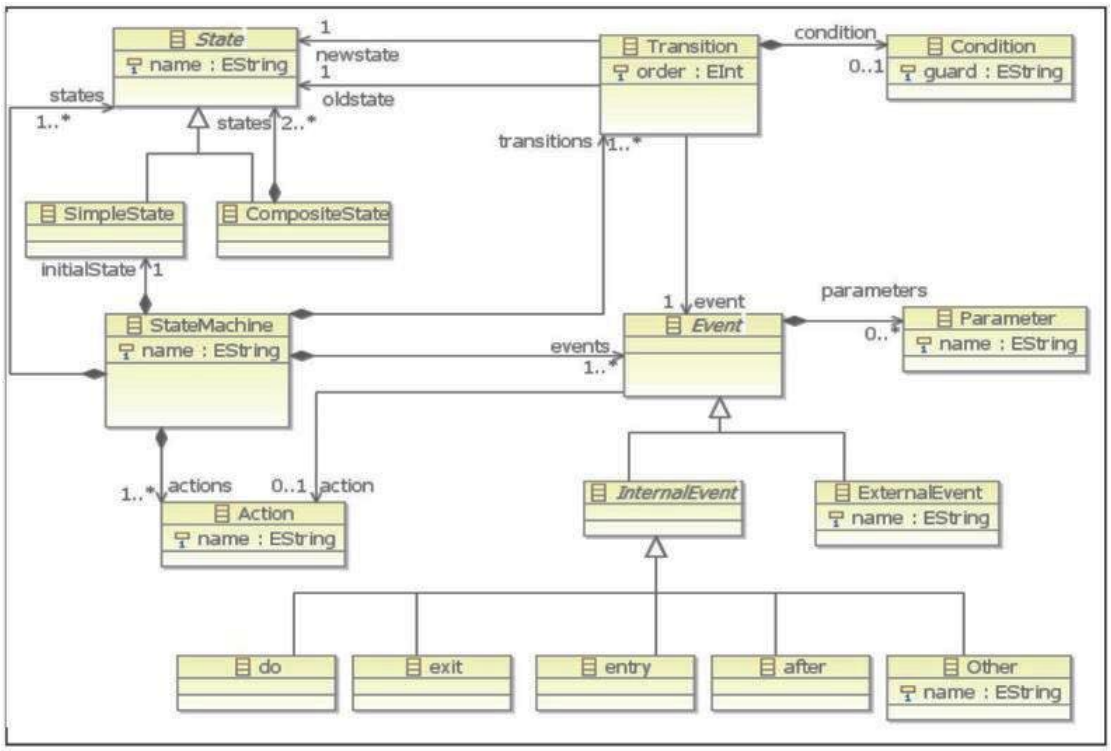


Figure 3.21 : Métamodèle de classe de domaine [42]

3.4.3 générations de code par transformation de modèle

- **Métamodèle cible : plate-forme Java**

Le choix de la plate-forme JAVA était arbitraire. En effet, nous avons dû choisir une plate-forme supportant le langage de programmation orienté objet. Nous avons utilisé le métamodèle proposé par le consortium OMG [42] dans lequel tous les éléments de la plate-forme Java sont représentés avec la même sémantique que celle spécifiée par Sun Des microsystèmes tels que JavaClass, JavaInterface organisés dans un JavaPackage où la classe Java comprend un ensemble de Champ et Méthode. Chaque champ du métamodèle de la plate-forme java a un JavaType identique à la méthode JavaParameter et sa valeur de retour. La Figure. 3.22 présente tous les éléments de ce métamodèle cible.

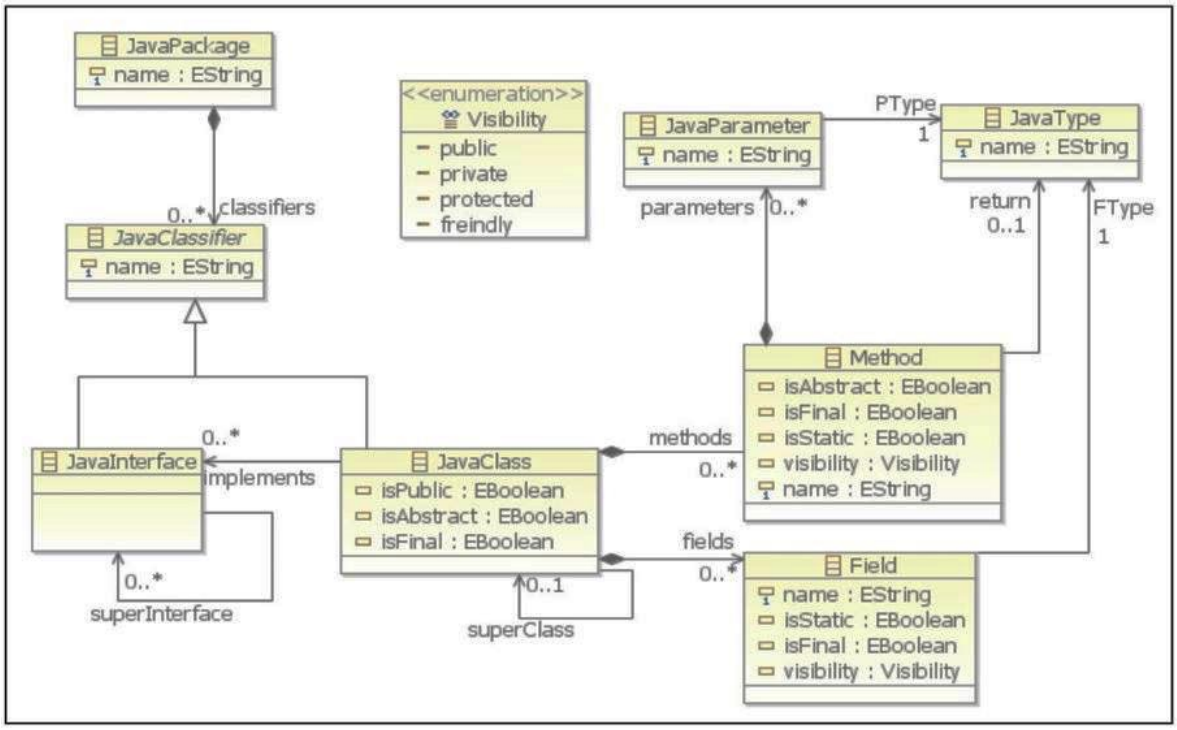


Figure 3.22 : Métamodèle de plateforme Java [42]

- **Génération du modèle java structurée :**

Dans cette section, nous présentons les principales règles de la transformation de modèle effectuée qui mettent en évidence la manière dont les différents éléments des métamodèles source sont mappés dans le métamodèle java et expliquons comment les signatures de méthodes attendues ont été générées [42].

Par conséquent, les premières règles divisent le package DCD en un package Java qui contient les classes Java obtenues à partir des classes de domaine dans le DCD. Les propriétés ou attributs de la classe simple sont transformés en champs dans la classe Java correspondante et l'association se termine en un champ avec le type d'un objet ou d'une collection d'objets en fonction de la multiplicité supérieure de ces propriétés. Ainsi, la multiplicité supérieure plusieurs *(égale à -1 dans le modèle DCD)* est mappée avec une collection d'objets ; sinon, il est mappé avec un champ simple du même type que l'objet à la fin de l'association. Tandis que les autres règles permettent d'affiner les classes Java par des méthodes issues d'actions TSD en mappant chaque action dans TSD liée à une transaction déclenchée par un événement dans des méthodes java, les paramètres de la méthode sont récupérés à partir des paramètres d'événement et leurs types à partir des propriétés correspondantes types déclarés dans le modèle DCD [42].

La figure 3.23 ci-dessous illustre les différentes règles de mappage des éléments de modèles d'entrée pour générer le modèle java cible.

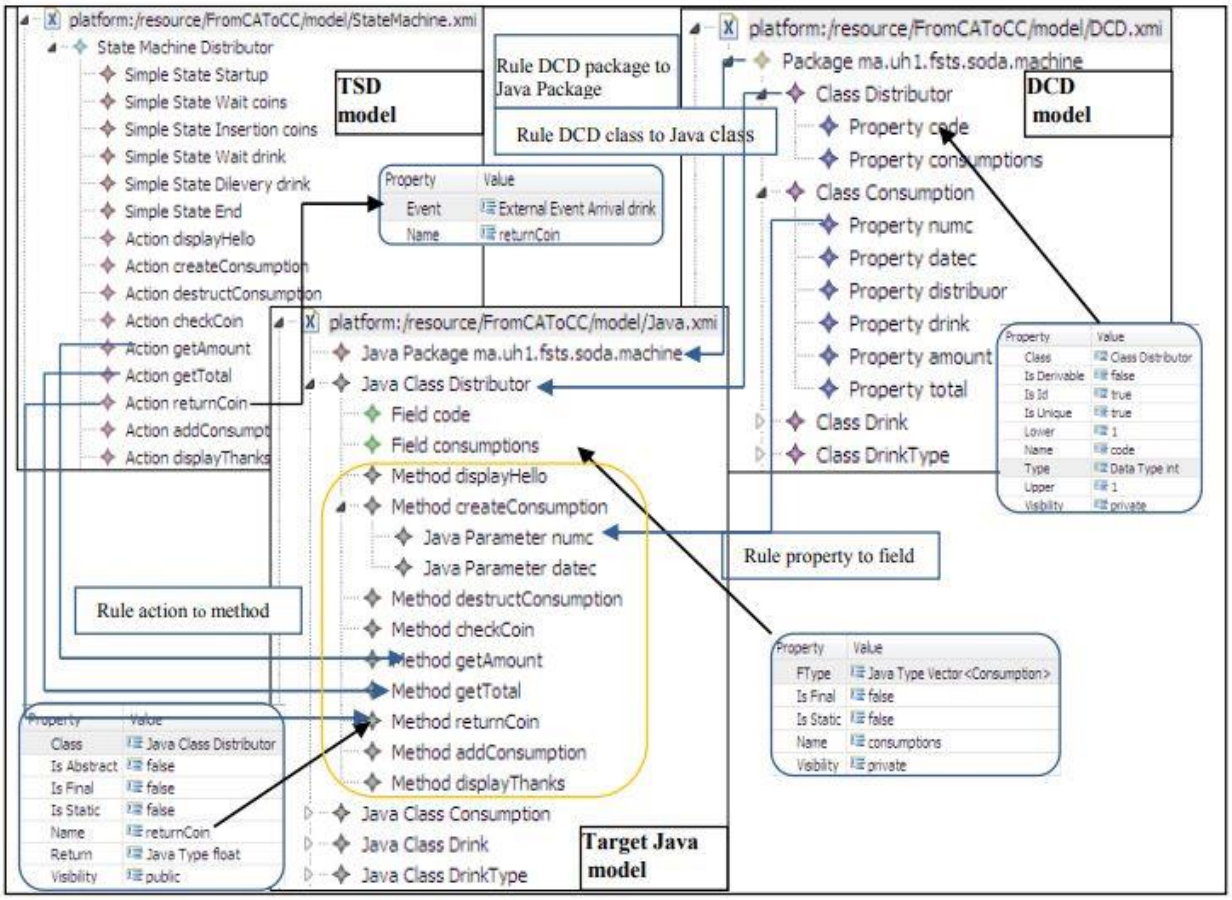


Figure 3.23 : modèle java générée et les modèles source de l'exemple en cours d'exécution [42]

3.4.4 Application du profil EJB

Profils EMF

Étant donné que le profil UML se situe au même niveau d'abstraction qu'UML lui-même, il ne peut être utilisé que pour étendre les modèles UML. Dans cet article, nous utilisons la plate-forme EMF qui utilise le méta métalangage ECORE pour créer différents métamodèles, nous ne pouvons donc pas utiliser les profils UML pour étendre notre DSML, d'où la nécessité d'utiliser le profil EMF [42].

De plus, le langage de transformation de modèle ATL utilisé dans cette proposition ne prend pas en charge le modèle UML, seuls les modèles basés sur le métamodèle Ecore peuvent être utilisés comme modèles source de la transformation.

L'objectif principal de l'application de profils dans cet article est de montrer comment le modèle structurel généré de la transformation effectuée peut être étendu avec de nouvelles fonctionnalités avant de générer le code d'implémentation. Comme exemple, nous avons appliqué le profil EJB3 qui est présenté dans la section suivante. L'utilisation de profils présente de nombreux

avantages comme la possibilité d'annoter le modèle le plus légèrement possible ; par conséquent, aucune adaptation des métamodèles existants ne devrait être nécessaire. De plus, il évite de polluer les métamodèles existants avec des préoccupations non directement liées au domaine de modélisation séparant les annotations du modèle de base pour permettre d'importer uniquement les annotations qui présentent un intérêt actuel pour un modélisateur particulier dans une situation particulière [42].

Pour intégrer le mécanisme de profil dans EMF, un langage de spécification de profils est nécessaire comme premier ingrédient. Ceci est facilement réalisé en créant un métamodèle basé sur Ecore appelé Profile Métamodèle qui sera instancié pour créer un profil spécifique, contenant des stéréotypes et des valeurs étiquetées.

Une fois qu'un profil spécifique est disponible, les utilisateurs devraient maintenant pouvoir appliquer ce profil à des modèles arbitraires en créant des applications stéréotypées contenant des valeurs concrètes pour les valeurs étiquetées définies dans les stéréotypes [42].

- **Profile EJB3**

Nous donnons ici un exemple d'extension du modèle java générée de la transformation effectuée en appliquant le profil EJB 3 établi à partir des profils EMF. Ainsi, nous pouvons générer un code source supplémentaire comme l'opération CRUD permettant la persistance par le biais des entités EJB et des sessions EJB.

La figure 3.24 ci-dessous présente les différents éléments de ces profils EJB représentés par un ensemble de stéréotypes et de valeurs étiquetées. Nous avons également présenté les méta classes du métamodèle Java qui ont été étendues. Par exemple, le stéréotype [42]

« Field » est utilisé pour étendre la méta classe Java Field (attribut de classe Java) avec les informations nécessaires pour le mappage complet de la colonne associée, telles que les contraintes nullables ou modifiables, que le champ soit un identifiant ou non et que la colonne nom qui est nécessaire si l'attribut et la colonne ont des noms différents.

Il définit à la fois *stateful* qui est un Bean session qui représente une session conversationnelle avec un client particulier, ces objets de session maintiennent automatiquement leur l'Etat conversationnel sur plusieurs méthodes invoquées par le client et les beans session sans état qui représentent un bean EJB sans état pour un client qui n'invoquera qu'une seule méthode. Le client d'un bean session peut être un client local, un client distant ou un client de service Web selon l'interface fournie par le bean et utilisée par le client. Un objet entité représente un objet persistant à grain fin. Le client d'un bean entité peut être un client local ou le client peut être un client distant. Une méthode EJB est déclarée par une déclaration de méthode Java dans une interface EJB Home ou Remote. La méthode EJB est une méthode EJB Home, si elle est déclarée dans une interface EJB Home, ou une méthode EJB Remote, si elle est déclarée dans une interface EJB Remote. La

déclaration d'une interface distante EJB étend la déclaration d'une interface Java avec des éléments de descripteur de déploiement EJB pour un EJB Enterprise Bean. Le nom de l'EJB Remote Interface et de l'EJB Home Interface associée sont spécifiés par les éléments remote et home dans l'élément entity ou session pour l'EJB Enterprise Bean [42].

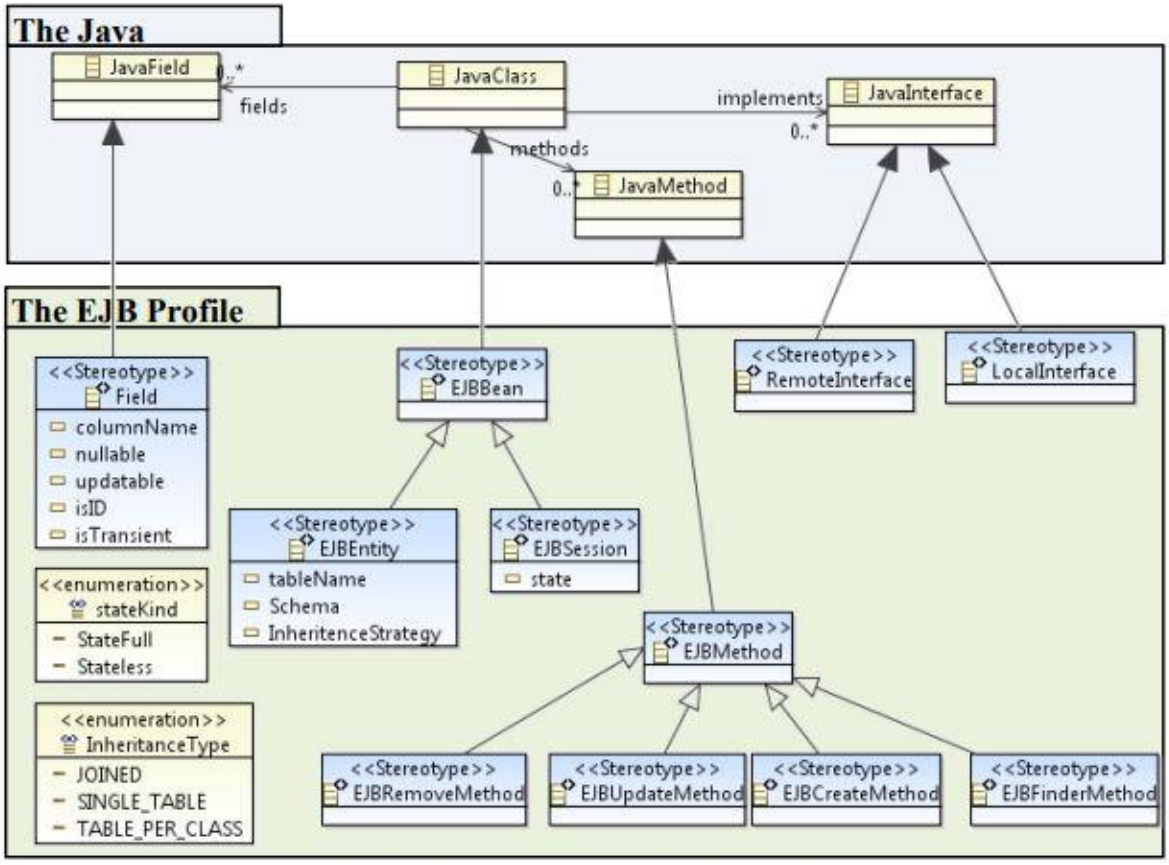


Figure 3.24 : Le profil EJB avec les différentes méta classes étendues [42]

3.4.5 Génération de code

Pour générer du code à partir du modèle java structurée, un reste de règles est défini pour décrire comment générer un code java qui crée les classes avec leurs champs et méthodes organisés dans un package java par rapport à ceux existants dans le modèle cible comme illustré à la Fig. .8. La figure 6 montre quelques règles de transformation pour mapper les différents éléments des métamodèles d'entrée à celui cible. Pour générer ce code, nous avons utilisé la requête ATL qui est une expression OCL qui permet de spécifier des requêtes sures. Ainsi, la requête *"generateCode ()"* utilisée dans la proposition nous a permis de naviguer à travers les éléments du modèle java et de générer automatiquement le package qui contient les classes java, leurs champs et les signatures des méthodes. Afin d'effectuer la génération de code, d'autres assistants ATL sont employés pour accomplir la transformation, tels que *"getParameters ()"* et *"getSignature ()"*. Enfin, nous nous concentrons sur la spécification des signatures des méthodes qui dépend des actions incluses dans la TSD. Certains événements liés à ces actions

peuvent nécessiter l'ajout de nouveaux paramètres pour déclencher l'état de transition. Par conséquent, chaque variable qui apparaît en tant que paramètre dans l'événement doit également apparaître en tant que paramètre dans la méthode, comme les paramètres *numc* et *datec* de la méthode *createConsumption () [42]*.

Notez que les types *Int* et *Date* de ces paramètres sont déduits des attributs DCD correspondants. Le code généré prend également en charge l'importation des bibliothèques telles que *java. Util* nécessaires pour utiliser la collection et le type de date dans le code généré en vérifiant s'il existe des champs dans le modèle java. Concernant la visibilité de la méthode, nous avons choisi de rendre la visibilité publique pour obtenir et afficher les méthodes et privée dans le cas contraire. Certains types de méthodes de retour sont également déduits du DCD car ils sont relatifs à des objets ou des propriétés existant dans le DCD [42].

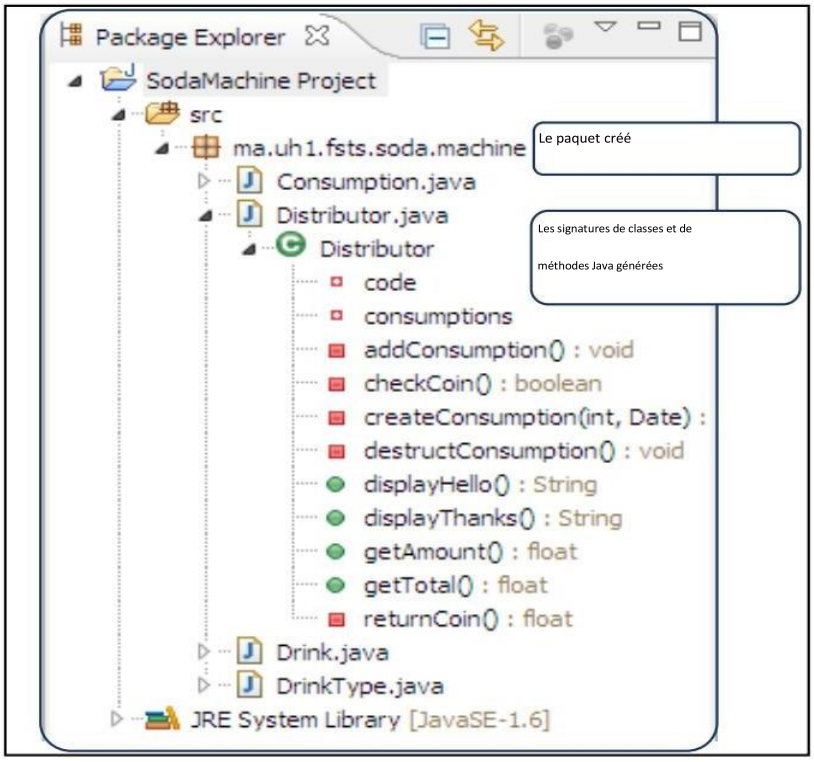


Figure 3.25: Le projet Java généré [42]

La deuxième transformation effectuée dans cet article est une transformation modèle-code qui permettra de transformer le modèle structurel généré du modèle java qui a été enrichi en appliquant le profil EJB, le code source selon la plate-forme JAVA. Ainsi, pour chaque classe Java un fichier java sera généré contenant le code source de la classe incluant leurs méthodes (signature et corps) et l'annotation EJB pour la classe JAVA stéréotypée avec « *EJBEntity* ». De plus, pour chaque classe persistante, un Stateless *EJBSessionbean* et son interface Remote ou Home correspondante seront générés en implémentant le code source détaillé de l'opération CRUD pour la manipulation des données ainsi que le

fichier de configuration pour la persistance "persistence.xml". Ces opérations sont : Save() qui effectue une sauvegarde initiale d'une entité *EntityClass* précédemment non sauvegardée; delete() pour supprimer une entité *EntityClass* persistante; update() qui permet de conserver une entité *EntityClass* précédemment enregistrée et de la renvoyer ou d'en renvoyer une copie à l'expéditeur, une copie du paramètre d'entité *EntityClass* est renvoyée lorsque le mécanisme de persistance JPA n'a pas précédemment suivi l'entité mise à jour; le findById() permettant de récupérer une *EntityClass* par identifiant et le *findAll* pour récupérer toutes les instances de cette EntityClass. Par exemple, pour une classe d'entité persistante *EntityClass*, le bean et l'interface sans état suivants seront générées pour prendre en charge toutes les opérations nécessaires à la gestion de l'ensemble du cycle de vie des données [42] :

```
import javax.ejb.Remote;
@Remote
public interface EntityClassRemote {

        public void save(EntityClass entity) ;
        public void delete(EntityClass entity) ;
        public EntityClass update(EntityClass entity) ;
        public EntityClass findById(idType idName) ;
        public List< EntityClass > findAll() ;

}
```

Figure 3.26: interface EntityClassRemote [42]

```
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;
@Stateless
public class EntityClass implements EntityClassLocal, EntityClassRemote {
        @PersistenceContext
        private EntityManager entityManager;

        public void save(EntityClass entity) {
                try {
                        entityManager.persist(entity);
                } catch (RuntimeException re) {
                        throw re;
                }
        }
        public void delete(EntityClass entity) {
                try {
                        entity = entityManager.getReference(EntityClass.class,
                                        entity.getId());
                        entityManager.remove(entity);
                } catch (RuntimeException re) {
                        throw re;
                }
        }
        public EntityClass update(EntityClass entity) {
                try {
                        EntityClass result = entityManager.merge(entity);
                        return result;
                } catch (RuntimeException re) {
                        throw re;
                }
        }
        public EntityClass findById(IDType id) {
                try {
                EntityClass instance = entityManager.find(EntityClass.class, id);
                        return instance;
                } catch (RuntimeException re) {
                        throw re;
                }
        }
        public List<EntityClass> findAll() {
                try {
                final String queryString = "select model from EntityClass model";
                Query query = entityManager.createQuery(queryString);
                return query.getResultList();
                } catch (RuntimeException re) {
                        throw re;
                }
        }
}
```

Figure 3.26: EntityClass [42]

3.5 Conclusion

Ce chapitre donne un aperçu des travaux antérieurs des ingénieurs logiciels dans le domaine de la récupération de code à partir du schéma des systèmes IoT, et nous expliquons également tout ce qui concerne modélisation et de génération de code pour l'IOT et Une approche de transformation de modèles pour le code génération à partir du diagramme de la machine d'état.

Dans le chapitre suivant, nous expliquerons la méthode ou la solution que nous ferons pour convertir le code de l'Internet des objets en un schéma que nous pouvons comprendre.

Chapitre 4 : Récupération d'une Architecture SAML à partir d'un code ThingML

4.1 Introduction

Dans ce chapitre, nous présentons la solution que nous allons utiliser et la méthode pour convertir le code ThingML en modèle CAPS, puis effectuons le processus inverse. Nous expliquerons également tous les titres suivants : fonctionnement général et Architecture de ThingML et Meta Model SAML et étapes de notre processus.

4.2 Fonctionnement général

Nous donnons d'abord un aperçu de l'approche de développement proposée basée sur l'extraction de schémas de systèmes IoT, puis nous introduisons le modèle IoT proposé pour Saml-Caps via ThingML, un langage de programmation pour les systèmes IOT.

Dans ce travail, comme le montre la figure 4.1, nous allons convertir en deux étapes de ThingML à saml, ce qui se traduira par un schéma CAPS, puis revenir de SAML, qui incarne le schéma CAPS pour nous, à ThingML.

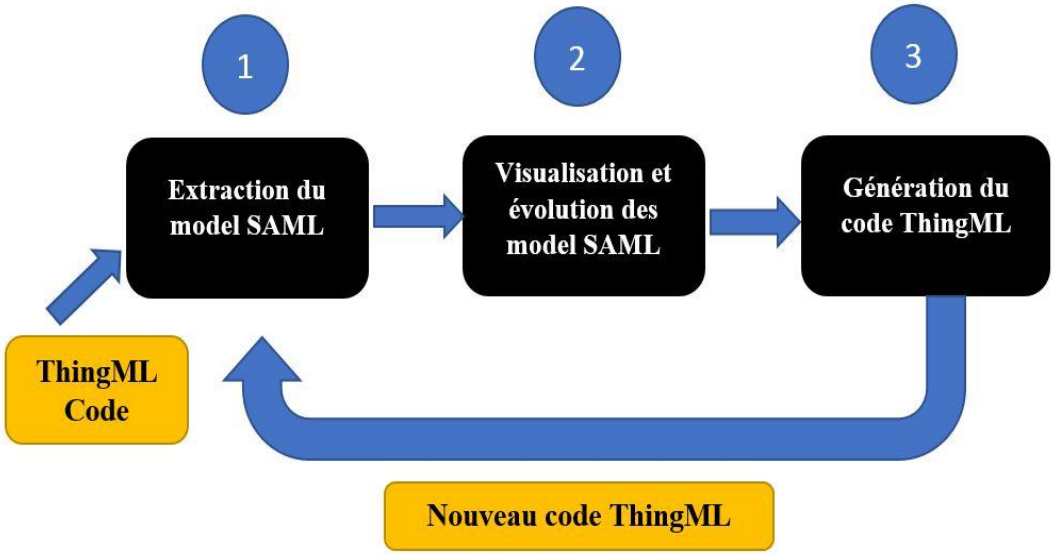


Figure 4.1 Processus proposé

CAPS et ThingML peuvent être considérés comme des langages de modélisation spécifiques à un domaine (DSML) lorsqu'ils sont comparés à UML, mais ils peuvent être utilisés pour une large gamme d'applications car elles ne sont pas limitées à un domaine d'activité spécifique. Puisque ThingML vise à modéliser des composants logiciels et à générer automatiquement des modules de code prêts à être

déployés et implémentés, si nous pouvions trouver un moyen de convertir ThingML en CAPS-SAML, nous pourrons alors avoir plusieurs modules de codes pour le même modèle conçu à l'aide de SAML. Cela nous donnerait une plus grande variété, une évolutivité et une meilleure efficacité.

Notre processus passe par les étapes suivantes :

1) Dans la première étape, nous allons convertir ThingML en SAML. Après avoir utilisé le fichier d'entrée ***InputThingML.xmi*** et extrait le fichier de sortie ***OutputSAML.xmi***, ce qui nous permettra de créer un diagramme Caps qui expliquera le fonctionnement du système Internet des objets de manière simple pour les développeurs à travers le diagramme,
2) Dans la deuxième étape, nous visualisons et évoluons des modèle SAML ou utilisons le Framework Caps.
3) Dans la troisième étape, nous passerons de SAML à ThingML. Après avoir utilisé le fichier d'entrée ***InputSAML.xmi*** et extrait le fichier de sortie ***OutputThingML.xmi***, ce qui nous permettra d'extraire le code du schéma CAPS en syntaxe ThingML qui expliquera le fonctionnement du système IoT par code.

4.2.1 ThingML Architecture

ThingML est considéré comme un Framework de génération de langage et de code pour les systèmes hétérogènes. Cette approche est venue soutenir le processus de génération de codes à partir de modèles. ThingML comprend un langage de modélisation un outil conçu pour prendre en charge la génération de code et une génération de code multiplateforme personnalisable cadre [27]. ThingML a été utilisé pour développer des systèmes allant de la recherche

Des études de cas au développement de produits dans des projets industriels. ThingML cible les systèmes IOT distribués et offre un grand avantage lors de l'application dans des environnements hétérogènes. Plates-formes.

ThingML a été développé sur la base du Model Driven Engineering (MDE) des principes. Alors que les modèles dans MDE peuvent être vraiment utiles dans les exigences ou phases de repos, ThingML vise à apporter MDE aux phases de conception tardive et de mise en œuvre du cycle de vie du logiciel ainsi qu'à soutenir la maintenance et les tâches d'évolution [27].

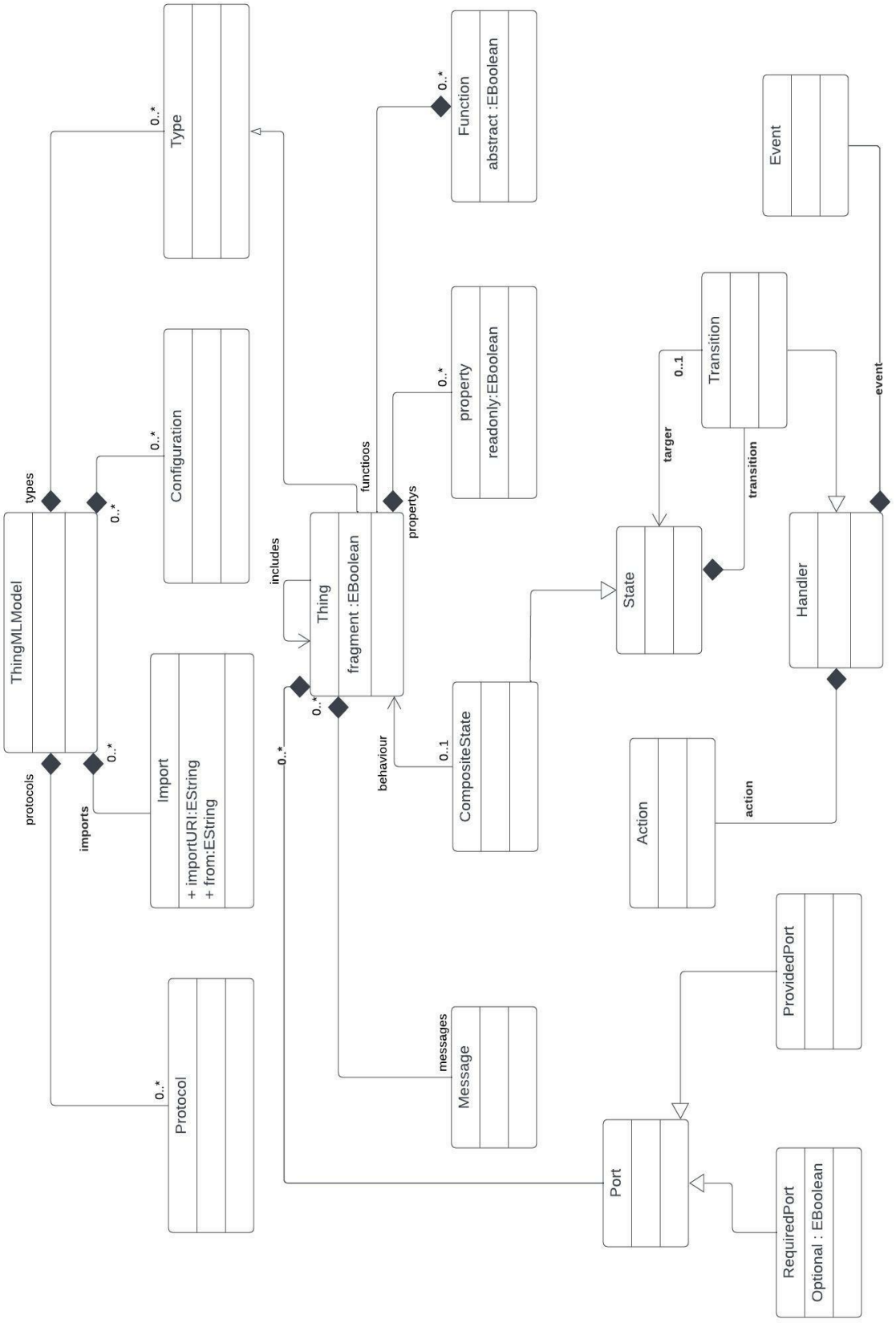


Figure 4.2 Extrait du métamodèle du langage ThingML

- **ThingML DSL**

Le langage ThingML est composé de deux structures clés : les choses qui représentent des composants logiciels et des Configurations qui décrivent leurs interconnexions. Une Chose peut être définie comme une unité d'implémentation, un composant ou un processus [27].

Une chose peut attribuer des propriétés, des fonctions, des messages et des ports. Ça peut aussi contiennent un ensemble de machines d'état. Les variables de propriétés sont définies localement à l'intérieur de la chose et accessible de l'intérieur. Les fonctions peuvent également être traitées en tant que fonctions locales à l'intérieur de la chose et ne peuvent pas être consultées ou vues de l'extérieur. Les ports sont la seule interface publique du langage ThingML. Ils ont l'habitude d'envoyer ou recevoir des messages qui sont définis dans une chose mais qui ne peuvent être utilisés pour envoyer ou recevoir via les ports de messages.

Le comportement interne d'une chose peut être obtenu en combinant :

• Programmation fondamentale pour clarifier les procédures soit en utilisant les Actions de la plateforme ThingML et langage d'expression ou en utilisant les fonctionnalités du langage cible. Les bibliothèques des langues ciblées peuvent être emballé et utilisé. Un mélange des deux options peut être mis en œuvre.

• Event-Condition-Action (ECA) qui définit simple si afin d'appliquer règles en réponse à la survenance d'événements.

• Machine d'état composite, pour réagir et coordonner les événements de manière dynamique. Ceci est conforme aux diagrammes UML.

- **Framework de génération de code ThingML**

Le Framework de génération de code ThingML est composé d'une famille de compilateurs qui sont capables de transformer un modèle ThingML en un code entièrement opérationnel dans différentes langues [27]. À ce jour, ThingML dispose de 4 langues entièrement prises en charge qui sont Java, C, JavaScript et d'autres langages sont en cours de développement.

4.2.2 SAML Meta Model

CAPS a été conçu et mis en œuvre sur la base de la distribution de la modélisation informations en 3 vues architecturales : le logiciel structurel et comportemental (SAML), la vue matérielle (HWML) et la vue de l'espace physique (SPML).

CAPS a également un moyen de combiner ces trois vues de modélisation ensemble, afin de dire que les éléments logiciels définis dans le modèle logiciel ont ces exigences matérielles spécifiques qui sont définis dans le modèle matériel. Ceci est fait par le MAPML qui fait le mappage entre le logiciel et les modèles matériels. De la même manière, le DEPML relie le modèle matériel au modèle vu de l'espace, pour spécifier l'emplacement de chaque élément matériel. [25]

Dans ce travail, nous nous concentrerons sur le langage de modélisation logicielle (SAML) qui sera converti en ThingML. CAPS permet aux architectes logiciels de définir l'architecture logicielle du système IOT en utilisant la modélisation SAML langue.

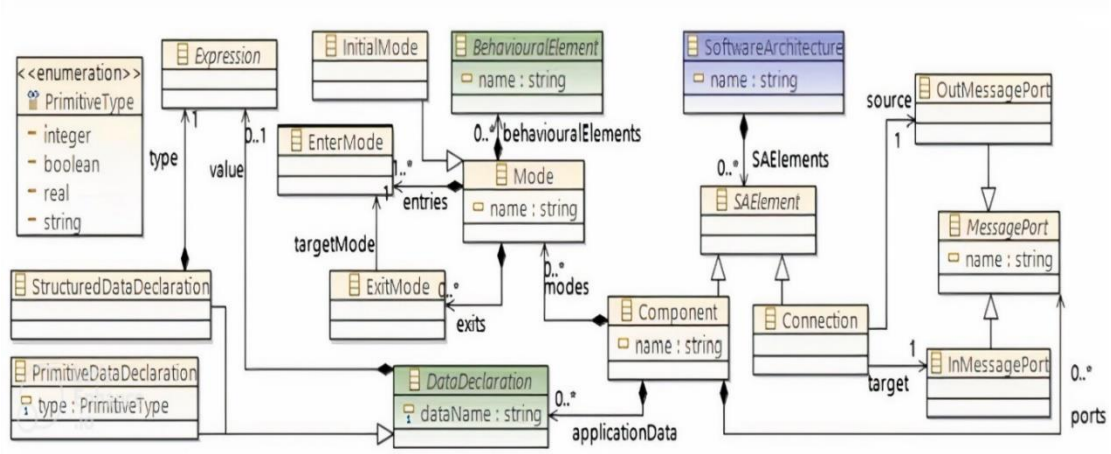


Figure 4.3 Concepts structurel du Métamodèle SAML [41]

SAML est essentiellement composé de ***components*** et de ***connections*** entre eux. Ces ***components*** échangeront des informations en utilisant ***message-ports***. ***Component*** peut être défini comme une unité de calcul avec un état interne et bien défini interfaces [25].

Le comportement de chaque ***Component*** est déterminé par un ensemble d'événements, gestes et conditions. Les variables locales peuvent également être définies à l'intérieur du ***Component*** scope qui précise les données applicatives qui seront induites par les événements, gestes et conditions.

Component peut aussi contenir plusieurs ***Modes*** qui préciseront l'état de ce ***Component***, par exemple un ***Mode*** peut être un ***Mode*** d'économie d'énergie ou un mode veille mode.

Component ne peut avoir qu'un seul ***Mode*** actif à la fois pour déterminer le statut de celui-ci. Un ***Mode*** a une action qui est utilisée pour le message passant d'un mode à un autre, cela s'appelle ***ExitMode*** qui passera des ***Messages*** dans un événement appelé ***EnterMode***. Chaque ***Mode*** peut contenir plusieurs éléments de comportement qui ensemble représenteront le flux de contrôle du ***Component***

4.3 Étapes de notre processus

Ce processus se décompose en deux étapes importantes : pour la première étape, la conversion de ThingML vers SAML, qui aboutira à un schéma Caps, pour la deuxième étape, la conversion de SAML vers ThingML.

Récupération d'architecture de logiciel - ThingML vers SAML

- **Étape 1 : donner un fichier ThingML**

À cette étape, nous allons donner au fichier ThingML un type xmi dans notre projet afin que nous puissions le lire et le convertir en suivant certaines règles via les jetons générés automatiquement à partir du modèle ThingML.

- **Étape 2 : conversion du fichier ThingML**

Cette étape est importante dans notre projet, car nous allons utiliser des règles de transformation pour passer de ThingML à SAML, ce qui se traduira par un schéma Caps. Toutes ces transformations sont plus précisément la transformation de ThingML Model vers Modèle SAML

Comme la sortie sera générée automatiquement des symboles à partir de modèles, et parmi les règles de conversion dont nous avons parlé sont :

- ✓ Convertit les objets de la classe *ThingMLModel* dans le modèle ThingML en éléments de la classe *SoftwareArchitecture* dans le modèle SAML-CAPS.

- ✓ Convertir les éléments de classe *Thing* dans le modèle ThingML en éléments de classe *Composant* dans le modèle SAML-CAPS.

- ✓ Convertissez la classe *Property* dans le modèle ThingML en classe *PrimitiveDataDeclaration* dans le modèle SAML-CAPS.

- ✓ Convertissez la classe *Message* dans le modèle ThingML en classe *PrimitiveDataDeclaration used message exchanging* dans le modèle SAML-CAPS.

- ✓ Convertissez la classe *RequiredPort* dans le modèle ThingML en classe *InMessagePort* dans le modèle SAML-CAPS.

- ✓ Convertissez la classe *ProvidedPort* dans le modèle ThingML en classe *OutMessagePort dans* le modèle SAML-CAPS.

- ✓ Convertissez la classe *State* dans le modèle ThingML en classe *Mode* dans le modèle SAML-CAPS.

- ✓ Convertissez la classe *Event* dans le modèle ThingML en classe *Event* dans le modèle SAML-CAPS.

- ✓ Convertissez la classe *Guard* dans le modèle ThingML en classe *Condition* dans le modèle SAML-CAPS.

- ✓ Convertissez la classe *Action* dans le modèle ThingML en classe *Action* dans le modèle SAML-CAPS.

- ✓ Convertissez la classe *Function* dans le modèle ThingML en classe *Action other then send message* dans le modèle SAML-CAPS.

- ✓ Convertissez la classe *Transition* dans le modèle ThingML en classe *Link* dans le modèle SAML-CAPS.

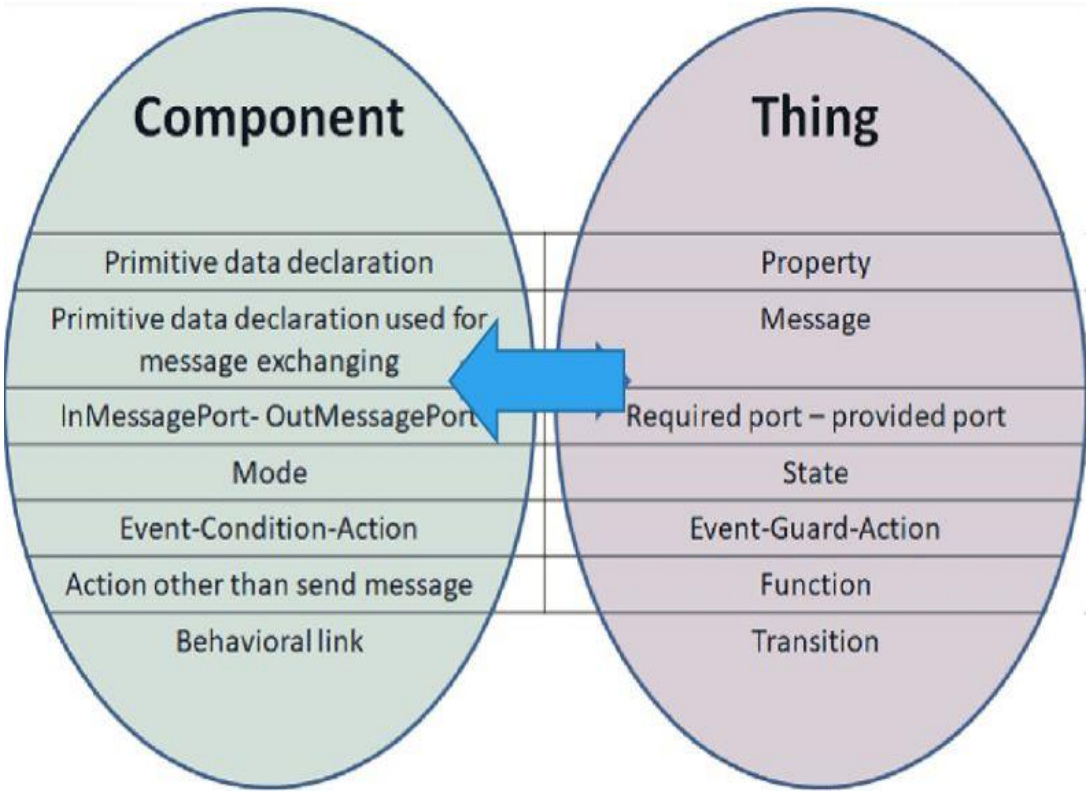


Figure 4.4 Conversion du ThingML vers SAML

- **Étape 3 : d'extraction du fichier SAML**

Dans cette dernière étape, nous aurons un fichier SAML xmi qui nous permettra de dessiner des diagrammes CAPS.

Récupération code de logiciel - SAML vers ThingML

- **Étape 1 : Donner un fichier SAML**

À cette étape, nous allons donner au fichier SAML un type xmi dans notre projet afin que nous puissions le lire et le convertir en suivant certaines règles via les jetons générés automatiquement à partir du modèle SAML-CAPS Par exemple *SoftwareArchitecture, Composant, PrimitiveDataDeclaration.*

- **Étape 2 : conversion du fichier SAML**

Cette étape est très importante dans notre projet, car nous allons utiliser des règles de transformation pour passer de ThingML à SAML, ce qui se traduira par un schéma CAPS. Toutes ces transformations sont essentiellement une transformation ThingML en un modèle SAML-CAPS où la sortie sera générée automatiquement à partir des symboles dans les modèles. Parmi les règles de transformation dont nous avons parlé :

- ✓ Convertit les objets de la classe *SoftwareArchitecture* dans le modèle SAML-CAPS en éléments de la classe *ThingMLModel* dans le modèle ThingML.

- ✓ Convertir les éléments de classe *Composant* dans le modèle SAML-CAPS en éléments de classe *Thing* dans le modèle ThingML.

- ✓ Convertissez la classe *PrimitiveDataDeclaration* dans le modèle SAML-CAPS en classe *Property* dans le modèle ThingML.

- ✓ Convertissez la classe *PrimitiveDataDeclaration used message exchanging* dans le modèle SAML-CAPS en classe *Message* dans le modèle ThingML.

- ✓ Convertissez la classe *InMessagePort* dans le modèle SAML-CAPS en classe *RequiredPort* dans le modèle ThingML.

- ✓ Convertissez la classe *OutMessagePort* dans le modèle SAML-CAPS en classe *ProvidedPort dans* le modèle ThingML.

- ✓ Convertissez la classe *Mode* dans le modèle SAML-CAPS en classe *State* dans le modèle ThingML.

- ✓ Convertissez la classe *Event* dans le modèle SAML-CAPS en classe *Event* dans le modèle ThingML.

- ✓ Convertissez la classe *Condition* dans le modèle SAML-CAPS en classe *Guard* dans le modèle ThingML.

- ✓ Convertissez la classe *Action* dans le modèle SAML-CAPS en classe *Action* dans le modèle ThingML.

- ✓ Convertissez la classe *Action other then send message* dans le modèle SAML-CAPS en classe *Function* dans le modèle ThingML.

- ✓ Convertissez la classe *Link* dans le modèle SAML-CAPS en classe *Transition* dans le modèle ThingML.

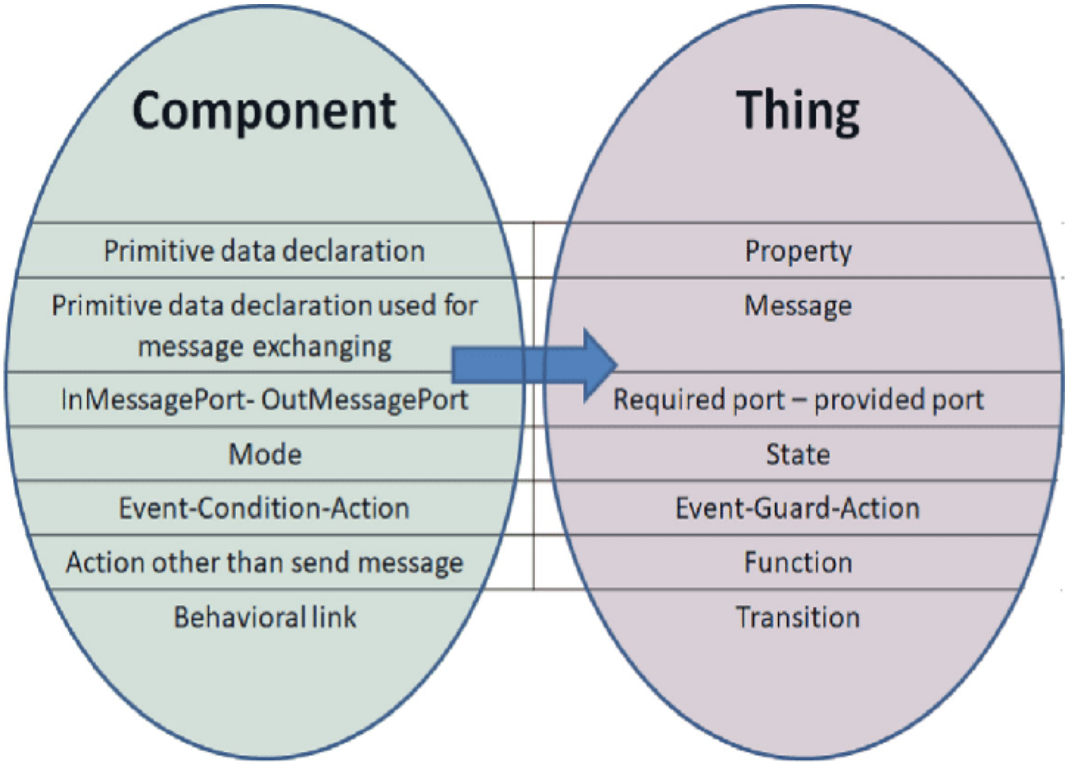


Figure 4.5 Mappage entre les éléments Component dans SAML et les éléments Thing dans ThingML [41]

- **Étape 3 : extraction du fichier ThingML**

Dans cette dernière étape, nous aurons le fichier ThingML. Xmi qui est le langage de programmation ThingML

4.4 Conclusion

Ce chapitre présente une solution optimale pour savoir comment extraire un modèle de CAPS du code ThingML, puis effectuer l'opération inverse, et nous expliquons chacun des titres suivants : fonctionnement général et Architecture de ThingML et Meta Model SAML et étapes de notre processus.

Dans le chapitre suivant, nous expliqueons notre application pratique sur le programme Eclipse et comment créer un modèle caps à partir du code ThingML et faire le processus inverse.

Chapitre 5 : Implémentation

5.1 Introduction

Dans ce chapitre, nous développons un mécanisme d'implémentation et de modélisation pour un schéma Caps via le code ThingML, un langage de programmation pour les systèmes IoT. Ensuite, nous effectuons le processus inverse, qui consiste à convertir le modèle CAPS en ThingML. Dans ce qui suit, nous expliquerons ce que nous avons fait dans notre travail.

5.2 Langages de programmation

Nous avons utilisé Java comme langage de programmation. Java est un langage de programmation largement utilisé, spécialement conçu pour être utilisé dans l'environnement distribue d'Internet. C'est le langage de programmation le plus populaire pour les applications de Smartphone Androïde, il est ´également l'un des plus utilisés pour le développement des appareils de pointe et de l'Internet des objets. On cite les avantages principaux de java [28] :

- Java est un langage orienté objets.
- Portable : pris en charge par différents matériels et offre une connectivité s´sécurisée, ce qui le rend plus préférable pour le système IoT.
- Créer des applications complètes s'exécutant sur un seul ordinateur ou être réparties entre des serveurs et des clients d'un réseau.
- Créer un petit module d'application ou une applet a utilisé dans le cadre d'une page Web.
- D´evelopper des applications pour les appareils mobiles. Tout d'abord, pour faire la programmation en Java, il faut t´télécharger et installer le kit de développement Java (JDK)

5.3 JDK

JDK (Java Développement Kit) est un environnement de développement utilisé pour le d´envelopper des applications Java. Il inclut Java Runtime Environnement (JRE), un compilateur (java.c), un générateur de documentation (java.doc) et d'autres

outils n´nécessaires au développement Java. Pour exécuter des applications Java, télécharger simplement le JRE. Cependant, pour d évelopper des applications Java ainsi que les exécuter, le JDK est nécessaire. [29].

5.4 Extraire l'éditeur d'architecture CAPS

Nous présentons ici toutes les étapes que nous suivons pour extraire l'ingénierie des systèmes CAPS

5.4.1 Eclipse Modeling Framework

Eclipse Modeling Framework (EMF) est un cadre de modélisation basé sur Eclipse et une fonction de génération de code pour la création d'outils et d'autres applications basées sur un modèle de données structuré.

À partir d'une spécification de modèle décrite dans XML Metadata Interchange (XMI), EMF fournit des outils et un support d'exécution pour produire un ensemble de classes Java pour le modèle, un ensemble de classes d'adaptateur qui permettent l'affichage et l'édition basée sur des commandes du modèle, et une base éditeur. Les modèles peuvent être spécifiés à l'aide de documents Java, UML, XML annotés ou d'outils de modélisation, puis importés dans EMF. Plus important encore, EMF fournit la base de l'interopérabilité avec d'autres outils et applications basés sur EMF [30].

- **Création de modèle Ecore**

Le méta-modèle Ecore génère les classes d'entités de notre application, pour le créer nous suivons

Ces étapes :

- Tout d'abord, nous créons un projet EMF vide en cliquant sur "File" -> "New" -> "Other...", choisissez parmi Eclipse Modeling Framework "Empty EMF Project", puis cliquez sur

Sur "Suivant" et en donnant un nom à ce projet puis "Terminer" comme le montre la figure 5.1.

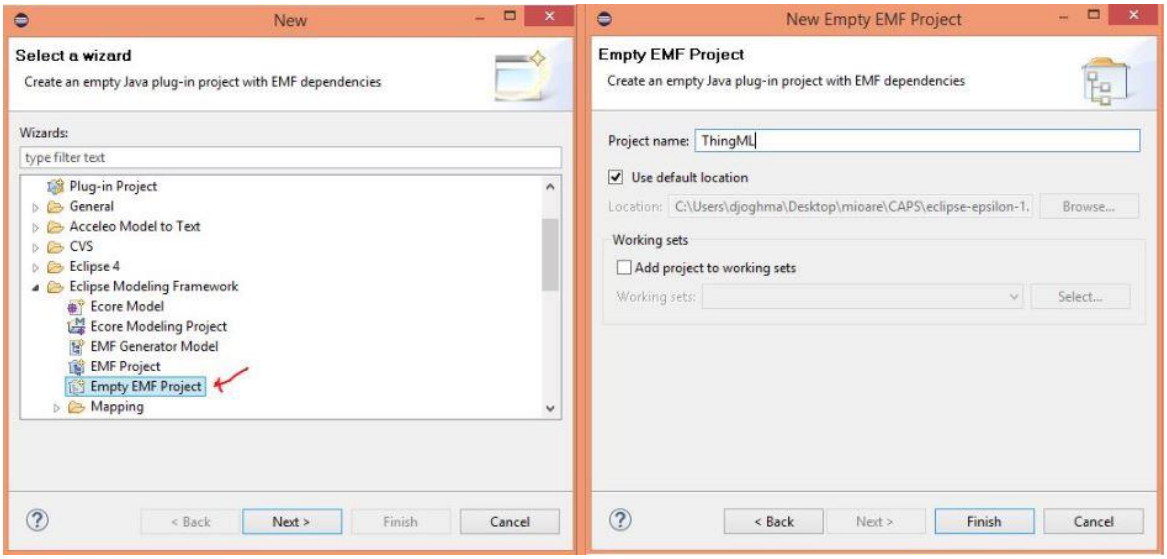


Figure 5.1 : Étapes de création d'un projet EMF vide.

- Maintenant, nous créons le fichier Ecore. Tout d'abord, nous allons dans le dossier "modèle" sur le projet créé, en faisant un clic droit "New" ! "Other" et en choisissant dans Eclipse Modeling Framework "Ecore Model", en cliquant sur "Suivant" puis en donnant un nom et en cliquant Bouton "Finish" comme le montre la figure 5.2

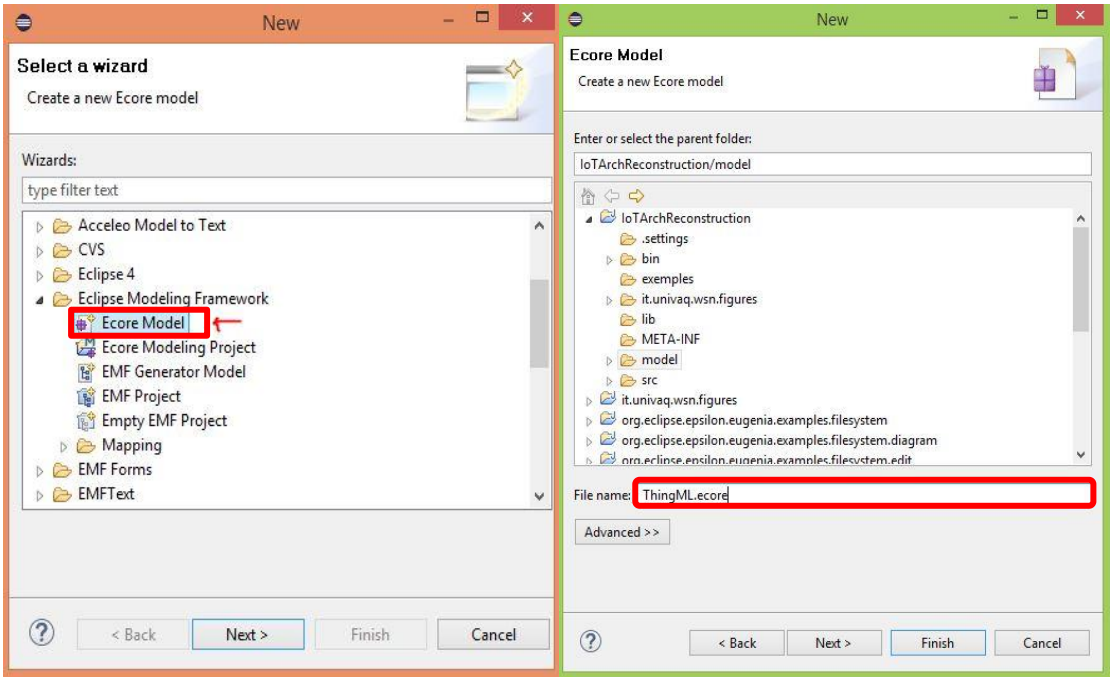


Figure 5.2 : Étapes de création du méta-modèle Ecore.

- Dans la vue des propriétés, nous donnons au package du nouveau modèle un nom et un URI comme illustré à la figure 5.3. Nous avons nommé le package et

le préfixe Ns du même "thingML", et changez l'URI Ns en" http://www.thingml.org/xtext/ThingML".

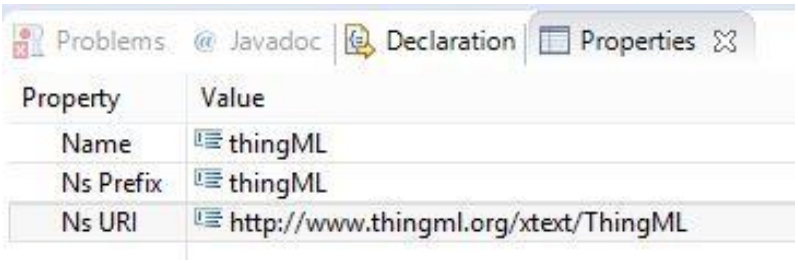


Figure 5.3 : La vue Propriétés.

- Maintenant, ajout des méta-classes du modèle Ecore (thingMLModel, Function, Property,...), et elles sont de type EClass. Notre modèle Ecore final "thingML.ecore" est montré dans la Figure 5.4.

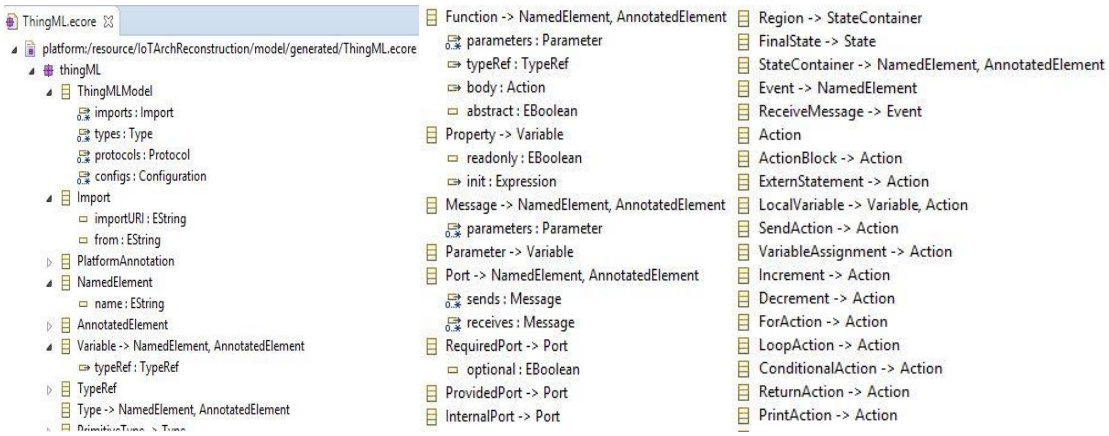


Figure 5.4 : Le modèle Ecore "ThingML.ecore".

- **Création de Genmodel**

Le fichier Genmodel contient des informations pour la génération des entités du modèle Ecore et code, pour le créer, nous suivons ces étapes:

- Tout d'abord, nous montrons la vue " GMF Dashboard " en cliquant sur " Window " ->" Show View " -> " other..." dans le dossier "général" choisissez "tableau de bord GMF", comme la figure 5.5 montrant

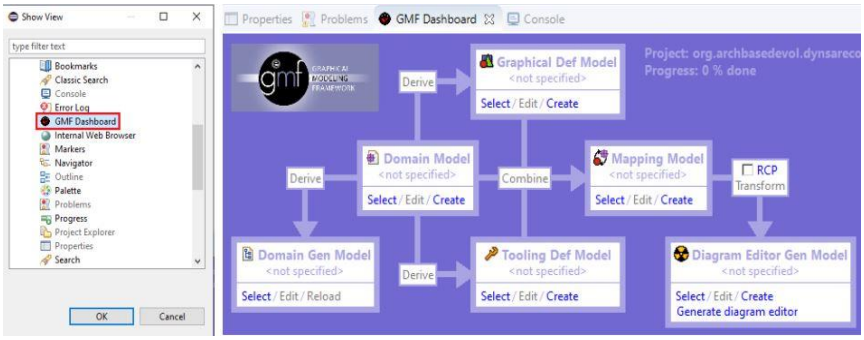


Figure 5.5 : Vue du tableau de bord du GMF.

- Depuis la vue GMF Dashboard, on clique sur "Select" de "Domain Model" puis choisir le modèle Ecore créé -> "Drive". Nous donnons un nom à ce genmodel "ThingML.genmodel", en cliquant sur " Next "-> Modèle Ecore -> " Next " -> " Load " -> " Finish ". (Voir Figure 5.6)

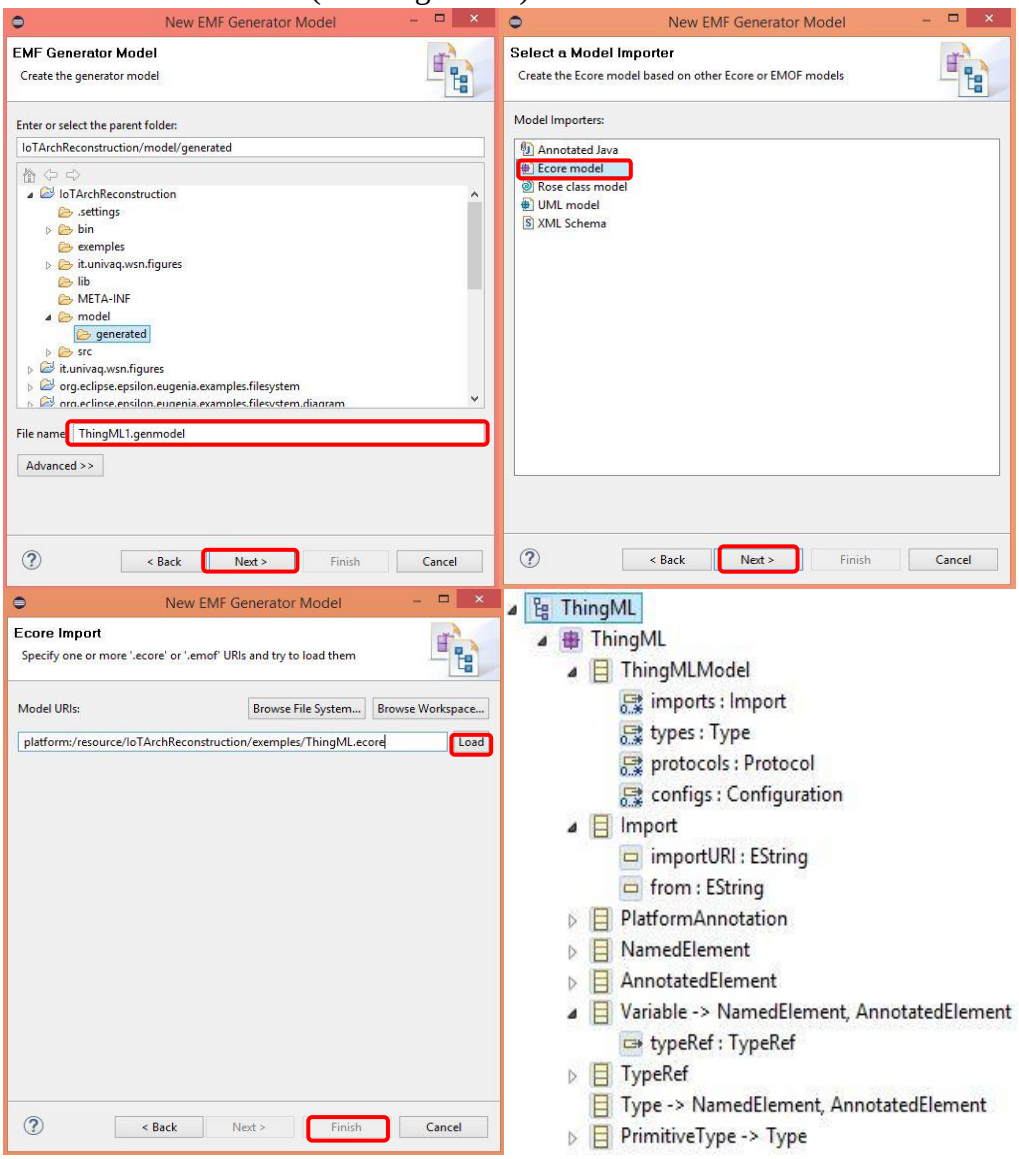


Figure 5.6 : Étapes de création de « ThingML.genmodel ».

- Ensuite, nous allons au nœud racine du "ThingML.genmodel" pour définir le niveau de conformité à 6.0 à partir de la vue des propriétés. Nous faisons un clic droit sur le nœud racine de genmodel et choisissons " Generate All ", cela donnera trois autres plugins "edit", " editor", et "tests" comme le montre la figure 5.7.

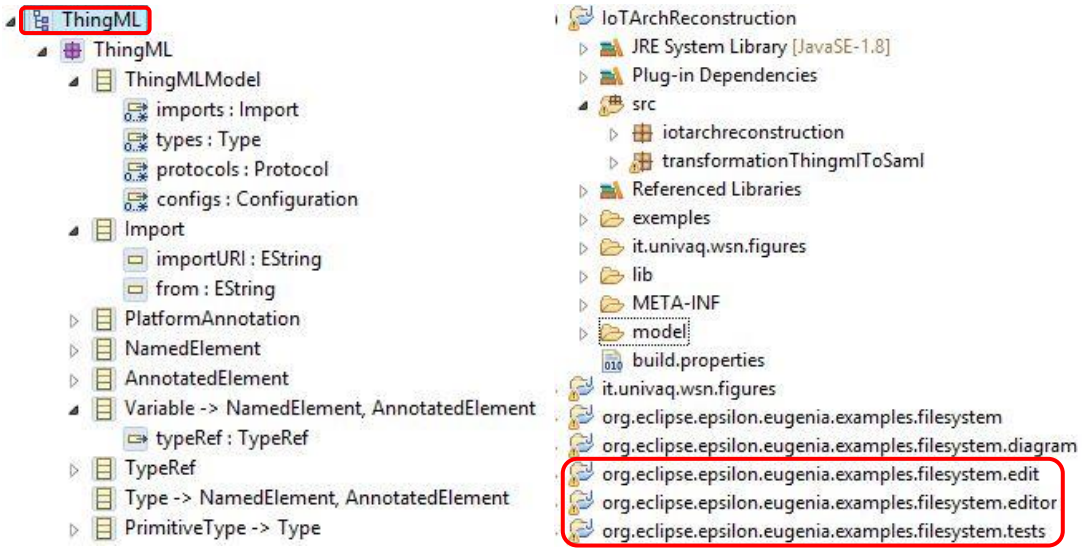


Figure 5.7 : Étapes de génération du code source.

5.4.2 Comment générer du code SAML

- Tout d'abord, nous créons une classe que nous appelons "TML2SAML" et c'est la classe dans laquelle nous allons programmer les règles de conversion comme le montre la figure 5.8

```
public class TML2SAML {

    private ResourceSetImpl resourceSetThingML;
    private ResourceSetImpl resourceSetSAML;

    CapsSAMLFactory samlFactory = CapsSAMLFactory.eINSTANCE;
    public TML2SAML() {
        initialize();
        initializeSAML();
    }
```

Figure 5.8 : Créer une classe "TML2SAML"

- Nous créons une méthode "initialize SAML" pour model "saml" et Nous créons une méthode "initialize " pour model "ThinML" qui Ce sont deux blocs de code qui n'a pas de nom ou de type de données associé et qui est placé en dehors de

toute méthode, constructeur ou autre bloc de code comme le montre la figure 5.9.

```
private void initializeSAML() {
    // TODO Auto-generated method stub


    // Create a resource set to hold the resources.
    resourceSetSAML = new ResourceSetImpl();
            // Register the appropriate resource factory to handle all
            // extensions.
    resourceSetSAML.getResourceFactoryRegistry().getExtensionToFactory
        .put(org.eclipse.emf.ecore.resource.Resource.Factory.Registry.
        // Register the package to ensure it is available during loadi
    resourceSetSAML.getPackageRegistry().put(CapsSAMLPackage.eNS_URI,

}


private void initialize() {



    // Create a resource set to hold the resources.
                resourceSetThingML = new ResourceSetImpl();
            // Register the appropriate resource factory to handle al
            // extensions.
                resourceSetThingML.getResourceFactoryRegistry().getEx
        .put(org.eclipse.emf.ecore.resource.Resource.Factory.Registry
        // Register the package to ensure it is available during load
            resourceSetThingML.getPackageRegistry().put(ThingMLPackag

}
```

Figure 5.9 : Créer "initializeSAML"et "initialize"

- Créez ensuite une méthode "loadThingML FromFile" comme le montre la figure 5.10.

```
public ThingMLModel loadThingMLFromFile(File inputThingMlModel) {

    URI uri = URI.createFileURI(inputThingMlModel.getAbsolutePath());
    org.eclipse.emf.ecore.resource.Resource resource = resourceSetThin
    ThingMLModel Thingmlmodel = (ThingMLModel) resource.getContents().

    return  Thingmlmodel;
```

Figure 5.10 : Créer "initializeSAML"et "initialize"

- Ensuite, nous créons une procédure "saveSAML Model" comme le montre la figure 5.11.

```
public void saveSAMLModel(File outputFile, SoftwareArchitecture sAMLModel) {


    URI uri = URI.createFileURI(outputFile.getAbsolutePath());
    org.eclipse.emf.ecore.resource.Resource resource = resourceSetSAML.creat
    try {
        resource.getContents().add(sAMLModel);

   FileOutputStream outputStream = new FileOutputStream(outputFile);
        resource.save(outputStream,null);

    //  resource.save(null);

    } catch (IOException e) {
        e.printStackTrace();
    }catch( java.lang.UnsupportedOperationException e){
        e.printStackTrace();
        //System.out.println(e.getMessage());
    }
```

Figure 5.11 : Créer procédure " saveSAML Model"

- À ce Etape, notre travail principal consiste à définir les règles de conversion en passant du modèle ThingML au modèle SAML comme le montre la figure 5.12.

```
SoftwareArchitecture transformToSaml(Vector<ThingMLModel> thingmlModels)
{

    SoftwareArchitecture samlSAModel = samlFactory.createSoftwareArchitectur

    for (ThingMLModel thingMLModel : thingmlModels) {

        for (Type type : thingMLModel.getTypes()) {

            if(type instanceof Thing){
```

Figure 5.12 : procédure de transformationToSaml

- Convertir Property en primitivedatadeclaration

  Nous convertissons la Property et ses attributs en modèle ThingML en primitivedatadeclaration en modèle SAML comme le montre la figure 5.13.

```
//  règle 1 :  from Properies To Primitive Data Declaration .....

for (Property property : thing.getProperties()) {

  PrimitiveDataDeclaration primitivedatadeclaration =
          samlFactory.createPrimitiveDataDeclaration()  ;
  primitivedatadeclaration.setDataName(property.getName());
  primitivedatadeclaration.getType();

component.getApplicationData().add(primitivedatadeclaration);
samlSAModel.getSAElements().add(component);

//  }
}
```

Figure 5.13 : Convertir Property en primitivedatadeclaration

- Convertir Message en primitivedatadeclaration

  Nous convertissons la Message et ses attributs en modèle ThingML en primitivedatadeclaration en modèle SAML comme le montre la figure 5.14.

```
//  règle 2 :  from message To Primitive Data Declaration .....

for (Message message : thing.getMessages()) {

    PrimitiveDataDeclaration primitivedatadeclaration2
    = samlFactory.createPrimitiveDataDeclaration()  ;
    primitivedatadeclaration2.setDataName(message.getName());
    //primitivedatadeclaration.setType("");
    //primitivedatadeclaration.setValue("");
    // primitivedatadeclaration.setValueExp("");
    component.getApplicationData().add(primitivedatadeclaration2);
    samlSAModel.getSAElements().add(component);

}
```

Figure 5.14 : Convertir Message en primitivedatadeclaration

- Convertir Requiredport en InMessagePort et Convertir Providedport en OutMessagePort

  Nous convertissons Requiredport et ses attributs en modèle ThingML en InMessagePort en modèle saml et nous convertissons Providedport et ses attributs en modèle ThingML en OutMessagePort en modèle saml comme le montre la figure 5.15.

```
//  règle 3 :  from requiredport To inmessageport.....
for (Port requiredport : thing.getPorts()){
    InMessagePort inmessageport = samlFactory.createInMessagePort()  ;
    inmessageport.setName(requiredport.getName());
    component.getPorts().add(inmessageport);
   samlSAModel.getSAElements().add(component);
}

//  règle 4 :  from providedport To outmessageport.....

for (Port providedport : thing.getPorts()){

    OutMessagePort outmessageport = samlFactory.createOutMessagePort()  ;
    outmessageport.setName(providedport.getName());
        component.getPorts().add(outmessageport);
       samlSAModel.getSAElements().add(component);

    }
```

Figure 5.15 : Convertir Requiredport en InMessagePort et Convertir Providedport en OutMessagePort

- Convertir State en Mode

  Nous convertissons State et ses attributs en modèle ThingML en Mode en modèle saml comme le montre la figure 5.16.

```
            //  règle 5 :  from state To mode.....

      if(thing.getBehaviour()!= null  ){
      Mode mode = samlFactory.createMode();

    mode.setName(thing.getBehaviour().getName());

    component.getModes().add(mode);
      samlSAModel.getSAElements().add(component);
      }
```

Figure 5.16 : Convertir State en Mode

- Dans les dernières étapes de cette recherche, nous allons créer une "classe principale" et lire le fichier "inputthingMLModel.xmi" et extraire le fichier "outputSoftwareArchitectureModel.xmi", ce qui nous permettra de créer un architecture Caps comme le montre la figure 5.17.

```
public static void main(String[] args) {
    // TODO Auto-generated method stub


     TML2SAML t2s = new TML2SAML ();

    File inputThingMlModel = new File("exemples/inputthingMLModel.xmi");

    ThingMLLoader thingMLLoader = new ThingMLLoader();

     ThingMLModel  thingMLModel = thingMLLoader.loadThingMLFromFile
             (inputThingMlModel);


      Vector<ThingMLModel> thingmlModels  = new Vector<ThingMLModel>();
      thingmlModels.add(thingMLModel);
      SoftwareArchitecture softwareArchitecture = t2s.transformToSaml
              (thingmlModels);



File outputFile = new File("exemples/outputSoftwareArchitectureModel.xmi");

t2s.saveSAMLModel(outputFile,softwareArchitecture) ;
```

Figure 5.17 : Classe main de Project

5.4.3 Créer l'éditeur graphique CAPS

Le Framework de modélisation graphique (CAPS) fournit un composant configurable et un cadre d'exécution pour le développement d'éditeurs graphiques basés sur le cadre de modélisation Eclipse (EMF) On utilise l'éditeur graphique pour afficher la structure du programme IoT récupéré Faciliter le processus de développement pour les développeurs. Pour le créer, nous suivons ces étapes :

- Tout d'abord, nous cliquons avec le bouton gauche de la souris sur le fichier "Friends", puis cliquez sur "Run As" puis choisissez " Eclipse application ".

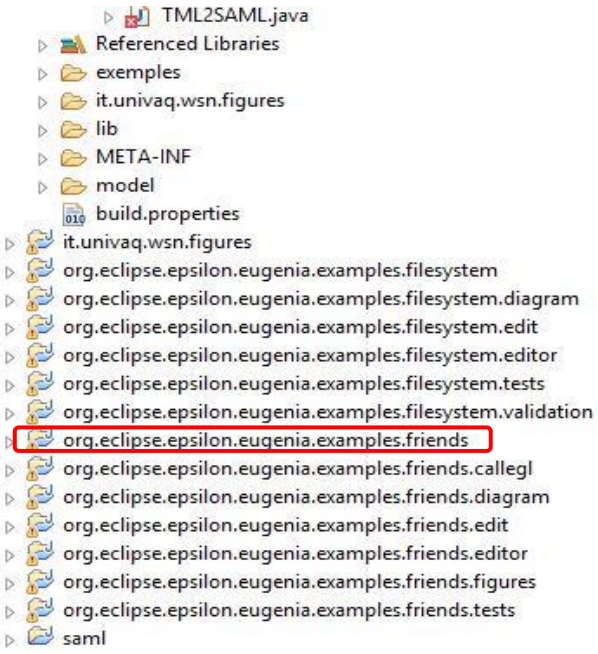


Figure 5.18: Étapes pour afficher le Framework Caps

- Ensuite vous allez ouvrir le programme Eclipse, vous créez un projet en appuyant sur "file" puis sur -> "New" puis sur -> "Project" puis sur -> "General Project" comme le montre la figure 5.19.

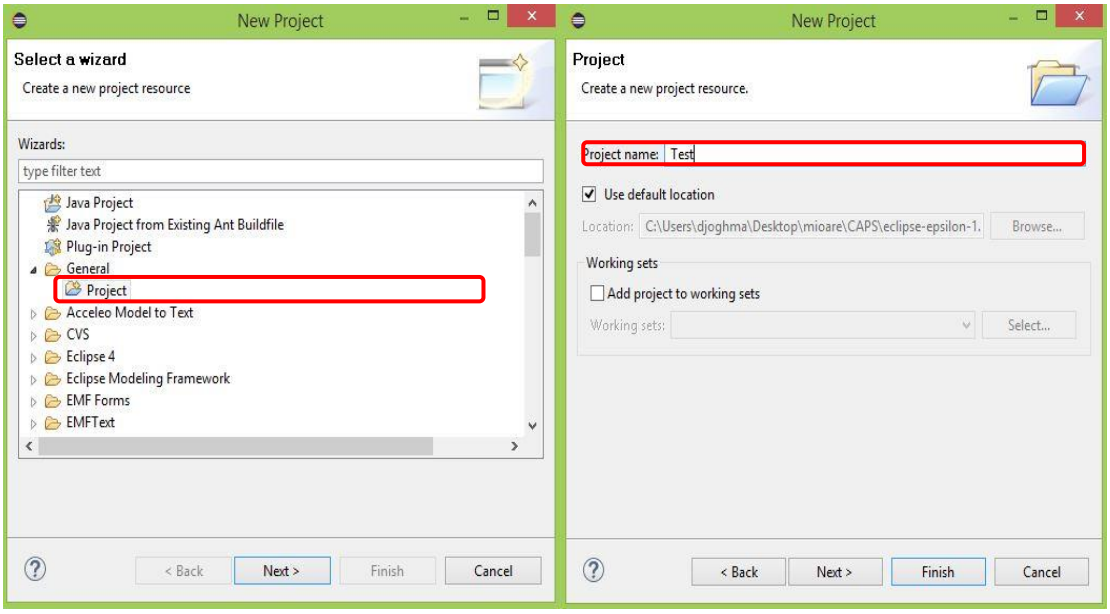


Figure 5.19: Créer un nouveau fichier dans Eclipse

- Après cette étape, on sélectionne "File" puis sur -> "New" puis sur -> "Other" puis sur -> "Example" puis sur -> "Friends diagrame " puis sur -> "Next " Choisissez le nom du fichier puis cliquez sur " Finish " comme le montre la figure 5.20

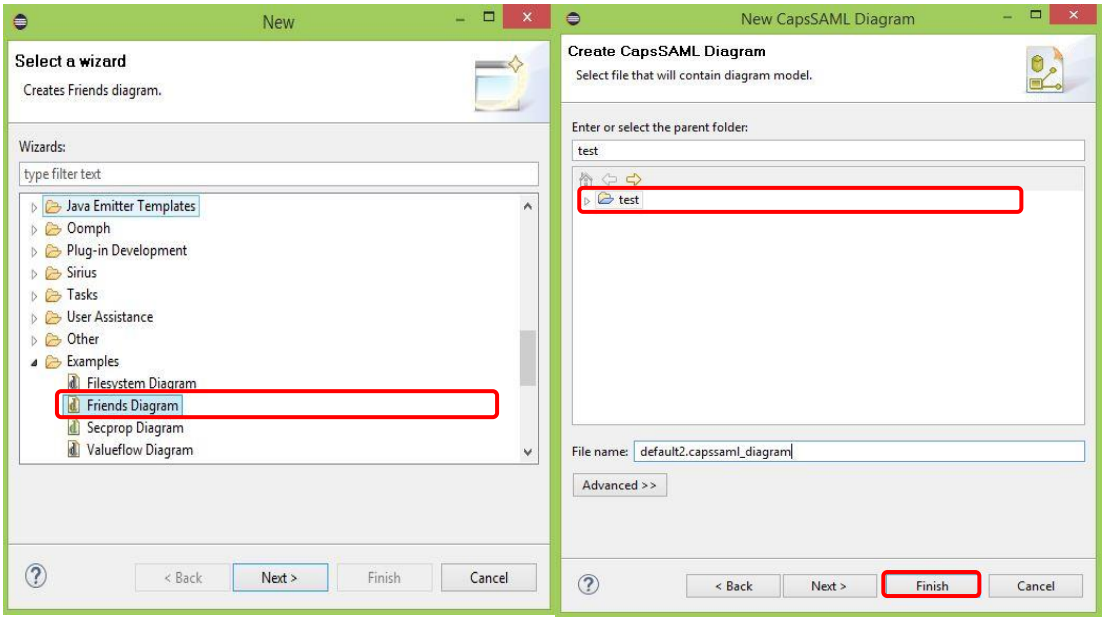


Figure 5.20: Créer un diagramme CAPS

- Il va créer pour nous une interface graphique contenant tous les composants comme "connection" et "mode" et "Alarm" et "CCTVCcapturev"… comme le montre la figure 5.21

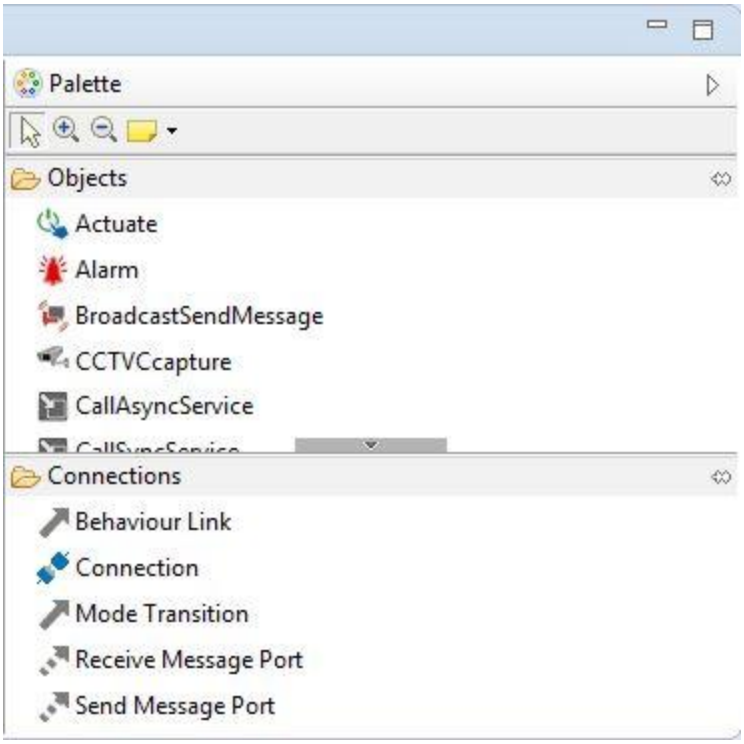


Figure 5.21: élément de diagramme CAPS

5.4.4 Mise en place de la procédure d'extraction des schémas CAPS à partir du code ThingML

- Avant de commencer la tâche de développement du système, nous devons créer une version XMI

Un exemple de ce code comme le montre la figure 5.22 un code qui contient le client et le serveur pour envoyer et recevoir des informations les uns avec les autres.

- Initialisation du modèle.
- Créer un contenu de modèle ThingML.
    1. Nous créons d'abord un 'ThingMLModel' qui est un conteneur pour les composants de la structure du programme.
    2. Maintenant, nous créons une "chose" pour chaque paquet de la structure du programme et tous ses attributs.
- Maintenant, pour chaque 'chose' que nous avons créée, nous créons les éléments qui lui sont associés (Property, Message, RequiredPort, ProvidedPort, State, Event, Guard), Il va créer un fichier xmi pour nous comme le montre la figure 5.23
- Enregistrez un fishier XMI.

```
import "datatypes.thingml" from stl

thing fragment PingMsgs {
    message ping(seq : UInt8);
    message pong(seq : UInt8);
}


thing PingServer includes PingMsgs {

    provided port ping_service {
        sends pong
        receives ping
    }

    statechart init Active {

        state Active {
            transition -> Active event m : ping_service?ping action ping_service!pong(m.seq)
        }
    }
}

   thing PingClient includes PingMsgs {

    required port ping_service {
        receives pong
        sends ping
    }

    statechart PingClientMachine init Ping {

    property counter : UInt8 = 0
```

Figure 5.22: Code ThingML

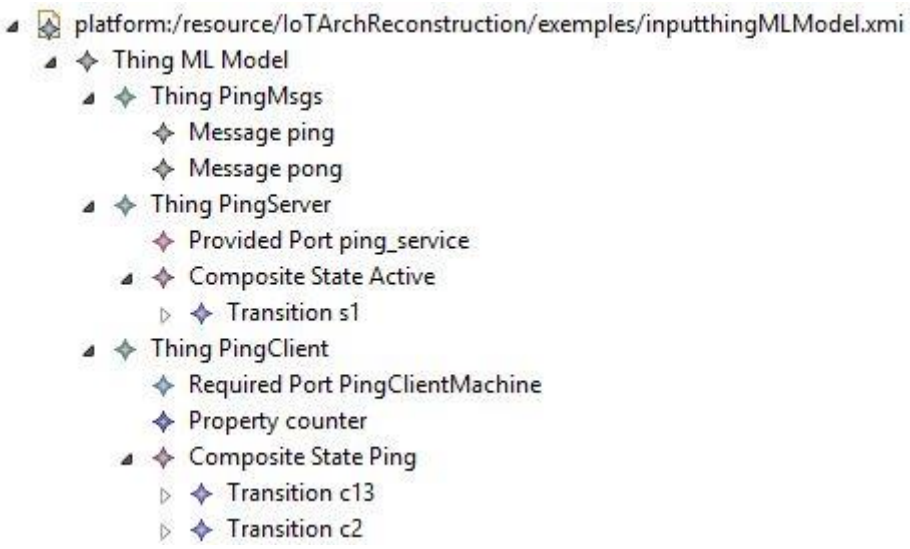


Figure 5.23: fichier ThingML.xmi

- Après cette étape, nous appuyons sur le bouton de exécution pour créer un nouveau fichier pour nous appeler "outputSoftwareArchitectureModel.xmi" comme le montre la figure 5.24

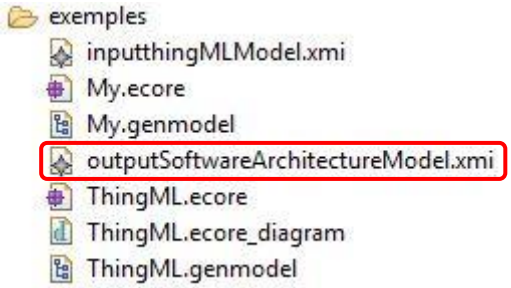


Figure 5.24: fichier OutPutSaml

- Nous ouvrons ce fichier pour le trouver comme suit, puisque grâce aux règles de conversion nous avons converti Thing en Component et Property en PrimitiveDataDeclaration et Message en PrimitiveDataDeclaration et RequiredPort en InMessagePort et ProvidedPort en OutMessagePort comme le montre la figure 5.25

```
<?xml version="1.0" encoding="ASCII"?>
<components:SoftwareArchitecture xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:components="components" xsi:schemaL
    <applicationData xsi:type="components:PrimitiveDataDeclaration" dataName="ping"/>
    <applicationData xsi:type="components:PrimitiveDataDeclaration" dataName="pong"/>
  </SAElements>
  <SAElements xsi:type="components:Component">
    <ports xsi:type="components:InMessagePort" name="ping_service"/>
    <ports xsi:type="components:OutMessagePort" name="ping_service"/>
    <modes name="Active"/>
  </SAElements>
  <SAElements xsi:type="components:Component">
    <ports xsi:type="components:InMessagePort" name="PingClientMachine"/>
    <ports xsi:type="components:OutMessagePort" name="PingClientMachine"/>
    <modes name="Ping"/>
    <applicationData xsi:type="components:PrimitiveDataDeclaration" dataName="counter"/>
  </SAElements>
</components:SoftwareArchitecture>
```

Figure 5.25: fichier outputSoftwareArchitectureModel.xmi

- Dans la dernière étape,et Après avoir copié le contenu de fichier outputSoftwareArchitectureModel.xmi créez l'éditeur graphique CAPS et suivez les étapes comme indiqué dans les Figures 5.18 et 5.19 et 5.20. Ensuite, nous coller le contenu du fichier *outputSoftwareArchitectureModel.xmi* dans le fichier default. Capssaml comme le montre la figure 5.26

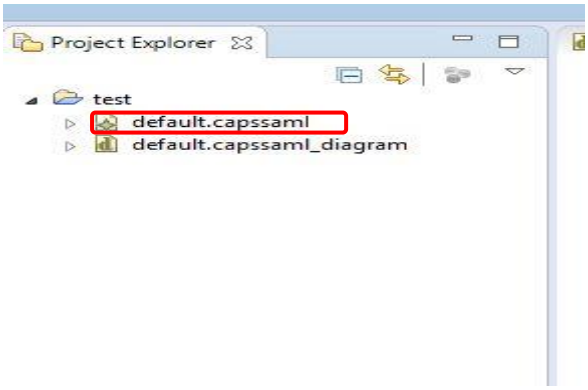


Figure 5.26 : fichier default. Capssaml

Cela permet de créer un diagramme Caps comme résultat final comme le montre la Figure 5.27. Le modèle montre trois composants et deux connecteurs.

- Component 1 : Contient deux PrimitiveDataDeclaration

    - PrimitiveDataDeclaration 1.1 son nom ping
    - PrimitiveDataDeclaration 1.2 son nom pong

- Component 2 : Contient un OutMessagePort et InMessagePort et un mode

    - OutMessagePort son nom ping_service
    - InMessagePort son nom ping_service
    - Mode Active

- Component 3 : Contient un OutMessagePort et InMessagePort et un mode et un PrimitiveDataDeclaration.

    - PrimitiveDataDeclaration son nom counter
    - OutMessagePort son nom PingClientMachine
    - InMessagePort son nom PingClientMachine
    - Mode son nom Ping

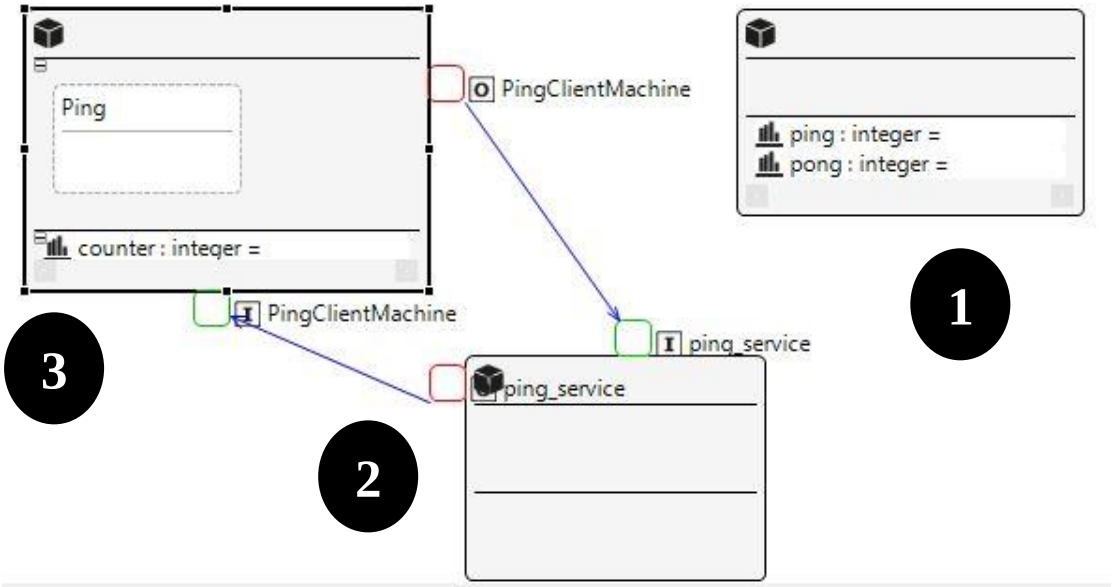


Figure 5.27 : Résultat final, un diagramme Caps

5.4.5 Conclusion

La partie réalisation donne une idée plus précise de notre proposition. Il explique de manière pratique notre travail de conversion du code thingml en code saml, puis d'extraction du modèle de Caps, puis de l'inverse. Enfin, grâce à cette modélisation, nous avons pu montrer l'utilité de notre proposition.

Conclusion générale

Reconstruire l'architecture des systèmes IOT signifie généralement extraire les schémas des systèmes IOT à l'aide de langages de programmation tels que ThingML.

L’objectif de la reconstruction de l'architecture des systèmes est de réduire le temps, efforts des développeurs pour faire des évolutions et faire face à la complexité des systèmes. Dans ce projet, notre objectif était d'aider les développeurs à développer et à maintenir leurs systèmes d'une manière plus rapide et plus facile par l’utilisation des architectures au lieu d’analyser le code source des systèmes IoT. Nous avons proposé de récupérer les architectures IOT à partir du code ThingML. Ces architectures sont créées avec le langage SAML, qui sont ensuite visualisées avec le Framework Caps. Notre processus commence par la définition des règles de transformation en se basant sur les métamodèles des langages de ThingML et de SAML. L’objectif était donc de reconstruire l'architecture des systèmes pour les systèmes IoT qui permettent aux développeurs de mettre à jour ou de supprimer des composants existants ou d'en ajouter de nouveaux. Le développeur commence par une visualisation graphique de l’architecture générée, afin de comprendre le système. Puis, il réalise ses éventuelles modifications. A la fin, la nouvelle architecture logicielle sera générée. Nous avons implémenté notre processus sous forme d’un projet Java.

Notre futur travail vise à :

• Tester notre application avec des plateformes autres qu'Eclipse pour prouver son efficacité.

• Evaluer l’efficacité de de notre processus sur des application IOT réelles.

REFERENCES:

[1] Vikas Hassija, Vinay Chamola, Vikas Saxena, Divyansh Jain, Pranav Goyal, and Biplab Sikdar. A survey on iot security: application areas, security threats, and solution architectures. *IEEE Access*, 7 :82721{82743, 2019.

[2] connectwave.fr. (2020). *Comment se compose un système IoT ?* consulté le 04 10, 2022 Récupéré sur connectwave.fr: https://www.connectwave.fr/techno-appli-iot/iot/reseaux-et-infrastructures-iot/

[3] jean paul khorez, ezikola mazoba. (2015). L'étude de internet des objets et contrôle d'accès aux données. Consulté le 04 10, 2022, sur memoireonline.com.

[4] Dr. Ovidiu Vermesan SINTEF, Norway, Dr. Peter FriessEU, Belgium, "Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems", river publishers' series in communications, 2013.

[5] Pierre-Jean Benghazi ‹Sylvain Bureau ‹Françoise Massit-Folléa (2015). *Importance et enjeux de l'internet des objets.* MSH.

[6] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer networks*, 52(12):2292{2330, 2008

[7] Daniele Puccinelli and Martin Haenggi. Wireless sensor networks: applications and challenges of ubiquitous sensing. *IEEE Circuits and systems magazine*, 5(3) :19{31, 2005. 73.

[8] CENTER, AUTOMATION&PLC KNOWLEDGE. Les domaines d'application de l'IoT et du machine to machine. [En ligne]. *Disponible sur:< https://www. Automation sense. Com/blog/blogobjets-connectes/les-domaines-d-application-de-l-iot-et-du-machine-to-machine. Html> (consulté le 02/02/2018)*.

[[9] acervolima : Avantages et inconvénients de l'IOT Consulté le 10 6, 2022 sur https://fr.acervolima.com : https://fr.acervolima.com/avantages-et-inconvenients-de-l-iot/

[10] wikipedia. (2021, 12 02). *Architecture logicielle*. Récupéré sur wikipedia: https://fr.wikipedia.org/wiki/Architecture_logicielle Consulté le 14 04, 2022

[11] geekforgeeks. (2021, 12 28). *Software Engineering | Software Evolution*. Consulté le 04 16, 2022, sur geekforgeeks: https://www.geeksforgeeks.org/software-engineering-software-evolution/

[12] Muccini, H., Sharaf, M.: Caps: a tool for architecting situational-aware cyberphysical systems. In: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), pp. 286–289. IEEE (2017)

[13] Sharaf, M., Abughazala, M., Muccini, H., Abusair, M.: CAPSim: simulation and code generation based on the CAPS. In: Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings, pp. 56–60. ACM (2017)

[14] Sharaf, M., Abughazala, M., Muccini, H., Abusair, M.: Simulating architectures of situational-aware cyber-physical space. In: Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings, pp. 66–67. ACM (2017)

[15] Muccini, H., Sharaf, M.: Caps: architecture description of situational aware cyber physical systems. In: 2017 IEEE International Conference on Software Architecture (ICSA), pp. 211–220. IEEE (2017)

[16] Harrand, N., Fleurey, F., Morin, B., Husa, K.E.: ThingML: a language and code generation framework for heterogeneous targets. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, pp. 125–135. ACM (2016)

[17] Sharaf, M., Abughazala, M., Muccini, H.: Arduino realization of CAPS IoT architecture descriptions. In: Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, p. 6. ACM (2018)

[18] Sharaf Mohammad, M. A. (2019, 12 03). Generating Heterogeneous Codes for IoT Systems. p. 3.

[19] H. Muccini and M. Sharaf, “Caps: Architecture description of situational aware cyber physical systems,” in *2017 IEEEInternational Conference on Software Architecture (ICSA)*, April 2017

[20 I. Malavolta, H. Muccini, and M. Sharaf, “A preliminary study on architecting cyber-physical systems,” in *Proceedings of the 2015 European Conference on Software Architecture Workshops*. ACM, 2015, p. 20.

[21] I. Crnkovic, I. Malavolta, H. Muccini, and M. Sharaf, “Onthe use of component-based principles and practices for architecting cyber-physical systems,” in *2016 19th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE)*, April 2016, pp. 23–32.

[22] C. Szyperski, *Component Software. Beyond Object Oriented Programming*. Addison Wesley, 1998

[23] ISO/IEC/IEEE, “ISO/IEC/IEEE 42010:2011 Systems and software engineering – Architecture description,” 2011.

[24] Sharaf, M. (2017). CAPS: Architecture Description of Situational Aware Cyber Physical Systems. p. 10.

[25] H. Muccini and M. Sharaf, \Caps: a tool for architecting situational-aware cyber-physical systems," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pp. 286{289, IEEE, 2017. 4, 5, 6

[26] Wikipédia. (s.d.). *Diagramme de classes.* (wikipedia, Éd.) Consulté le 04 24, 2022, sur fr.wikipedia.org: https://fr.wikipedia.org/wiki/Diagramme_de_classes

[27] N. Harrand, F. Fleurey, B. Morin, and K. E. Husa, \Thingml: a language and code generation framework for heterogeneous targets," in Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, pp. 125{135, ACM, 2016. 7, 8, 9

[28] ipeti.forumpro *definition de langage java java-script.*. Consulté le 11 05 2022, sur ipeti.forumpro.fr: http://ipeti.forumpro.fr/t21-definition-de-langage-java-java-script.

[29] wikipedia Consulté le 11 05 2022, sur fr.wikipedia.org https://fr.wikipedia.org/wiki/NetBeans

[30] wikipedia Consulté le 16 06 2022, sur fr.wikipedia. https://en.wikipedia.org/wiki/Eclipse_Modeling_Framework

[31] *Clements, Paul; Felix Bachmann; Len Bass; David Garlan; James Ivers; Reed Little; Paulo Merson; Robert Nord; Judith Stafford (2010). Documenting Software Architectures: Views and Beyond, Second Edition. Boston: Addison-Wesley. ISBN 978-0-321-55268-6.*

[32] Jump up to: *Perry, D. E.; Wolf, A. L. (1992). "Foundations for the study of software architecture" (PDF). ACM SIGSOFT Software Engineering Notes.* ***17*** *(4): 40. CiteSeerX 10.1.1.40.5174. doi:10.1145/141874.141884. S2CID 628695.*

[33] *"Software Architecture". www.sei.cmu.edu. Retrieved 2018-07-23.*

[34] trésor. Consulté le 10 6, 2022, sur www.tresor.economie.gouv.fr : https://www.tresor.economie.gouv.fr/Articles/2018/05/09/le-developpement-de-l-iot-au-japon-pilier-de-la-strategie-gouvernementale-et-opportunites-pour-les-entreprises-francaises

[35] makemybot, Consulté le 10 6, 2022,sur https://makemybot.com : https://makemybot.com/product/arduino-ultimate-kit/

[36] wikipedia, Consulté le 10 6, 2022,sur https://fr.wikipedia.org/ : https://fr.wikipedia.org/wiki/5G_pour_l%27Internet_des_objets

[37] investir. Lesechos. Consulté le 10 6, 2022, sur https://investir.lesechos.fr : https://investir.lesechos.fr/placements/vie-pratique/dossiers/faut-il-ceder-aux-sirenes-de-la-banque-en-ligne/avantages-et-inconvenients-des-banques-100-numerique-1809690.php

[38] softfluent, Quelle architecture logicielle pour son application ? Consulté le 10 6, 2022, sur https://www.softfluent.fr : https://www.softfluent.fr/blog/architecture-logicielle-pour-application/

[39] softfluent, Architecture logicielle et modèles de conception Consulté le 10 6, 2022, sur https://www.softfluent.fr/blog/architecture-logicielle-et-modeles-de-conception/

[40] dilshan ukwattage , Evolution of Software Consulté le 10 6, 2022, sur Architecture https://faun.pub/evolution-of-software-architecture-949e579f7221

[41] Sharaf Mohammad (2019). “modling and code generatore framework for IOT”

[42] omar el mendiant *Lab. LAVETE, FSTS, Hassan 1st University, Settat Morocco* “a model transformation approach for code generation from state machine diagram”