République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université MOHAMMED SEDDIK BENYAHIA de JIJEL
Faculté de Sciences Exactes et INFORMATIQUE
Département Informatique

Mémoire de Fin d'étude

Présentée pour l'obtention du diplôme de :

MASTER

En **INFORMATIQUE**

Option : Systèmes d'information et aide à la décision

Thème

Application de l'Ingénierie Dirigée par les Modèles dans le Développement des applications de l'Internet des Objets : une Approche Basée sur la Simulation

Réalisé Par :

Bouchebtoul Wissem
Siffour Asma

Encadré Par :

Mr. kerkouche Elhillali
Mr. Fortas Abdelouahab

Promotion : 2021/2022

Dédicace

A ma chère mère

Quoi que je fasse ou que je dise , je ne saurai point te remercier comme il se doit . Ton affection me couvre , ta bienveillance me guide et ta présence à mes côtés a toujours été ma source de force pour affronter les différents obstacles.

A mon chèr père

Tu as toujours été à mes côtés pour me soutenir et m'encourager . Que ce travail traduit ma gratitude et mon affection.

A mes très chers frères Chiheb et AbdArrehmane et mes belles sœurs Souhila Abir et Lina Puisse Dieu vous donne santé , bonheur , courage et surtout réussite.

A mes amis très proches

A tous les membres de ma famille et A tous ceux qui ont eu et qui ont confiance en nous.

Wissem

Dédicace

A ma très chère mère

Vous avez su porter pour nous les soins et consentir les efforts pour notre éducation . Aucune dédicace ne saurait exprimer tout le respect et l'amour que nous vous portons , vous nous avez toujours fait confiance . Veuillez trouvez en ce travail la consolation et le témoin de la patience.

A mon très cher père

Malgré les grandes responsabilités que vous assumez dans vos travaux autant que pères de familles , vous avez toujours été près de nous , pour nous écouter nous soutenir , nous suivre et nous encourager . Puisse ce travail diminuer souffrances et vous porter bonheur.

A mon cher Frère Salim et mes belles soeurs Hakima et Sabrina

Vous avez toujours été pour nous d'une aide précieuse . Nous vous remercions pour tous les bienfaits chacun a pu faire pour nous.

A tous les membres de ma famille et de la famille Boussaid. A tous ceux qui ont eu et qui ont confiance en nous.

A tous mes amis et camarades de classe, puisse Dieu conserver notre amitié.

Asma

Remerciements

Tout d'abord, nous remercions le tout puissant notre Dieu a tout son compromis de nous faire arriver à ce niveau-là et le courage et la force de continue et accomplir nos études.

Nous remercions nos famille pour leur contribution, leur soutien, leur patience et leurs sacrifices et aide pour que nous réussissions, de nous avoir donné l'importance, de prendre toujours soin de nous, de nous faire confiance et de toujours nous démontrer l'amour qu'elles nous portent.

Nous adressons nos sincères remerciements à notre encadreur Dr Kerkouche El-hillali d'avoir accepté de nous suivre dans ce projet et pour l'honneur de nous avoir guider pour réaliser ce travail.

Nous remercions notre Co-encadreur Mr Abdelouahab Fortas pour avoir consacrer son temps précieux et pour tous les efforts et l'importance qu'il nous a donnée.

Nous remercions chacun des membres du jury pour l'intérêt porté à notre travail en acceptant de l'examiner et de l'enrichir avec leurs propositions.

on remercie nos camarades de promotion, pour leurs disponibilités et leurs gentillesses.

Enfin, on adresse nos plus sincères remerciements à tous nos proches et amis, qui nous ont toujours encouragées au cours de la réalisation de ce mémoire.

Merci à tous et à toutes.

Résumé

Les systèmes de l'Internet des objets (IoT) sont basés sur des composants matériels hétérogènes allant des microcontrôleurs aux puissants serveurs cloud. Le développement d'applications IoT nécessite une expertise dans de nombreux domaines, tels que l'électronique, les protocoles de communication et les langages de programmation. Par conséquent, le développement d'applications IoT est un véritable défi. Ce mémoire présente une approche qui peut aider les utilisateurs ayant une expérience minimale du développement à créer et à tester rapidement des applications IoT. Nous proposons d'utiliser le langage spécifique au domaine ThingML pour décrire l'application d'une manière indépendante de la plate-forme. Afin de faciliter le processus de modélisation, nous développons un éditeur de modélisation hybride graphique-textuel pour le langage ThingML. Cet éditeur combine les notations textuelles et graphiques pour cumuler leurs avantages. Après la modélisation, les modèles obtenus seront utilisés pour générer le code source d'applications pour de multiples plates-formes à l'aide du cadre de génération de code ThingML. Nous adoptons une approche de simulation en utilisant le logiciel Proteus pour tester l'application. Proteus permet de concevoir le circuit matériel de l'application. De plus, il permet de simuler l'exécution du code source précédemment généré. Les résultats expérimentaux montrent que notre approche peut réduire le temps et l'effort nécessaires pour construire des applications IoT. Grâce au framework ThingML, qui permet de générer un code entièrement opérationnel. De plus, les utilisateurs peuvent tester leurs applications sans la disponibilité des dispositifs IoT.

Mots-clés *: Internet des objets, Ingénierie dirigée par les modèles, Simulation, Langage spécifique à un domaine, Génération de code*

Abstract

The Internet of Things (IoT) systems are based on heterogeneous hardware components ranging from micro-controllers to powerful cloud servers. The development of IoT applications requires expertise in many fields, such as electronics, communication protocols, and programming languages. Therefore, developing IoT applications is a challenging issue. This paper presents an approach that can help users with minimal development experiences to build and test IoT applications rapidly. We propose to use the ThingML domain-specific language to describe the application in a platform-independent way. In order to facilitate the modeling process, we develop a hybrid graphical-textual modeling editor for the ThingML language. This editor combines textual and graphical notations to accumulate their advantages. After modeling, the resulting models will be used to generate application source code for multiple platforms using the ThingML code generation framework. We adopt a simulation approach using the Proteus software to test the application. Proteus enables the design of the application hardware circuit. Furthermore, it allows simulating the execution of the previously generated source code. Experimental results show that our approach can reduce the time and effort needed to build IoT applications. Thanks to the ThingML framework, which allows the generation of fully operational code. Also, The users can test their applications without the availability of the IoT devices.

Keywords : *Internet of Things, Model Driven Engineering, Simulation, Domain-Specific language, Code generation.*

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

LISTES DES ACRONYMES

IoT	Internet of Things.
IdO	Internet des Objets.
IEEE	Institute of Electrical and Electronics Engineers.
RFID	Radio Frequency IDentification.
SYSML	System Modeling Language.
ThingML	Internet of Things Modeling Language
UML	Unified Modeling Language
UML4IOT	Unified Modeling Langage For the Internet of Things
BPMN	Business Process Modeling Notation
MDE	Model Driven Engineering
IDM	Ingénierie Dirigée par les Modèle
OMG	Object Management Group
MOF	Meta Object Facility
XML	Extensible Markup Language
ATL	Atlas Transformation Language
EMF	Eclipse Modeling Framework
DSL	Domain Specific Language
EBNF	Extended Backus Naur Form
VSM	Viewpoint Specification Model
IDE	Integrated Development Environment
CAO	Construction Assistée par Ordinateur)
PCB	Printed Circuit Board

INTRODUCTION GÉNÉRALE

Beaucoup d'entre nous ont rêvé de maisons intelligentes où les outils et les machines sont capables d'exécuter nos commandes automatiquement, la sonnette d'alarme et la machine à café font le moment où vous voulez commencer votre journée, illuminent la lumière dès que vous traversez la maison, certains ordinateurs cachés répondent à vos commandes vocales pour lire votre emploi du temps et vos messages si prêts, et allument les nouvelles à la télévision. Votre voiture peut vous conduire au travail sur la route la moins fréquentée et vous donner l'occasion de lire ou de vous préparer pour une entrevue matinale pendant que vous voyagez. Vous avez certainement lu ou vu de telles choses dans des histoires de fiction pratiques pendant des décennies, mais maintenant elles sont déjà possibles ou sur le point de le devenir. Toutes ces technologies modernes forment la base de ce que l'on appelle l'Internet des objets (IoT).

Aujourd'hui, l'IoT connaît un développement remarquable où le nombre de dispositifs liés à l'Internet a atteint des dizaines de milliards, et ce nombre devrait augmenter continuellement [1]. Son utilisation couvre divers domaines de la vie, tels que les villes intelligentes, les maisons intelligentes, l'industrie, les transports, la santé, etc. Le développement de technologies modernes telles que les réseaux de capteurs sans fil et les systèmes d'identification (RFID) a contribué à doter les objets traditionnels de nouvelles fonctionnalités telles que l'identification, la capture, la communication et le calcul [2]. Ces caractéristiques ont transformé les objets traditionnels en objets intelligents qui peuvent collecter des informations sur leur environnement, effectuer des calculs, et communiquer pour atteindre des objectifs spécifiques [3].

Les systèmes IoT sont basés sur des composants matériels hétérogènes, allant des microcontrôleurs aux puissants serveurs cloud. Ils se distinguent également par l'hétérogénéité des logiciels, des protocoles de communication et des formats de données [4], [5]. Par conséquent, le développement d'applications IoT est un véritable défi. Il nécessite une expertise dans de nombreux domaines, tels que l'électronique, les protocoles de communication et les langages de programmation.

L'ingénierie dirigée par les modèles (IDM) peut aider à relever les défis techniques de l'IoT, tels que l'hétérogénéité [5]. Cette approche décrit un système par un modèle conforme au métamodèle du langage de description. Les modèles qui en résultent facilitent la compréhension et l'analyse du système par les utilisateurs. Avec les outils IDM, il est possible de transformer automatiquement ces modèles en d'autres modèles ou codes

sources. D'autre part, les techniques de simulation offrent de nombreux avantages. Elles consistent à tester des systèmes développés sans équipement, ce qui peut aider de grands groupes à développer et à vérifier leurs applications avant la disponibilité des dispositifs IoT.

Ce mémoire propose une approche basée sur l'IDM et la simulation pour développer et tester les applications IoT. Elle peut aider les utilisateurs ayant une expérience minimale du développement à construire et à tester rapidement des applications IoT. Dans ce sens, pour surmonter le problème de l'hétérogénéité des langages de programmation, nous proposons d'utiliser l'approche ThingML [6, 7]. L'approche ThingML est basée sur un langage spécifique au domaine (DSL) et un cadre de génération de code [7]. Le DSL textuel ThingML permet de décrire les applications IoT d'une manière indépendante de la plate-forme. Dans les modèles ThingML, le comportement dynamique des composants est décrit à l'aide d'un mélange de statecharts, de communication par messages asynchrones, d'un langage d'action indépendant de la plate-forme et de langages cibles. Par conséquent, ces spécifications peuvent inclure de nombreux détails qui diminuent leur lisibilité. Dans ce contexte, nous développons un éditeur de modélisation hybride graphique-textuel pour le langage ThingML. Les éditeurs hybrides graphique-textuel présentent les meilleures solutions de modélisation qui combinent les notations textuelles avec les notations graphiques et cumulent leurs avantages [8]. Il est bien connu que la spécification graphique est mieux adaptée pour décrire les composants du système et leurs relations. D'autre part, les états et les transitions des statecharts sont spécifiés graphiquement, tandis que les gardes dans les transitions et les actions dans les états sont spécifiées à l'aide d'un langage d'expression textuel. Ainsi, chaque aspect du système sera décrit en utilisant la vue la plus appropriée (textuelle/graphique). Nous utilisons des cadres et des outils bien connus sous la plate-forme Eclipse pour atteindre cet objectif.

Le cadre de génération de code ThingML permet de générer le code source de l'application dans plusieurs langages à partir des spécifications ThingML, où plusieurs plateformes matérielles prennent en charge le code source de l'application [7]. Enfin, nous adoptons une approche de simulation en utilisant le logiciel Proteus pour tester l'application. Proteus [9] permet de concevoir le circuit matériel de l'application. Il permet également de simuler l'exécution du code source de l'application généré précédemment. Avec le logiciel Proteus, les utilisateurs peuvent tester leurs applications sans avoir besoin de disposer des dispositifs IoT. Les principales contributions de ce mémoire sont les suivantes :

— Nous développons un éditeur hybride graphique-textuel pour le langage de modélisation ThingML. Cet éditeur peut faciliter le processus de développement d'applications IoT.

— Nous proposons une approche basée sur le langage ThingML et Proteus pour développer et simuler des applications IoT.

— Nous développons une étude de cas pour démontrer notre méthodologie basée sur IDM.

— Ce mémoire démontre que les techniques IDM sont aptes à développer des langages et des outils pour répondre avec succès à la complexité des technologies hétérogènes dans le contexte des applications IoT.

Plan de mémoire :

Ce mémoire est organisé en quatre chapitres, ces chapitres sont précédés par une introduction générale et suivis par une conclusion générale :

— **Chapitre 1 :** Présente l'Internet des Objets (IoT). Il est dédié à éclaircir la notion de l'IoT, plus particulièrement : ses concepts, son historique, ses domaines d'applications, son architecture, ses exigences et ses défis.

— **Chapitre 2 :** Présente le langage ThingML, son historique, son cycle de vie et ses concepts illustrés par un exemple.

— **Chapitre 3 :** Présente notre approche, les principaux outils utilisés dans cette approche et l'éditeur graphique que nous avons développé.

— **Chapitre 4 :** Présente deux études de cas afin d'illustrer notre approche.

Ce mémoire s'achève par une conclusion générale présentant un récapitulatif du contexte de notre travail et les perspectives que nous envisageons pour compléter ce travail.

CHAPITRE 1

INTERNET DES OBJETS

1.1 Introduction

L'Internet des objets est un concept sophistiqué d'Internet de sorte que toutes les choses dans nos vies ont la capacité de se connecter à Internet ou les uns aux autres pour envoyer et recevoir des données pour effectuer des fonctions spécifiques à travers le réseau, et cette technologie est censée rendre nos vies plus simples pour améliorer notre situation, où les choses peuvent interagir les unes avec les autres d'une part et avec l'humain d'autre part pour permettre de nombreuses nouvelles applications dans les domaines médical, industriel, économique, éducatif, et même au niveau de la vie quotidienne de l'individu, la base du sujet dépend sur un scénario d'interaction en ligne pour fournir les meilleurs services aux humains, dans le sens où toutes les choses dans nos vies ont la capacité de communiquer entre elles ou avec Internet pour effectuer leurs propres fonctions spécifiques ou transférer des données entre elles via certains des capteurs spéciaux qui leur sont associés.

Dans ce chapitre, nous allons introduire les concepts généraux de l'Internet des objets (IoT), et présenter ses domaines d'applications en suite nous allons voir la modélisation des applications IoT.

1.2 Définition

L'Internet des Objets (IdO), communément appelé en anglais Internet of Things (IoT) est un terme émergent pour la nouvelle génération d'Internet qui permet la compréhension entre les appareils interconnectés (via le protocole Internet). Ces appareils comprennent divers outils, capteurs, outils d'intelligence artificielle et plus encore. Cette définition va au-delà du concept traditionnel les gens se connectent avec des ordinateurs et des smartphones sur un seul réseau mondial et via le protocole Internet traditionnel bien connu. Ce qui distingue les choses, c'est qu'elles permettent à une personne d'être libre de l'endroit, c'est-à-dire qu'une personne peut contrôler les outils sans avoir à être dans un endroit spécifique pour traiter avec un appareil particulier. Grâce à des protocoles Internet et à des traitements de données bien connus, d'autres appareils peuvent être contactés et

compris sans intervention humaine. Les innovations technologiques de l'Internet des objets permettent la surveillance, le contrôle, l'automatisation (exécution automatique des objets) et vérifiez l'état d'un large éventail d'appareils et d'objets différents pouvant être utilisés dans les maisons intelligentes, les institutions et les voitures autonomes. Donc, les choses qui communiquent et comprennent directement en ligne sans intervention humaine directe, ces choses peuvent être la voiture, la télévision ou divers articles ménagers tels que réfrigérateur, machine à laver, alarmes et climatiseurs, ou peuvent inclure des biens et des produits disponibles sur les étagères des magasins et autres.

De nombreux chercheurs ont déjà fait de nombreuses définitions de l'IoT en raison de la convergence de différentes technologies, et la définition de l'IoT est en constante évolution. L'IoT a été décrit comme "des objets dotés d'identités et de personnalités virtuelles qui fonctionnent dans des espaces intelligents en utilisant des interfaces intelligentes pour se connecter et communiquer dans des contextes sociaux, environnementaux et d'utilisation" [10], chaque objet est identifié comme un nœud et connecté les uns aux autres dans le réseau. Plus clairement, les objets d'un système en temps réel sont équipés d'éléments de détection, de microcontrôleurs, d'éléments de communication, de dispositifs de stockage et de récupération d'informations et de protocoles appropriés. Cela permet un réseau d'informations transparent et communicant activement, très robuste et intégré dans chaque objet et système IoT [11]. Pour sa part, l'IEEE définit l'IoT comme un « réseau d'éléments chacun muni de capteurs qui sont connectés à Internet ». L'IoT-GSI définit également un objet connecté comme un appareil avec les sept propriétés suivantes [12] :

- 1- Capteurs.
- 2- Connectivité à Internet.
- 3- Processeurs.
- 4- Efficacité énergétique.
- 5- Coût optimisé.
- 6- Fiabilité.
- 7- Sécurité.

1.2.1 Qu'est-ce qu'un objet ?

Un objet est une entité physique, par exemple : un livre, une voiture, une machine à café électrique ou un téléphone mobile. Dans le contexte précis de l'Internet des objets, et ce quelle que soit la vision, cet objet possède au minimum un identifiant unique attaché à une identité exprimant d'une part ses propriétés immuables (type, couleur, poids, etc.) et d'autre part son état c'est-à-dire l'ensemble de ses caractéristiques pouvant évoluer au cours du temps (position, niveau de batterie, etc.).

De nombreuses définitions s'accordent à dire qu'un objet possède des capacités de calcul, d'acquisition (capteur) et d'action (actionneur), cependant elles excluent les objets inertes identités par RFID. Ce dernier, est une puce RFID classique ne peut pas être considérée comme un dispositif de calcul, elle se résume à un identifiant stocké dans une mémoire. De même, si l'on considère les cas extrêmes, un simple code-barres peut être employé pour identifier un objet, ce dernier étant exempt d'une quelconque partie électronique ; un livre et son code ISBN, par exemple. Ainsi, il est raisonnable de considérer

que l'Internet des objets est composé d'objets actifs, capables d'accomplir des calculs, d'effectuer des mesures sur l'environnement ou d'influer sur celui-ci, et d'objets passifs qui n'ont pas d'autres aptitudes que celles d'être suivis et détectés par des objets actifs. Par extension, l'identité d'un objet passif n'est pas directement stockée dans celui-ci, à l'exception de l'identifiant, et nécessite l'utilisation d'une infrastructure tierce capable de stocker ces informations. Au contraire, un objet actif peut stocker toute partie de son identité et échanger directement ces informations avec d'autres objets actifs. Cependant, cette capacité à stocker sa propre identité n'est pas obligatoire pour un objet actif et dépend de ses ressources matérielles, notamment la mémoire, la complexité et du volume de ladite identité [13]. En IoT, chaque objet doit avoir les caractéristiques suivantes [14] :

- **L'existence :** Selon le Larousse l'existence est le fait d'avoir une réalité et d'avoir une présence en un lieu. Un objet tel qu'un véhicule existe dans le monde physique, mais grâce à un équipement de communication intégré et des technologies spécifiques associées, le véhicule possède également une identité (une existence / une réalité) dans le monde virtuel.
- **L'autonomie :** L'autonomie est la capacité d'un objet à se contrôler soi-même. Elle représente les propriétés d'une entité qui est capable de fonctionner de manière indépendante et sans être commandé de l'extérieur. En effet, chaque objet possède une identité lui représentant, peut librement rassembler, analyser, traiter, générer et échanger des informations. Il peut également prendre des décisions sans aucune intervention humaine.
- **La connectivité :** C'est à dire ce qu'une entité offre comme connexion à d'autres entités de son environnement. Ainsi, n'importe quel objet autorisé peut initier une communication avec d'autres objets et utiliser leurs informations.
- **L'interactivité et l'interopérabilité :** l'internet des objets est un concept général qui englobe différents domaines utilisant des systèmes, architectures et matériels différents. Par conséquent, un objet doit pouvoir interagir et collaborer avec d'autres objets hétérogènes.

Ces derniers peuvent être des humains ou des machines, réelles ou virtuelles, qui produisent et consomment une grande variété de services.

1.2.2 Les Éléments d'Internet des Objets

On présente ici une taxonomie qui aidera à définir les composants nécessaires à L'Internet des objets. Trois composants IoT permettent l'emplacement parfait :

- Hardware composé de capteurs, actionneurs et hardware de communication intégré (Radio Frequency Identification (RFI), Wireless Sensors Network (WSN), WLAN, Bluetooth LowEnergy (BLE), ZigBee) ;
- Outils de stockage et computation de borde (frontière) sur demande, pour l'analyse des données ;
- Présentation et interface de visualisation facile à comprendre et outils d'interprétations que puissent être largement accessibles sur différentes plateformes et pouvant être conçues pour différentes applications.

1.3 Historique de l'Internet des objets

Ia utilisé pour la première fois le terme Internet des objets ou en anglais (Internet of Things) en 1990. Mais l'idée réelle d'appareils connectés existe depuis plus longtemps, du mois depuis les années 70. Un chercheur technique britannique nommé Kevin Ashton a inventé le terme « Internet des objets » en 1999, mais les développeurs jouent avec l'idée d'appareils connectés à Internet depuis le début des années 1980.

La conception de contrôle à distance des appareils électriques et électroniques "objets" est connu depuis le début des années 1990, lorsque John Romkey a créé le premier appareil compatible Internet un grille- pain qu'il pouvait allumer et éteindre via Internet. Le terme "Internet des objets "a été inventé pour la première fois par Kevin Ashton lors d'une présentation en 1999 chez Procter et Gamble (P et G), liant la nouvelle idée de la RFID (Radio Frequence Identification) à la chaîne d'approvisionnement. L'Internet des objets est devenu un système qui utilise une variété de technologies, de L'Internet à la communication sans fil systèmes micro électromécaniques, jusqu'aux systèmes embarqués [15].

Depuis 2000, la connectivité Internet fait désormais partie du développement des entreprises industrielles et des produits permettant de fournir ou de récupérer des informations. Cependant, pour une plus grande précision, l'interaction humaine est devenue nécessaire dans le traitement de ces techniques. Le véritable rôle de l'Internet des objets dans le dépassement de cet obstacle a commencé à obtenir des choses en fonction des besoins humains sans interaction humaine.

1.4 Domaines d'application de l'Internet des objets

L'Internet des objets a fourni beaucoup de confort dans la vie humaine à l'époque actuelle et ses utilisations se sont étendues des utilisations personnelles aux utilisations familiales et générales ainsi que dans les affaires, les industries et de nombreux domaines. Voici quelques domaines d'application sont touchés par l'IoT :

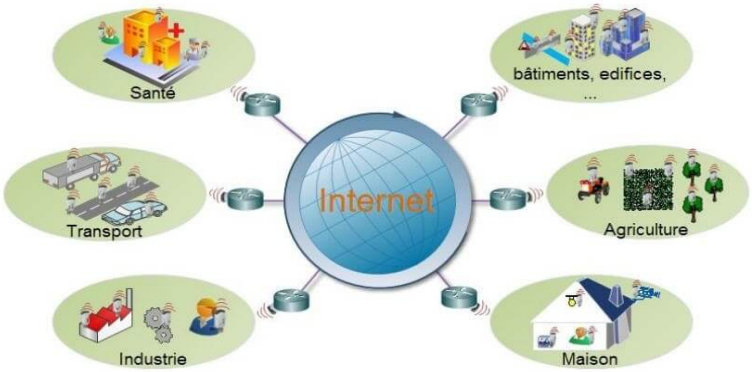

FIGURE 1.1 – domaines d'internet des objets

1.4.1 Le domaine médical

Est l'un des premiers secteurs à avoir adopté l'IdO et le domaine a connu un très grand nombre d'applications permettant à un patient et à son docteur de recevoir des

informations, parfois même en temps réels, qu'il aurait été impossible de connaître avant l'apparition d'IoT. Par exemple [15] :

— Les dispositifs médicaux portables et la surveillance à distance des patients, qui fournissent des soins de santé plus sûrs et plus efficaces grâce à la surveillance des signes vitaux des patients, au suivi après la chirurgie et au respect du traitement, les médecins peuvent surveiller l'état de santé des patients et réagir à distance en temps réel.

— Suivi des actifs médicaux, avec l'utilisation de Bluetooth Low Energy (BLE) pour la surveillance et la localisation des équipements médicaux, des médicaments et des fournitures.

— Au-delà des dispositifs portables, il existe des dispositifs médicaux implantables qui sont des systèmes très complexes, stylisés et fiables qui sont insérés dans le corps pour restaurer ou améliorer les fonctions humaines.

1.4.2 Domaine de transport

L'Internet des objets améliore la communication, le contrôle et la distribution des données. Ces applications comprennent les véhicules personnels, les véhicules utilitaires, les trains, les drones et autres équipements, surveillent les embouteillages et fournissent également des services tels que la réduction des accidents en surveillant les charges des véhicules, en vérifiant la sécurité et la proximité des marchandises avec d'autres bus, les techniques GPS et d'autres services pour commander des taxis via des appareils intelligents et traiter des questions financières sans avoir à garer le bus et à payer en espèces.

1.4.3 Maisons intelligentes

Les portes de garage connectées ou les serrures de porte numériques peuvent vous ramener à la maison en utilisant les données de votre téléphone au lieu des clés traditionnelles ;l'une des applications théoriques dont les gens discutent beaucoup est un réfrigérateur qui reconnaît les ingrédients à l'intérieur et vous enseigne les pénuries alimentaires où vous indique quel dîner peut être préparé à partir des ingrédients disponibles.

1.4.4 Domaine Agriculture

L'utilisation des nouvelles technologies, telles que l'imagerie satellitaire et l'informatique, les systèmes de positionnement par satellite de comme GPS, aussi par l'utilisation des capteurs qui vont s'occuper de récolter les informations utiles sur l'état du sol, taux d'humidité, taux des sels minéraux, etc., et envoyer ces informations au fermier pour prendre les mesures nécessaires garantissant la bonne production et les conditions des plantes, contrôler les conditions microclimatiques et surveiller les conditions météorologiques qui peuvent endommager les cultures [16].

1.4.5 Domaine industriel

Comme statuer sur le travail des machines telles que les lignes de production,l'emballage, etc. afin de réduire les travailleurs et donc de réduire les coûts pour le fabricant. La mesure

de la qualité de l'air intérieur : surveillance des niveaux de gaz toxiques et d'oxygène à l'intérieur des usines chimiques pour assurer la sécurité des travailleurs et des biens, la surveillance de la température : contrôle de la température à l'intérieur des réfrigérateurs industriels et médicaux avec des marchandises sensibles, l'autodiagnostique du véhicule : collecte d'informations sur le bus interne du véhicule afin d'envoyer des alarmes en temps réel aux urgences ou fournir des conseils aux conducteurs, et la localisation à l'intérieur : emplacement intérieur des ressources en utilisant des étiquettes actives et passives [17]. Domaine commercial : comme l'envoi, la réception de marchandises et leur suivi à distance.

1.4.6 Domaine sport

De nombreux objets connectés, comme les montres ou les bracelets connectés, permettent de compter les pas effectués dans la journée, la distance parcourue par course, votre temps actif, les calories brûlées, et la nuit en comptant votre temps de sommeil [18].

1.4.7 Villes intelligentes

Surveillance des vibrations et des conditions matérielles dans les bâtiments, les ponts et les monuments historiques.

— **La gestion des déchets :** Détection des niveaux d'ordures dans les conteneurs pour optimiser les voies de collecte.

— **Les routes intelligentes :** Autoroutes intelligentes avec messages d'avertissement et de détournements en fonction des conditions climatiques et des événements inattendus tels que les accidents ou les embouteillages.

— **Le stationnement intelligent :** Suivi de la disponibilité des places de parking dans la ville.

— l'éclairage intelligent : Eclairage des réverbères intelligent et adapté aux conditions météorologiques.

— **Les embouteillages :** Surveillance des véhicules et des piétons pour optimiser les itinéraires de conduite et de marche.

— **La cartographie urbaine du bruit :** Surveillance sonore dans les différentes zones urbaines en temps réel [17].

1.4.8 Domaine militaire

Les opérations militaires contemporaines deviennent de plus en plus complexes, multiformes et imprévisibles [19].

FIGURE 1.2 – L'Internet des objets rencontre l'armée et le champ de bataille

Par exemple :

— le problème de surveillance de la santé et la vie des soldats a été identifié et résolu par des modèles pour générer des données et visualiser les informations, sans oublier la détection des ennemis qui automatisées à l'aide de réseaux de capteurs sans fil.

— La situation critique sur le champ de bataille peut être bien gérée et traitée en introduisant des applications IoT.

— La sécurité de ces munitions est une préoccupation importante pour les forces de sécurité. Le transport de munitions dans un tel état nécessite un niveau de sécurité élevé afin d'éviter leur perte.

— Le soutien logistique dans les actions militaires est l'une des parties importantes et intégrales du système basé sur l'IoT. Ce système détecte les besoins de logistique sur le front de guerre et prend les mesures nécessaires.

1.5 Le fonctionnement de l'IdO

Pour assurer le bon fonctionnement de l'Internet des Objets (IdO), il existe plusieurs étapes à respecter. Les a résumés dans cinq étapes notamment [20] :

— Tout d'abord, chaque « objet » sur l'Internet des Objets doit avoir une identité unique. Grâce à l'évolution de l'adresse IP (Internet Protocol) et ses générations on peut fournir des milliers de différentes adresses IP au point que nous nous devrions pouvoir attribuer un identifiant unique à chaque objet physique de la planète.

— Deuxièmement, les « objets » doivent communiquer entre eux. Grâce aux nouvelles technologies sans fil qui existent et qui rendent les communications possibles, telles que le Wifi, LoRaWAN, les communications à faible champ (Bluetooth), la communication en champ proche (NFC), la RFID, ainsi que les technologies ZigBee, Z-Wave.

— Troisièmement, afin d'obtenir des informations sur l'environnement, chaque « objet » doit avoir des capteurs. Il existe de nombreux capteurs notamment les capteurs de température, d'humidité, de lumière, de mouvement, de pression, infrarouge, à ultrasons, etc. Les nouveaux capteurs deviennent de plus en plus petits, économiques et durables.

— Quatrièmement, grâce au microcontrôleur que les « objets » doivent avoir, on peut gérer les capteurs, les communications ainsi que l'exécution des tâches. Il existe de nombreux microcontrôleurs pouvant être utilisés dans IdO selon le besoin.

— Enfin, afin de pouvoir exploiter la puissance de calcul, le stockage d'un serveur informatique, ainsi l'analyse et l'affichage des données, les services cloud (le cloud (nuage en français) désigne un endroit où sont stockées des ressources informatiques auxquelles on peut accéder à distance via un réseau de communication (bien souvent Internet).En clair, au lieu d'utiliser son ordinateur personnel pour lancer une application ou stocker ses données, on se connecte à des serveurs qui font eux-mêmes le travail : c'est le cloud computing) sont recommandés afin de pouvoir voir ce qui se passe et d'agir via des applications téléphoniques. Beaucoup de grandes entreprises y travaillent déjà, telles que Watson, IBM, la plate-forme Google Cloud, Azure, Microsoft, Oracle Cloud, etc.

1.6 Caractéristiques de l'Internet des Objets

L'Internet des objets possède de nombreuses fonctionnalités, dont certaines proviennent de [19] :

1.6.1 Interconnexion

La liaison ou la connectivité est l'aspect le plus important de l'Internet des objets. L'écosystème de l'Internet des objets (capteurs, moteurs de calcul, centres de données, etc.) ne peut pas fonctionner correctement sans une connexion transparente entre des composants ou des objets interconnectés. Il existe de nombreuses façons de connecter des appareils IoT, notamment les ondes radio, Bluetooth, Wi-Fi.
Hétérogénéité : Les appareils composants IoT sont hétérogènes, car ils reposent sur des plates-formes et des réseaux matériels différents, peuvent être des capteurs sans fil, des appareils cellulaires, etc.

1.6.2 Taille

Le nombre d'appareils à gérer et à communiquer entre eux sera plus grand que ceux actuellement connectés à Internet. Montre ici l'importance de la gestion, du traitement et de l'utilisation des données échangées entre ces appareils par application.

1.6.3 Évolutivité

Chaque jour, de plus en plus d'éléments se connectent à la zone IoT. Par conséquent, les appareils de l'Internet des objets devraient être en mesure de faire face à l'expansion massive. Les données ainsi créées sont énormes et doivent être traitées correctement.

1.6.4 Les services reliés aux objets

L'IoT est capable de fournir des services qui sont liés aux objets tout en tenant compte des contraintes telles que la protection de la vie privée ainsi que la cohérence sémantique

entre les objets physiques et leur(s) objet(s) virtuel(s) associé(s).

1.6.5 Les changements dynamiques

L'état des dispositifs (par exemple, connecté/déconnecté) change de manière dynamique tout comme le contexte dans lequel ces dispositifs fonctionnent qu'il soit relié au cadre spatio-temporel ou également dans le cadre de la vitesse ou encore de la localisation. Il est important de noter que leur nombre est également susceptible d'évoluer lui aussi.

1.6.6 Une très grande échelle

Dans le futur, l'ensemble des dispositifs sera au moins dix fois plus nombreux qu'à l'heure actuelle comme le précise l'Union Internationale des télécommunications. Ils devront donc être à la fois gérés et être capables de communiquer entre eux. Le rapport qui existe entre les connexions établies par des dispositifs ainsi que celles établies par des personnes deviendra beaucoup plus favorable aux premières. C'est une des raisons pour laquelle la gestion ainsi que l'interprétation des données générées seront critiques. Compte tenu des milliards de zettaoctets de données qui feront partie de notre futur, l'extensibilité est une caractéristique évidente et une exigence de l'IoT, car certaines solutions issues de cette technologie ne révèleront leur potentiel qu'à l'échelle à partir de laquelle elles deviendront exploitables.

1.7 Architecture IoT

L'architecture d'un système IoT est composée de plusieurs niveaux qui communiquent entre eux pour relier le monde tangible des objets au monde virtuel des réseaux et du cloud. Même s'il n'existe pas d'architecture IoT unique universellement acceptée, le format le plus basique et le plus largement accepté est une architecture IoT à trois couches : la couche de perception, la couche réseau et la couche application. Ces trois couches contiennent toutes une grande quantité d'informations avec différentes technologies et caractéristiques, comme le montre la figure 1.3.

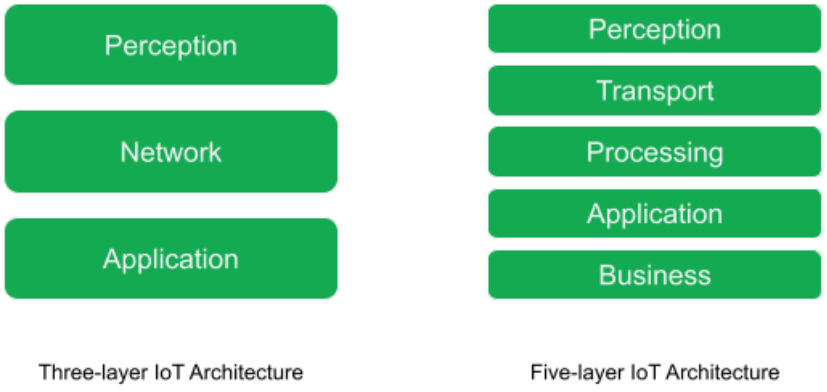

FIGURE 1.3 – Les couches de l'IoT

— **Couche de perception :** Est la couche physique, qui comporte des capteurs pour détecter et recueillir des informations sur l'environnement. L'objectif de cette couche est de définir la signification physique de chaque élément du système IoT, comme les emplacements, elle recueille également les informations sur chaque objet du système et transforme ces données en signaux, elle contient les technologies utilisées dans l'IoT telles que les cartes à puce, les étiquettes RFID (caméras, and Radio Frequency Identification Devices), les lecteurs et les réseaux de capteurs, etc. Le système RFID permet d'obtenir des informations sur les objets à tout moment et en tout lieu. Chaque étiquette électronique RFID possède un identifiant unique appelé code produit électronique (EPC Electronic Product Code), qui est le seul identifiant consultable attribué à chaque cible physique. Des informations supplémentaires sur le produit sont fournies par une chaîne de chiffres qui lui est imposée, comme le fabricant et la catégorie du produit, avec sa date de fabrication et sa date d'expiration, etc. [21].

— **Couche réseau :** Cette couche fournit un soutien réseau de base et le transfert de données par un réseau sans ou avec fil. Ses fonctions sont également utilisées pour transmettre et traiter les données des capteurs. Le domaine de réseau comprend différents réseaux d'accès, qui fournissent une connectivité à travers diverses technologies, telles que xSDL, Satellite, etc. vers des dispositifs et/ou des passerelles. Ils fournissent également une connectivité au réseau central qui inclut une connectivité hétérogène et multi technologies, telle que 3GPP, TISPAN et LTE-A [22]. Les communications entre les appareils et les services cloud ou les passerelles impliquent différentes technologies : Ethernet, Réseaux cellulaires, LPWAN (Low-power Wide-area Network) et Wi-Fi, le travail de la couche réseau, elle connecte donc ces appareils à d'autres objets intelligents, serveurs et appareils réseau.

— **La couche application :** Est responsable de la fourniture de services spécifiques à l'utilisateur. L'objectif de cette couche est de déterminer les types d'applications qui seront utilisées dans l'IoT. Elle développe également les applications IoT pour qu'elles soient plus intelligentes, authentifiées et sûres. Elle définit diverses applications dans lesquelles l'Interner des objets peut être déployé, par exemple les maisons intelligentes, les villes intelligentes et la santé intelligente. Inclut les applications IoT et les infrastructures serveur / cloud. Ces derniers doivent partager leurs contenus, éventuellement les sauvegarder sur d'autres appareils, programmes d'analyse et / ou personnes qui ont besoin de surveiller la réponse en temps réel. Ils incluent également des fonctionnalités de service, qui fournissent des fonctions partagées entre différentes applications via des abstractions de haut niveau ouvertes et des interfaces qui masquent les spécificités des réseaux sous-jacents [22]. Et c'est dans cette couche que dans cette couche toutes les décisions de contrôle, de sécurité et de gestion des applications sont prises [23].

L'architecture à trois couches définit l'idée principale de L'Internet des objets, mais elle n'est pas suffisante pour la recherche sur l'IoT, car celle-ci se concentre souvent sur des aspects plus fins de L'Internet des objets. C'est pourquoi de nombreuses autres architectures en couches sont proposées une l'architecture à cinq couches, qui comprend en plus les couches de traitement et d'affaires. Les cinq couches sont la perception, le transport, le traitement, l'application et métier. Le rôle des couches perception et application est le

même que dans l'architecture à trois couches. Nous décrivons la fonction des trois autres couches.

— **Couche métier :** L'objectif de cette couche est de définir la charge et la gestion des applications IoT. Elle est également responsable de la confidentialité de l'utilisateur et de toutes les recherches liées aux applications IoT. Le succès de tout appareil ne dépend pas seulement des technologies qui y sont utilisées, mais aussi de la manière dont il est livré à ses consommateurs. La couche de gestion effectue ces tâches pour l'appareil. Cela implique de créer des organigrammes, des graphiques, une analyse des résultats, et comment l'appareil peut être amélioré, etc... [24].

— **Couche traitement :** Sa responsabilité consiste à traiter les informations recueillies par la couche de perception. Le processus de traitement comporte deux aspects principaux : le stockage et l'analyse. L'objectif de cette couche est extrêmement difficile à atteindre en raison de l'énorme quantité d'informations recueillies sur les éléments du système. Elle utilise donc des techniques telles que les logiciels de base de données, l'informatique en nuage, l'informatique omniprésente et le traitement intelligent pour traiter et stocker les informations [24].

— **Couche transport :** Elle ressemble à la couche réseau dans l'architecture à trois couches. Elle transmet et reçoit les informations de la couche de perception à la couche de traitement et vice-versa. Elle contient de nombreuses technologies telles que sans fil, 3G, LAN, l'infrarouge, le Wi-Fi et le Bluetooth. De plus, l'objectif de cette couche est d'adresser chaque chose dans le système en utilisant l'IPV6 [24].

1.8 Défis liés à l'Internet des Objets

Comme toute technologie moderne avec son apparition et sa propagation, les craintes qui y sont associées viennent avec elle et ces craintes sont les plus grandes préoccupations de ses utilisateurs et la principale source de préoccupation est la sécurité de l'information et la confidentialité des données des bénéficiaires et les préoccupations et les défis auxquels l'Internet des objets est confronté peuvent être résumés aux points suivants [24] :

— Bien que l'utilisation de l'Internet des objets soit répandue et étendue, certaines vulnérabilités de sécurité peuvent conduire à pirater des appareils et à obtenir des informations sur les bénéficiaires.

— Les services de santé basés sur Internet sont affectés en cas d'erreurs involontaires.

— Manque de normes Internet pour les objets dans la collecte, la préservation et le transfert de données.

— L'accès à un appareil de communication bon marché est l'un des défis les plus importants auxquels sont confrontés les fabricants de technologies aujourd'hui, car la propagation attendue de l'Internet des objets sera fortement conditionnée par la capacité des fabricants à produire un dispositif de communication peu coûteux capable d'interagir avec l'environnement environnant grâce à des capteurs et des règles, en plus de tout dommage aux systèmes intelligents peut nécessiter des coûts matériels élevés afin de pouvoir les redémarrer à nouveau.

— Il y a toujours un débat sur la nécessité d'une norme complète et unifiée pour L'Internet des objets, certains la rejettent et d'autres pensent qu'elle est inévitable, tandis que les centristes suggèrent qu'il existe plusieurs normes flexibles qui ne sont pas strictement limitées dans cette technologie.

— Écologisation de l'IoT : La consommation d'énergie du réseau augmente à un rythme très élevé en raison de l'augmentation des débits de données, de l'augmentation du nombre de services connectés à Internet et de la croissance rapide des périphériques connectés à Internet. Le futur IoT entraînera une augmentation significative de la consommation d'énergie du réseau. Ainsi, des technologies vertes doivent être adoptées pour rendre les dispositifs de réseau aussi économes en énergie que possible.

1.9 Modélisation des applications IoT

De nombreux travaux de recherche ont été réalisés en utilisant l'approche basée sur les modèles pour développer et analyser les applications IoT.

— **SysML4IoT :** Costa et al. [25] ont présenté une méthode basée sur l'IDM pour concevoir et analyser les applications IoT. Les auteurs ont proposé le langage de modélisation SysML4IoT pour les systèmes IoT. SysML4IoT est un profil SysML basé sur le modèle de référence IoT-A. Les auteurs ont ensuite défini un mappage de SysML4IoT à SysML4IoT. Ils ont ensuite défini un mappage des modèles SysML4IoT dans le langage d'entrée du vérificateur de modèles NuSMV pour vérifier les propriétés de qualité de service du système développé.

— **UML4IoT :** Dans [26], les auteurs ont proposé UML4IoT comme profil UML pour représenter les constructions de base du protocole LWM2M et le paradigme architectural REST pour les environnements de fabrication. L'approche UML4IoT automatise la génération de l'interface conforme à l'IoT nécessaire pour intégrer les composants cyber-physiques dans les systèmes de fabrication IoT modernes.

— **MDE4IoT :** Ciccozzi et al. [27] ont introduit MDE4IoT comme un cadre IDM pour soutenir la modélisation et l'auto adaptation des configurations émergentes des systèmes IoT. Les auteurs visent à relever de nombreux défis tels que l'hétérogénéité et la complexité des systèmes à l'aide d'une abstraction de haut niveau, le développement collaboratif par la séparation des préoccupations et les adaptations à l'exécution qui sont censées être réalisées automatiquement par des transformations de modèles. Le cadre MDE4IoT est constitué de profils spécifiques au domaine basés sur UML. Il définit l'aspect comportemental comportemental des composants logiciels en utilisant des machines d'état et le langage d'action pour UML fondamental (ALF).

— **ThingML :** ThingML [7] est une approche basée sur les techniques de l'IDM pour développer des applications IoT. Elle comprend un langage textuel spécifique au domaine (DSL) et un cadre de génération de code. Dans la littérature, plusieurs travaux de recherche se sont intéressés à l'utilisation de ThingML comme langage de modélisation ou comme cadre de génération de code multiplateforme. Xu et al. [28] ont proposé d'étendre ThingML pour permettre aux concepteurs de modéliser les variations de performance affectées par des environnements externes incertains.

Les modèles obtenus sont transformés en un réseau d'automates temporisés tarifés (NPTA) dans le but de réaliser une analyse quantitative de la qualité de service des applications IoT en utilisant UPPAAL-SMC. Dans [29], les auteurs ont proposé une approche pour transformer les modèles CAPS dans le langage ThingML. Elle vise à générer du code à partir des spécifications CAPS, où CAPS est un cadre de modélisation orienté architecture pour le développement de systèmes IoT. CyprIoT [30] est un cadre permettant de modéliser et de contrôler les systèmes IoT basés sur un réseau. Il utilise une partie de ThingML pour modéliser les objets IoT et leurs comportements. En outre, il utilise et étend le cadre de génération de code thingML. Ihirwe et al. [31] ont présenté l'approche de modélisation basée sur les composants CHESSIoT pour soutenir la conception et l'analyse des systèmes IoT industriels. CHESSIoT fournit un moyen d'effectuer une modélisation basée sur les événements à des fins de génération de code. Les spécifications CHESSIoT seront transformées en modèles ThingML afin de tirer parti des générateurs de code de ThingML.

1.10 Conclusion

Dans ce chapitre on a présenté quelques notions sur l'Internet des objets, tel que la définition l'architecture, les couches, ses applications dans les domaines civil et militaire, ensuite nous avons entamé sa modélisation avec les notations principales.

CHAPITRE 2

L'APPROCHE THINGML

2.1 Introduction

ThingML est une approche basée sur les techniques de l'IDM pour développer des applications IoT. Elle a été promue comme une initiative open source pour résoudre le problème de l'hétérogénéité dans l'Internet des objets. ThingML comprend un langage textuel spécifique au domaine (DSL) et un cadre de génération de code. Le DSL ThingML est un profil UML permettant de concevoir des applications IoT indépendantes de la plateforme. Le cadre de génération de code comprend une collection de compilateurs permettant de transformer les modèles ThingML vers divers langages cibles.

Dans ce chapitre, nous présentons dans un premier temps, la définition et l'historique de ThingML. Ensuite, nous présentons de manière détaillée les concepts de ThingML. Enfin, à la section 3.7, nous présentons un exemple de spécification ThingML.

2.2 Définition

L'approche ThingML comprend un langage de modélisation, un ensemble d'outils et une méthodologie. ThingML combine des constructions de modélisation logicielle éprouvées alignées avec UML (statecharts et composants), un langage impératif et des constructions ciblées sur les applications IoT. Les outils comprennent des éditeurs, des transformations (par exemple, l'exportation vers UML), et un cadre de génération de code multi-plateforme avancé qui prend en charge plusieurs langages de programmation cibles. ThingML offre plus de personnalisation de son code, en donnant aux développeurs l'abstraction qu'ils doivent améliorer la productivité pour réduire l'idée d'une « boîte noire », qui associée aux générateurs de code (c'est Une boite mystérieuse, qui va nous permettre d'apprendre les bases de la programmation de générateur de code) pour une meilleure compréhension dans le processus de génération de code [32].

On peut définir aussi ThingML (Internet of Things Modeling Language) comme un outil de modélisation open source pour les applications IoT. Actuellement, ThingML propose un langage de modélisation et un support d'outils pour modéliser les composants du système, leurs interfaces de communication ainsi que leurs comportements. Ce der-

nier se fait à travers des machines d'état. L'objectif de ThingML est d'apporter l'IDM aux phases tardives de conception et de mise en œuvre du cycle de vie du logiciel ainsi que pour soutenir la maintenance et les tâches d'évolution. Le but de ThingML est de permettre la modélisation des composants logiciels et de générer automatiquement des modules de code complets prêts à être déployés [33].

2.3 Historique de ThingML

En termes de domaines d'application, les premiers cas d'utilisation de ThingML étaient d'appliquer l'IDM dans le développement des systèmes embarqués relie avec des partenaires industriels (par exemple des réseaux de capteurs dans le domaine du pétrole et du gaz), l'équipe de développement pendant la modélisation ont évité tous les outils UML et insiste généralement sur les machines d'états pour concevoir, discuter et documenter la fonctionnalité de haut niveau des différents composants de leurs systèmes. La modélisation basée en général sur une notation textuelle pour formaliser systématique les états machines. Plus tard, les modèles seraient généralement transformés en code dans une manière systématique mais entièrement manuelle, même si un certain nombre des outils open source existaient également pour générer code des machines d'état [33].

2.3.1 Version 01

La première version de ThingML a été proposée sur la base d'outils de machines à états non-UML dédiés au développement de systèmes embarqués. L'approche de ThingML a été construite comme une architecture à trois couches : une couche domaine, une couche application et une couche matérielle et pilote [33].

— *La couche de domaine* spécifie un ensemble de composants de domaine (matériel / logiciel) ainsi que leurs interfaces en termes de messages asynchrones. Un DSL textuel simple a permis de définir le modèle de domaine et un générateur de code a été utilisé pour générer systématiquement des API du modèle de domaine et du code de communication [33].

— *La couche d'application* a été entièrement modélisée à l'aide d'outils de machine d'état commerciaux (IAR VisualSTATE) et ne pouvait que manipuler les composants et les messages définis dans le modèle de domaine. Le générateur de code IAR VisualSTATE a été utilisé pour générer un modèle complet mettant en œuvre l'application qui pourrait être liée à l'API du modèle de domaine. Dans la machine d'état, les variables et les actions étaient écrites directement en C [33].

— *La couche matérielle et pilote* consistait en d'implémenter manuellement les API définies dans le domaine maquette. L'approche garantissait que chaque composante était mise en œuvre dans un module distinct.

L'avantage de cette première version était un bon découplage entre les développeurs des différents pilotes et les développeurs d'une application. L'utilisation d'un outil de modélisation commercial dédié aux machines d'état et aux systèmes embarqués éprouvé bien plus pratique que la confection d'un outil UML générique. Le DSL textuel utilisé pour la définition du domaine modèle s'est également avéré facile à utiliser par les développeurs [33].

2.3.2 Deuxième version :

La deuxième version de ThingML a été conçue comme un langage de modélisation textuel spécifique à un seul domaine. Il conserve les principes de l'approche précédente : la définition obligatoire d'un composant avec des interfaces (les fragments) de messages asynchrones explicites et la possibilité de définir les composants indépendants et les composants spécifiques de la plate-forme. Le modèle de composant précédemment défini est étendu avec un modèle de machine d'état pour spécifier le comportement des composants et un langage d'action définit les gardes indépendants de la plate-forme et les actions dans l'état machines. Pour le code spécifique à la plate-forme, un mécanisme de kick-down permet d'intégrer le code dans le modèle. Pour terminer, un modèle de configuration permet également d'assembler des composants afin de générer le code.

Cette version de ThingML a été appliquée sur un ensemble de cas d'utilisation et un ensemble de générateurs de code ont été développés et spécialisés pour s'adapter à différentes plates-formes cibles et langages de programmation.

L'expérience a montré que la notation textuelle pour la machine d'état est bénéfique pour l'édition et la mise en œuvre du comportement des composants. Elle facilite l'utilisation pour les développeurs, géré de grandes machines d'état, supporte la charge de stockage et la gestion des versions dans n'importe quel référentiel de code, etc. Le seul inconvénient est qu'elle ne permet pas d'avoir facilement une vue d'ensemble des machines d'état. Pour fournir cette vue, il doit exporter la représentation graphique UML de machines d'état ThingML (une transformation graphique) [33].

2.3.3 Troisième version

La troisième version de ThingML préserve le même langage avec un changement complet du générateur de code qu'il est en fait une famille de compilateurs1. Les sections suivantes détaillent la conception de cette version de ThingML [33].

2.4 Le langage spécifique au domaine ThingML

2.4.1 Grammaire de ThingML

ThingML est un profil UML conçu pour modéliser les applications IoT en tant que langage textuel de modélisation spécifique au domaine (DMSL) [6,7]. Sa syntaxe textuelle est formalisée sous forme de grammaire dans le cadre Xtext [34] basé sur le cadre de modélisation Eclipse (EMF) [35], qui permet le développement de DSL textuels. Xtext utilise un langage de type EBNF (Extended Backus-Naur Form) pour définir la grammaire du DSL afin de générer automatiquement un éditeur textuel complet et un méta-modèle Ecore du langage développé. La figure 3.1 représente un extrait de la grammaire ThingML exprimée dans le langage EBNF. Cet extrait montre une partie de la syntaxe textuelle concrète de la machine à états qui consiste en des états. Un état peut être un état simple, un état final ou un état composite. Il peut inclure des propriétés et des transitions (internes), et effectuer des actions à l'entrée et à la sortie. Une transition doit avoir une cible, et elle peut avoir un événement déclencheur, une condition de garde et des actions à effectuer pendant la transition.

```
/*********************************************************************
*       STATE MECHINES                                               *
*********************************************************************/
State returns State:
       StateMachine | FinalState | CompositeState |
       'state' name=ID ( annotations+=PlatformAnnotation )* '{'
              (properties+=Property)*
              (
                 ('on' 'entry' entry=Action)? &
                 ('on' 'exit' exit=Action)? &
                 (properties+=Property | internal+=InternalTransition |
                  outgoing+=Transition)*
              )
       '}';
Handler:
       Transition | InternalTransition
;
Transition returns Transition:
       'transition' (name=ID)? '->' target=[State|ID]
       ( annotations+=PlatformAnnotation )*
       ('event' event=Event)?
       ('guard' guard=Expression)?
       ('action' action=Action)?;

InternalTransition returns InternalTransition:
       {InternalTransition}
       'internal' (name=ID)?
       ( annotations+=PlatformAnnotation )*
       ('event' event=Event)?
       ('guard' guard=Expression)?
       ('action' action=Action)?;
```

FIGURE 2.1 – Un extrait de la grammaire ThingML exprimée en EBNF [36]

2.4.2 Méta-Modèle de ThingML

Comme mentionné ci-dessus, Xtext [34] utilise un langage de type EBNF pour définir la grammaire DSL afin de générer automatiquement un méta-modèle basé sur Ecore. Le méta-modèle ThingML comprend quatre-vingt-huit classes. La figure 2.2 représente un extrait du méta-modèle ThingML. Elle montre les principaux concepts du modèle ThingML et les relations entre ces concepts. En résumé, un modèle ThingML, décrit par la classe ThingMLModel dans ce méta-modèle, peut être exprimé comme une combinaison de things et de configurations. La classe Thing comprend (relation de composition) plusieurs composants tels que des messages, des machines à états et des ports. Chacun de ces composants est décrit par une classe dans le méta-modèle [37].

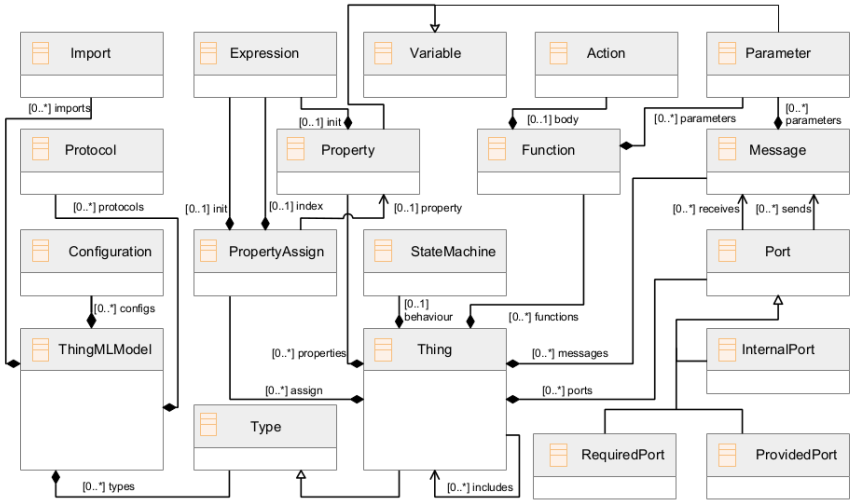

FIGURE 2.2 – Un extrait du méta-modèle ThingML [37]

La figure 2.3 montre la partie machine à états dans le méta-modèle ThingML. La classe State présente la classe principale ; elle contient une action à l'entrée, une action à la sortie, des propriétés et des transitions. Les états peuvent également contenir des états composites qui peuvent être séquentiels ou simultanés. Une transition a un état cible et peut contenir un événement, une condition de garde et une action. Un événement est un message qui arrive via un port [37].

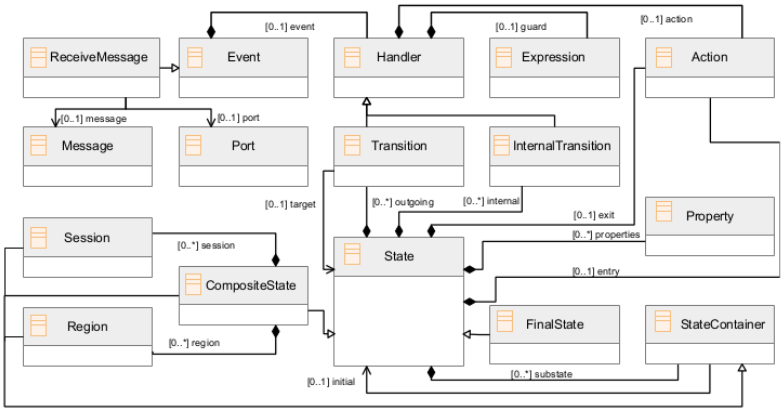

FIGURE 2.3 – Un extrait du méta-modèle ThingML (la partie de machine d'état) [37]

La figure 2.4 présente la partie configuration dans le méta-modèle ThingML. Une configuration peut inclure des instances et des connecteurs. Un connecteur possède une instance client (définie par la relation cli) et une instance serveur (définie par la relation srv). Il possède également un port fourni (défini par la relation provided) et un port requis (défini par la relation required). La relation type relie les deux classes, Instance et Thing [37].

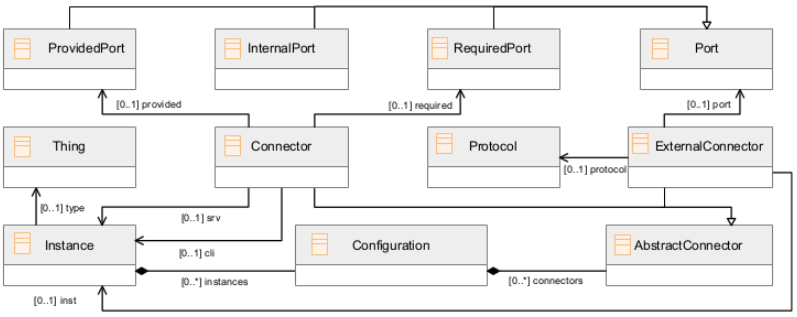

FIGURE 2.4 – Un extrait du méta-modèle ThingML (la partie de la configuration) [37]

2.5 Concepts de ThingML

Le DSL ThingML s'appuie principalement sur deux structures [7] : Thing et Configuration, où le Thing représente les composants de l'application, et la Configuration décrit l'instanciation de ces composants et leur interconnexion. En outre, les comportements des composants sont modélisés à l'aide de Statecharts et d'un langage d'action impératif indépendant de la plate-forme.

2.5.1 Thing

Thing est un concept important de ThingML, représenté un composant logiciel qui peut importer une instance pour créer des configurations concrètes. Un thing peut comprend des messages, des ports, des fonctions, des propriétés, des fragments et les machines d'état. On utilise le fragment lorsqu'on définit un thing inclus dans d'autres things. Un thing fragmenté ne peut pas être instancié.

Pour que les thing communiquent entre eux il y a des messages asynchrones transportés via des ports sont responsables de ça, ils contiennent des paramètres de n'importe quel type de données pris en charge par le langage ThingML. Pour terminer, les propriétés définissent des variables locales qui ne sont accessibles que par machines d'état et fonctions. La déclarations de Things sera comme ce qu'est montré dans le code suivant :

```
thing thing_name1 includes thing_frag_name
{
    \\ La declaration des messages, des ports, des fonctions, des proprietes,
       statecharts ...
}
```

Fragments : Nous avons dit avant qu'on utilise le fragment lorsqu'on définit un thing inclus dans d'autres things ce dernier ne peut pas être instancié. Pour clarifier plus, si un thing a un ou plusieurs ports qu'elle fournit, ces ports peuvent être définis comme un thing fragmenté, ou définie dans le thing il-même. Le nom doit se terminer par "Msgs" pour signaler qu'il s'agit de l'interface. Le fragment est créé par les mots-clés "thing fragment". À l'intérieur du fragment, tous les messages que le port est censé envoyer poignée doit être définie [38].

```
thing fragment thing_frag_name_Msgs
```

```
{
   \\ declarations des message et des port
}
```

2.5.2 Messages

Les messages sont la moyenne principale de communication entre les things. Ils sont envoyés dans les deux sens via les ports peuvent être déclarés dans un fragment, ou directement dans un thing, afin d'être utilisé par les ports. La déclaration est le mot clé message suivi du nom du message. Les messages peuvent aussi contenir des paramètres de n'importe quel type de données pris en charge par le langage ThingML [38].

```
message Msg_name1();
message Msg_name2(par : DataType);
```

2.5.3 Ports

La communication entre les things dans ThingML passe par l'utilisation de ports. Un port peut à la fois envoyer et recevoir des messages vers et depuis d'autres things. Un thing peut fournir un port ; ce port devient alors disponible pour d'autres things à utiliser. Les ports peuvent également être requis, cela signifie qu'un objet peut indiquer qu'il utilise un autre port des objets. La communication via les ports se fait avec passage des messages asynchrones qui sont présentés dans la section suivante. Les messages envoyés via le port sont le principal moyen de déclencher transitions, événements internes et faire en sorte que la machine d'état change d'état et faire avancer le programme. Il y a trois types de ports [38] :

— **Internal port :** Un thing peut utiliser un pour envoyer et receviez même message.

```
internal port Port_name2
{
    sends Msg_name2
    receives Msg_name1
}
```

— **Required port :** Un port peut également être requis, cela signifie qu'un objet peut utiliser un autre port des objets. Dans le code suivant, un exemple de Required port est présenté :

```
required port Port_name1
{
    sends Msg_name1
    receives Msg_name2
}
```

— **Provided port :** Un objet peut fournir un port ; ce port devient disponible pour d'autres objets à utiliser. Un exemple de Provided port est présenté le code suivant :

```
provided port Port_name2
{
    sends Msg_name2
    receives Msg_name1
}
```

2.5.4 Properties

Les propriétés définissent des variables locales qui ne sont accessibles que par leur thing , machines d'état et fonctions.

```
property prop_name : DataType = 0
```

2.5.5 Machine d'état

Dans ThingML, la machine à état est la partie principale des things, en fait appelée, "diagramme d'état" et un état initial est requis par le diagramme d'état ThingML comme point d'entrée, effectue tout le calcul et le travail. Leur structure peut comprendre un ou plusieurs états, des transitions, des régions parallèles, états composites et états finaux. La machine à états réagit en fonction des événements correspondants trois manières : à l'entrée des états, à la sortie des états ou pendant les messages qui doivent être reçus et mis en correspondance avec une transition. Les transitions sont le seul moyen de déplacer la machine d'état d'un état à l'autre. Ils se déclenchent lorsqu'un message arrive via les ports et les valeurs des propriétés locales et leurs conditions de garde sont évaluées comme vraies.) [38].

Les états :

Les états sont les principaux blocs de dans la construction du diagramme d'état les états des blocs principaux et le langage de ThingML fournit trois types d'états différents qui peuvent être utilisés (états d'entrée, états de sortie et états avec transition). Ils ont tous des différences domaines d'expertise et s'ils sont combinés, ils peuvent créer un état assez complexe machine [38].

```
statechart thing_name1_behav init State_name1
{
    state State_name1 {
        on entry ... Actions_Entry ...
        on exit ... Actions_Exit ...
        transition -> CompositeState_name
            event Msg_name2 ? Port_name1
            guard ... Cond ...
            action ... Actions-trans ...
}
```

États composites :

Les états composites sont une construction où un état contenir plusieurs états et ainsi fonctionner comme une machine à états. L'état composite passe dans un état initial interne, lorsqu'il est entré, et traversera ensuite ses états jusqu'à ce qu'un message soit reçu, dans ce moment il permet de sortir et de revenir à un état dans la machine d'état principale. L'état composite est défini avec le mot clé composite devant le mot-clé state, et il doit contenir au moins un état [38]. Un exemple d'état composite est présenté dans le code suivant :

```
composite state CompositeState_name init State_name2
{
    state State_name2 { ... }
    ...
}
```

Régions :
Une région est une construction qui apparaît au même niveau que les états. Elle peut contenir plusieurs états, tout comme la construction de diagramme d'état ou l'état composite. La région commencera à gérer sa propre machine d'état dans parallèle avec la carte d'état environnante. Les états à l'intérieur d'une région ne peuvent pas transiter aux états extérieurs à la région, mais ils peuvent déclencher des messages pouvant être reçus en dehors de la région. La région est définie par le mot clé région suivi d'un nom puis le mot clé init et le nom d'un état défini à l'intérieur la région [38].

```
region Region_name init State_name3
{
    state State_name3 { ... }
    ...
}
```

Transitions :
Les transitions sont le moyen principal pour transiter l'état d'une machine d'un état à un autre. Les transitions tracent un chemin d'un état à un autre et doivent être définies à l'intérieur de l'état à partir duquel la transition doit être déclenchée. Lorsqu'une transition est déclenchée, le bloc états "à la sortie" sera exécuté s'il est défini et ensuite l'état passera de lui-même à l'état cible. Si la transition a une action définie, cette action sera exécutée après la sortie de l'état et avant le potentiel sur bloc d'entrée de l'état cible. Une transition est définie par le mot-clé "transition" suivi d'une option nom, puis une flèche suivie du nom de l'état cible.
La transition doit s'inscrire pour un message spécifique à écouter sur un port, et la transition ne peut pas être déclenchée pour s'exécuter avant que le message ne soit reçu de ce port. L'écouteur d'événement est déclaré par le mot-clé event suivi du nom du port, d'un point d'exclamation et du nom du message [38].

```
transition -> CompositeState_name
    event Msg_name2 ? Port_name1
    guard ... Cond ...
    action ... Actions-trans ...
```

2.5.6 Configuration

La configuration permet de décrire les applications concrètes. Elle est un ensemble d'instances d'objets définies, et un ensemble de connecteurs entre ces instances [38].

- **Instance :** L'instance hérite de toutes les caractéristiques de l'objet parent telles que les messages, les ports, les propriétés et le comportement [38].
- **Connecteur :** Le connecteur représente un lien entre deux ports des instances. Par défaut, il est un connecteur local, ce qui permet uniquement de gérer la communication entre les instances déployées sur le même nœud. Dans une configuration réaliste, la logique doit être répartie sur plusieurs nœuds et la communication entre les nœuds sera forcément nécessaire donc il doit connecter un port avec l'extérieur de la configuration [38].

```
configuration Config_name
{
    instance Instance_name1 : thing_name1
```

```
    instance Instance_name2 : thing_name2
    connector Instance_name1.Port_name1 => Instance_name2.Port_name2
}
```

2.5.7 Langage d'action indépendant de la plate-forme

ThingML décrit les expressions arithmétiques et booléennes, l'envoi et la réception de messages, la déclaration de variables ou des fonctions locales et l'appel de fonctions à l'aide d'un langage d'action indépendant de la plate-forme. Notez que le langage d'action prend en charge la programmation structurée par des actions conditionnelles if (-else), et par des actions itératives while et for (boucles). Le tableau 1 présente un extrait simplifié de la syntaxe de ce langage d'action décrit dans la forme Backus-Naur (BNF) [37].

TABLE 2.1 – La syntaxe du langage d'action indépendant de la plate-forme en BNF [37]

Exp : :=	Byte | Char | String | Var | AExp | BExp
AExp : :=	Int | Float | AVar | - AExp | AExp bin_op AExp
BExp : :=	Bool | BVar | BExp bool_op BExp | **not** BExp | AExp rel_op AExp
bin_op : :=	**+** | **-** | ***** | **/** | **%**
rel_op : :=	**==** | **!=** | <= | < | >= | >
bool_op : :=	**and** | **or**
Para : :=	ID **:** DataType
Action : :=	**do** Action **end** | Action Action | **var** Para = Exp | Var = Exp | AVar = AExp | BVar = BExp | ID **!** ID [(parameters)] | AVar **++** | AVar **--** | **while** (BExp) Action | **for** (Para [**,** Para] **in** Array) Action | **if** (BExp) **then** Action [**else** Action] | **return** Exp | **print** Exp | **println** Exp | **error** Exp | **errorln** Exp | FunctionCallStatement

2.5.8 Langages cibles

Le langage ThingML fournit également un ensemble d'annotations qui permettent aux concepteurs d'utiliser les langages cibles et de bénéficier des bibliothèques existantes.

2.6 Générateur de code ThingML

Pour permettre un développement efficace et rapide des applications, ThingML fournit un cadre de génération de code capable de produire un code entièrement opérationnel pour de multiples plateformes. Le tableau 2.2 présente les plateformes prises en charge par le cadre de génération de code ThingML qui a été utilisé dans plusieurs projets commerciaux et industriels [7]. La structure de ce cadre le rend hautement personnalisable, ce qui permet au développeur de personnaliser efficacement et facilement certaines parties du processus de génération de code en fonction des particularités des applications développées [41].

Table 2.2 – Platforms supported by the code generation framework

Platform	Memory	Type	Target language
Avr 8bits[2]	2-8 KB	Micro Controller	C/C++
TI MSP430	8 KB	Micro Controller	C/C++
ARM Cortex PSoC 4	32KB	Embedded Processor	C/C++
Espruino (ARM)	48KB	Embedded Processor	javascript(JS)
MIPS(Atheros AR9331)	64 MB	Embedded Processor	C/C++,JS,java
Raspberry Pi	0.5-1GB	Embedded Processor	C/C++,JS,java
Intel Edison	1 GB	Embedded Processor	C/C++,JS,java
x86	GBs	Processor	C/C++,JS,java
Linux/Windows	GBs	Cloud	C/C++,JS,java

2.7 Exemple de spécification ThingML

Dans cette section, nous considérons une conception ThingML extraite du projet de recherche HEADS (Heterogeneous and Distributed Services for the Future Computing Continuum) [39]. Le modèle PingPong [40] présente un programme ThingML entièrement indépendant de la plate-forme (il utilise uniquement des instructions ThingML). Il montre les principales constructions du langage ThingML. Cette conception montre comment utiliser deux composants pour échanger des messages asynchrones. Le comportement de ces deux composants est décrit par des machines à états qui réagissent en fonction des événements qui arrivent. Ces événements correspondent à des messages entrants qui sont envoyés par l'autre composant.

Le modèle PingPong comprend quatre composants : les Things PingServer et PingClient, PingMsgs comme fragment de thing, et la configuration PingConfig. Dans ce modèle, le composant PingClient envoie un message paramétré ping au composant PingServer qui répond par un message pong après avoir reçu le message ping. Le statut de PingClient est modifié en fonction de la valeur des propriétés locales et du paramètre du message pong. Enfin, la configuration PingConfig représente une application concrète composée de deux instances, client et serveur, et d'un connecteur.

Le Listing 2.1 décrit le fragment de thing PingMsgs avec des messages paramétrés ping et pong. Les messages ping et pong sont utilisés pour mettre en œuvre le comportement des objets PingClient et PingServer. Ils comprennent un paramètre avec le type de données UInt8 de ThingML. Ce paramètre décrit le nombre de messages ping ou pong envoyés.

Listing 2.1 – ThingML implementation of PingMsgs thing fragment

```
thing fragment PingMsgs
{
    message ping ( req : UInt8 ) ;
    message pong ( req ' : UInt8 ) ;
}
```

Le Listing 2.2 montre la spécification ThingML de composant PingServer, y compris le fragment de thing PingMsgs, un port fourni et un statechart définissant leur comportement. Le port pingservice permet au thing PingServer d'échanger des messages avec d'autres things. Il permet d'envoyer le message pong et de recevoir le message ping. Le diagramme d'état PingServerMachine qui implémente le comportement de PingServer comprend deux états : Wainting et Pong. Il entre initialement dans l'état Waiting. Il passe de l'état Waiting à l'état Pong lorsqu'un message ping arrive sur le port pingservice. En

entrant dans l'état Pong, PingServer envoie un message pong via le port pingservice. Puis il passe à nouveau à l'état Waiting. La figure 2.5 présente graphiquement le diagramme d'état de la chose PingServer.

Listing 2.2 – ThingML implementation of PingServer thing

```
thing PingServer includes PingMsgs
{
    provided port pingservice
    {
       sends pong
       receives ping
    }
    statechart PingServerMachine init Waiting
    {
        property count : UInt8 = 0
        state Waiting
        {
           transition - > Pong
               event m pingservice ? ping // ? Receive the ping message on
                  pingservice port
               action count = m.req
        }
        state Pong
        {
            on entry pingservice ! pong ( count ) // Send message pong on
               pingservice port .
            on exit print "Send␣Pong" , count , "..."
            transition - > Waiting
        }
    }
}
```

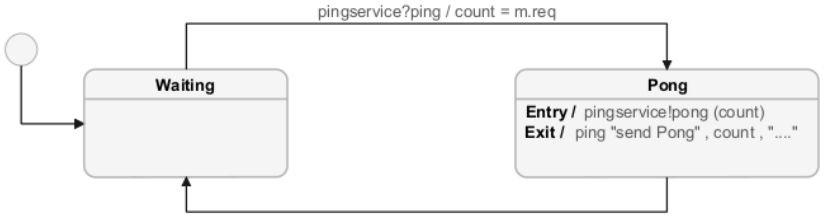

FIGURE 2.5 – Le diagramme d'état de PingServer

Le Listing 2.3 présente la spécification ThingML de PingClient, y compris le fragment de thing PingMsgs, deux propriétés, un port requis et un diagramme d'état définissant leur comportement.

Listing 2.3 – ThingML implementation of PingClient thing

```
thing PingClient includes PingMsgs{
    readonly property count_max : UInt8 = 5
    property counter : UInt8 = 1
    required port pingservice
```

```
{
    receives pong
    sends ping
}
statechart PingClient Machine init Ping
{
    state Ping
    {
        on entry do
            print "Send␣Ping" , counter , "..."
            pingservice ! ping ( counter )
        end
        transition > Waiting
    }
    state Waiting
    {
        transition - > Ping event e pingservice ? pong
            guard e.req ' == counter and counter < count_max
            action do println "␣[␣OK␣]␣"
                      counter< counter + 1
            end
        transition - > Stop event e guard e.req ' ! = counter
        transition - > OK event e pingservice ? pong
            guard e.req ' == counter and counter > = count_max
    }
    final state OK
    {
        on entry println "␣[␣OK␣]␣Bye␣.␣"
    }
    final state Stop
    {
       on entry println "␣[␣Error␣]␣" }
    }
}
```

Dans cette spécification, nous déclarons un port requis qui permet au thing *PingClient* d'échanger des messages avec d'autres things. Ce port permet d'envoyer le message ping et de recevoir le message pong. Comme le montre la spécification, les types de messages envoyés (resp. reçus) via le port requis correspondent aux types de messages reçus (resp. envoyés) via le port fourni. Le diagramme d'état PingClientMachine qui implémente le comportement *PingClient* comprend quatre états (Waiting, Ping, OK, et Stop), où OK et Stop sont les états finaux. Le thing PingClient entre initialement dans l'état Ping. En entrant dans l'état Ping, PingClient envoie un message ping via le port pingservice. Il passe ensuite à l'état Waiting, qui comporte trois transitions. Ces transitions se déclenchent lorsqu'un message ping arrive via le port pingservice, et que leurs conditions de garde sont satisfaites. Par conséquent, l'état PingClient passe de l'état Waiting à l'état Ping, OK, ou Stop, en suivant les conditions de garde. La Figure 2.6 présente graphiquement le diagramme d'état du PingClient.

Le Listing 2.4 décrit l'implémentation ThingML de l'application concrète PingConfig. La configuration PingConfig comprend deux instances et un connecteur. Le client est une instance de la chose PingClient, et le serveur est une instance de la chose PingServer. Le connecteur relie ces instances par l'intermédiaire de leurs ports, le port requis de l'instance client se trouvant à la première extrémité et le port fourni de l'instance serveur

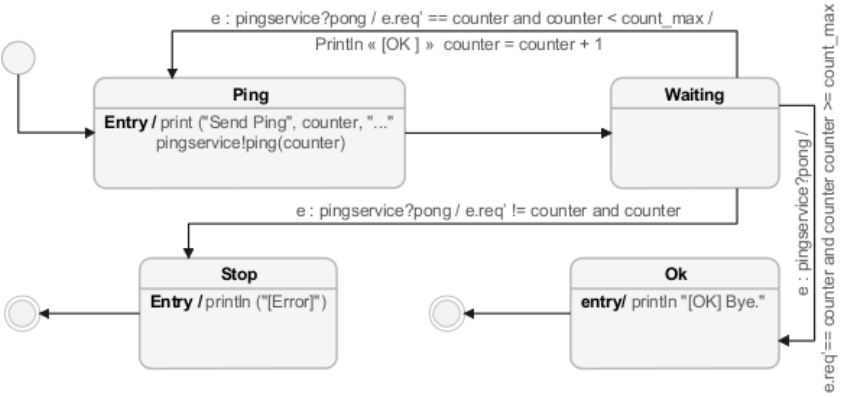

FIGURE 2.6 – Le diagramme d'état du PingClient

à la deuxième extrémité.

Listing 2.4 – ThingML implementation of PingConfig configuration

```
configuration PingConfig
{
    instance client : PingClient
    instance server : PingServer
    connector client.pingservice = > server pingservice
}
```

2.8 Conclusion

Dans ce chapitre, nous avons essayé de collecter de maximum d'informations sur le langage de modélisation ThingML soit les définitions, les éléments principaux avec ses déclarations et un exemple simple pour aider le lecteur de comprendre l'approche ThingML.

CHAPITRE 3

APPROCHE PROPOSÉE

3.1 Introduction

Dans ce chapitre, nous présenterons une approche qui vise à simuler le code source généré à partir de la spécification ThingML à l'aide du logiciel Proteus. Ensuite, nous présenterons les différents outils utilisés dans cette approche. Enfin, nous décrirons l'éditeur hybride proposé.

3.2 Présentation de notre approche

La figure 3.1 donne une présentation générale de notre approche. Dans la première étape, nous utilisons le langage ThingML pour décrire les applications IoT à l'aide de l'éditeur hybride proposé. Ensuite, nous transformons les spécifications obtenues vers le code source utilisant le générateur de code ThingML. Enfin, nous utilisons le logiciel Proteus pour concevoir le circuit matériel de l'application et simuler l'exécution du code source précédemment généré. Notre approche est structurée selon les étapes suivantes :

— **Étape 1 :** Conception et modélisation d'une application IoT utilisant le langage ThingML.

— **Étape 2 :** Après la modélisation des applications IOT , on passe à *la génération de code source*. La génération de code est effectuée à l'aide du cadre de génération de code ThingML. Dans ce travail, nous utiliserons la plateforme matérielle Arduino. Le code source qui en résulte est appelé *Sketches*.

— **Étape 3 :** L'étape suivante du processus consiste à compiler le code de Sketches en code de langage machine que l'Arduino exécute (un fichier d'extension "*.hex*"). Arduino IDE est utilisé pour effectuer cette compilation.

— **Étape 4 :** La simulation réalisée à l'aide du logiciel Proteus se compose de trois sous-opérations : conception du circuit matériel de l'application, définition des paramètres (définition de fichier programme (fichier exécutable)) et lancement de la simulation.

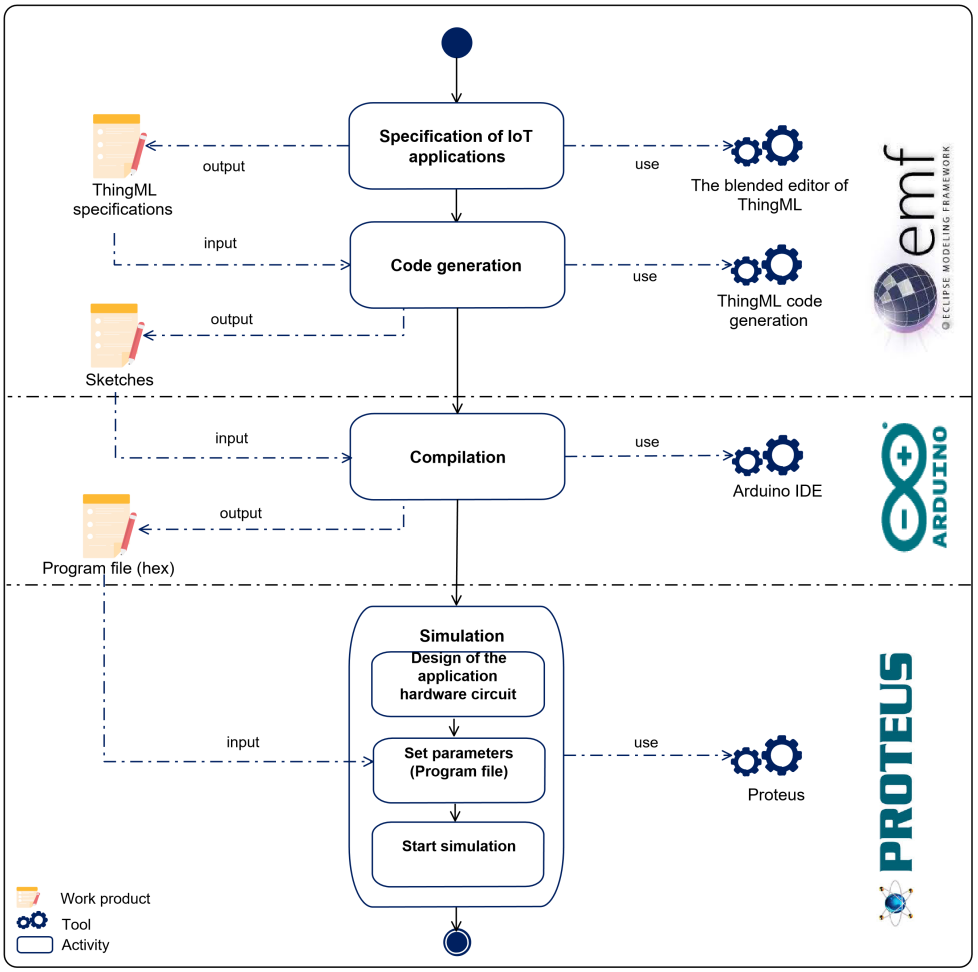

FIGURE 3.1 – Présentation générale de notre approche

3.3 Technologies sous-jacentes

Cette section fournit une vue générale de projet de modélisation Eclipse, de la plate-forme Arduino et du logiciel Proteus utilisé dans ce travail.

3.3.1 Projet de modélisation Eclipse

Le projet de modélisation d'Eclipse est une collection d'outils et de cadres pour l'IDM sur la plate-forme Eclipse. Les cadres du projet de modélisation Eclipse utilisés dans cette étude sont fournis dans les sections suivantes.

3.3.1.1 Cadre de modélisation Eclipse (EMF)

Un projet EMF est un cadre de modélisation open-source qui constitue la base de la construction de tous les cadres et les outils du projet de modélisation Eclipse. Il utilise le langage de méta-modélisation orienté objet Ecore pour définir la syntaxe abstraite des langages de modélisation développés (méta-modèle). Pour définir la syntaxe concrète et construire des bancs d'éditeurs pour le langage de modélisation (graphique ou textuel), les concepteurs peuvent utiliser de nombreux comme Sirius et Xtext.

3.3.1.2 Cadre Sirius

Le cadre Sirius est un projet Eclipse open-source qui permet la création d'éditeurs graphiques de DSL. Il est construit sur les technologies de modélisation d'Eclipse, comme EMF et GMF. L'objectif est de fournir un atelier générique pour l'ingénierie d'architecture basée sur des modèles qui pourrait simplifier la production, réduire le temps de conception et augmenter la productivité lors de la construction d'un éditeur graphique. Contrairement à GMF, qui génère une quantité massive de code très complexe, avec Sirius, aucun code n'est généré à partir de la spécification, mais elle est interprétée. Par conséquent, les modifications apportées à la spécification prennent immédiatement effet dans l'éditeur de diagrammes. Ainsi, le développeur peut être testé et utilisé dans la même instance Eclipse. Sirius est également intégré avec d'autres technologies qui donnent plus de force à l'outil, comme l'intégration Sirius-Xtext [42].

Sirius utilise une méthode basée sur Viewpoint, permettant aux développeurs de langage de concevoir de nombreuses syntaxes graphiques concrètes pour un seul modèle EMF. En conséquence, un seul modèle peut être modifié à l'aide de divers éditeurs basés sur Sirius. Sirius permet de développer des éditeurs graphiques à l'aide d'un projet de spécification de Viewpoint (VSP). Une spécification de Viewpoint est une structure hiérarchique qui décrit la structure, l'apparence et le comportement des modèles, par exemple, une spécification de Viewpoint de diagramme contient des nœuds et des arêtes représentant les objets du modèle et leurs relations.

3.3.1.3 Cadre Xtext

Xtext est un composant EMF qui permet de construire des DSL basés sur le texte. Il définit une grammaire DSL en utilisant un langage similaire à l'EBNF (Extended Backus-Naur Form), qui peut être utilisé pour produire un méta-modèle basé sur Ecore et une infrastructure associée telle qu'un analyseur syntaxique et un éditeur de liens. En outre, Xtext fournit un éditeur de texte complet avec des fonctions d'aide au développeur tel que l'affichage du code, la syntaxe et les indicateurs d'erreur [34].

3.3.2 Plate-forme Arduino

Dans le domaine de l'électronique, Arduino un ensemble d'outils matériels et logiciel pour le prototypage électronique et l'apprentissage de la programmation des microcontrôleurs, c'est une plateforme open-source bien connue. Elle a été conçue pour être conviviale pour ceux qui n'ont aucune connaissance préalable en électronique. Arduino permet de créer des objets capables de contrôler un moteur, d'allumer une lumière, d'envoyer des alertes, etc [43]. Il est principalement basé sur deux composants : matériel et logiciel.

- *Le matériel Arduino :* La carte Arduino possède un microcontrôleur facilement programmable ainsi que de nombreuses entrées-sorties. Elle est capable de contrôler et de répondre aux composants qui lui sont connectés. Ces composants peuvent être des capteurs ou des actionneurs (lumières et LEDs, relais, écrans, moteurs) qui leur permettent de communiquer avec le monde extérieur [44].

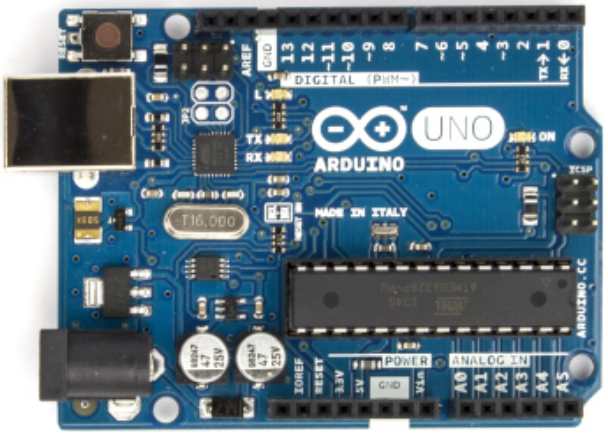

FIGURE 3.2 – La carte Arduino

- *Le logiciel Arduino :* La carte Arduino peut facilement être programmée en utilisant l'environnement de développement intégré (IDE) Arduino : C'est le logiciel de programmation de la carte Arduino permet aux utilisateurs de créer des programmes appelés sketches, en utilisant une version simplifiée de C++. Ensuite, il convertit ces programmes en code objet compatible avec le matériel Arduino. De plus, l'IDE télécharge ces codes objets sur la carte Arduino via une connexion USB [42]. La figure 3.3 représente la structure générale du programme sur la carte Arduino :

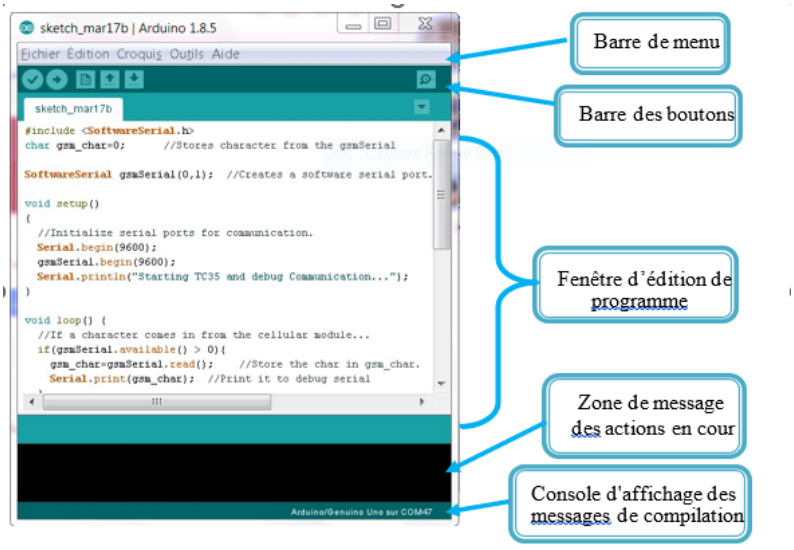

FIGURE 3.3 – Interface IDE Arduino

3.3.3 Proteus software

Développée par Labcenter Electronics, Proteus est une suite logicielle utilisée principalement pour concevoir des schémas électriques. Elle comprend différents paquets tels que Proteus PCB Design pour le circuit imprimé et Proteus Virtual System Modeling (VSM) pour la simulation. Proteus Virtual System Modeling (VSM) pour la simulation.

Le paquet VSM est utilisé pour simuler des circuits avec des microprocesseurs. Il permet de prototyper rapidement des conceptions de matériel et de micrologiciels. Proteus VSM permet de simuler l'interaction entre un logiciel fonctionnant sur un microcontrôleur et toute électronique analogique ou numérique qui lui est connectée. De plus, il simule l'exécution du code objet (code machine), tout comme une vraie puce. En outre, Proteus VSM supporte plusieurs familles de microprocesseurs (PIC16, PIC18, AVR, Arduino, ARM, ...). La suite logicielle Proteus possède plusieurs avantages comme [45] :

— Pack contenant des logiciels faciles et rapides à comprendre et utiliser.

— Le support technique est performant.

— L'outil de création de prototypes virtuel permet de réduire les coûts matériels et logiciels lors de la conception d'un projet.

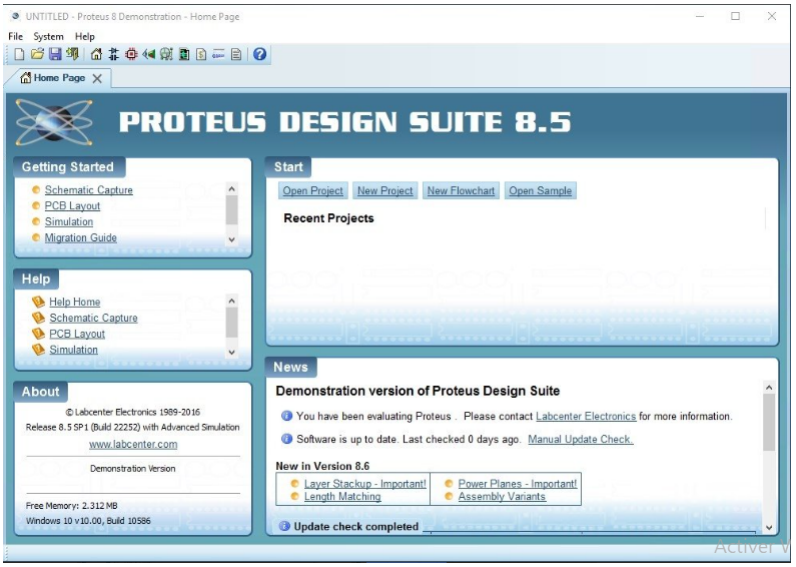

FIGURE 3.4 – Logiciel de CAO Proteus

3.4 Éditeur de modélisation hybride graphique et textuel

Un diagramme peut être construit à partir des données conservées dans des fichiers texte en utilisant Sirius et Xtext. Dans ce cas, vous obtenez une représentation graphique de la même information et vous pouvez l'éditer en utilisant soit le diagramme, soit l'éditeur de texte. La synchronisation entre les deux représentations se fait lors de l'enregistrement d'un éditeur [46].

Les éditeurs hybrides graphiques-textuels présentent des avantages tels que la modélisation plus rapide des tâches, les langages textuels peuvent réduire le nombre de clics lors de la création et de la modification des modèles, mais il a été démontré que les langages graphiques réduisent le temps de liaison des éléments de modèle entre eux, et la possibilité d'éditer les modèles en dehors d'un environnement de modélisation [46].

Comme montré ci-dessus, Sirius utilise une méthode basée sur Viewpoint, permettant aux développeurs de langage de concevoir de nombreuses syntaxes graphiques concrètes

pour un seul modèle EMF. Dans ce travail, nous avons développé trois Viewpoint pour la visualisation et l'édition des Things, de la machine à états et de configurations :

— **Le viewpoint Things :** un langage de modélisation orienté objet dans lequel les caractéristiques structurelles, les classes et les héritages sont modélisés graphiquement, mais le corps des opérations est spécifié textuellement.

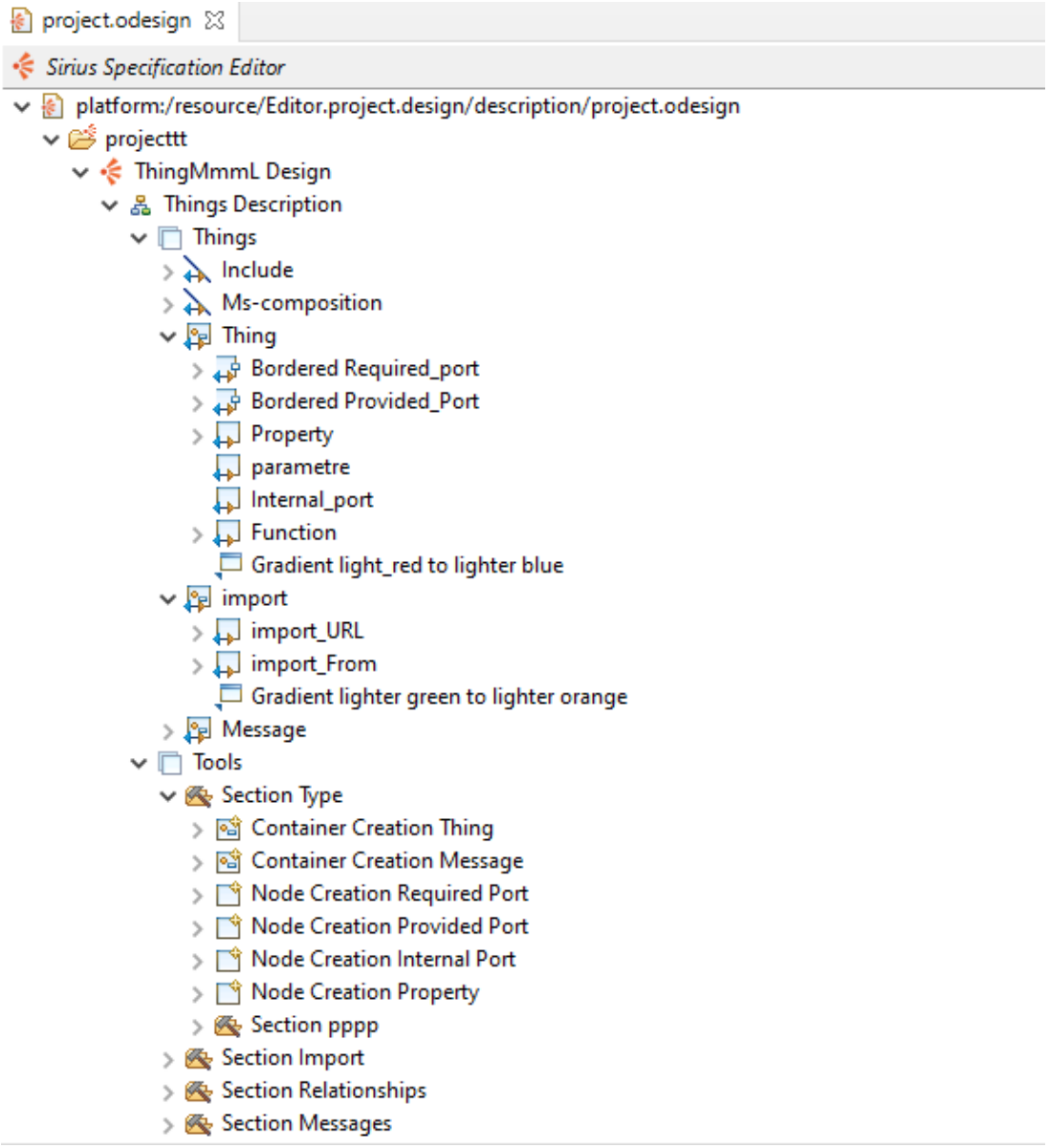

FIGURE 3.5 – La présentation de viewpoints Things

— **Le viewpoint State machine :** un langage de machine à états dans lequel les états et les transitions sont spécifiés graphiquement, mais les gardes dans les transitions et les actions dans les états sont spécifiées à l'aide d'une expression textuelle.

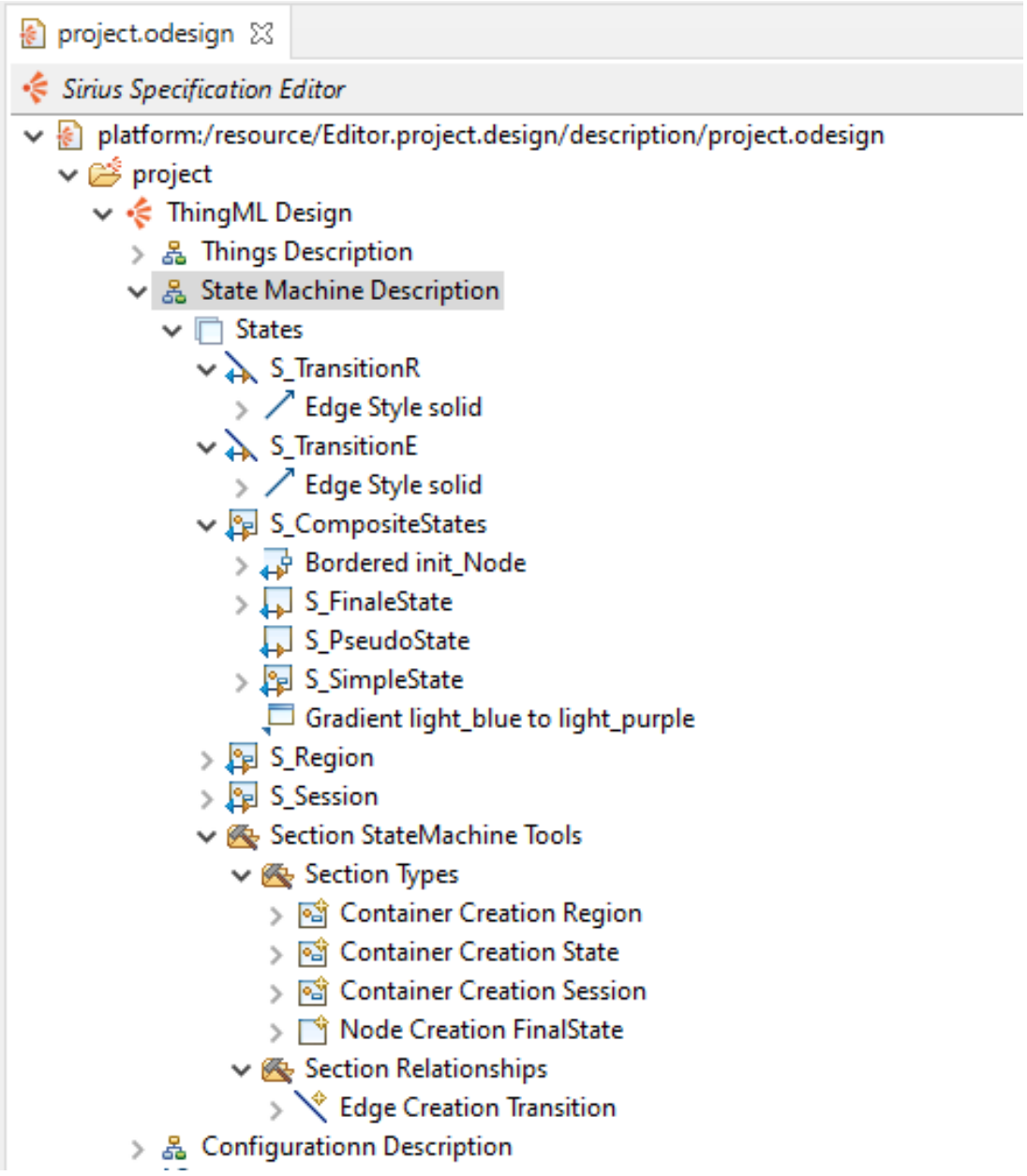

FIGURE 3.6 – La présentation de viewpoints State machine

— **Le viewpoint Configuration :** la figure 3.7 montre le viewpoint Configuration qui nous permet d'éditer et de visualiser les configurations qui consistent en des instances et des connecteurs.

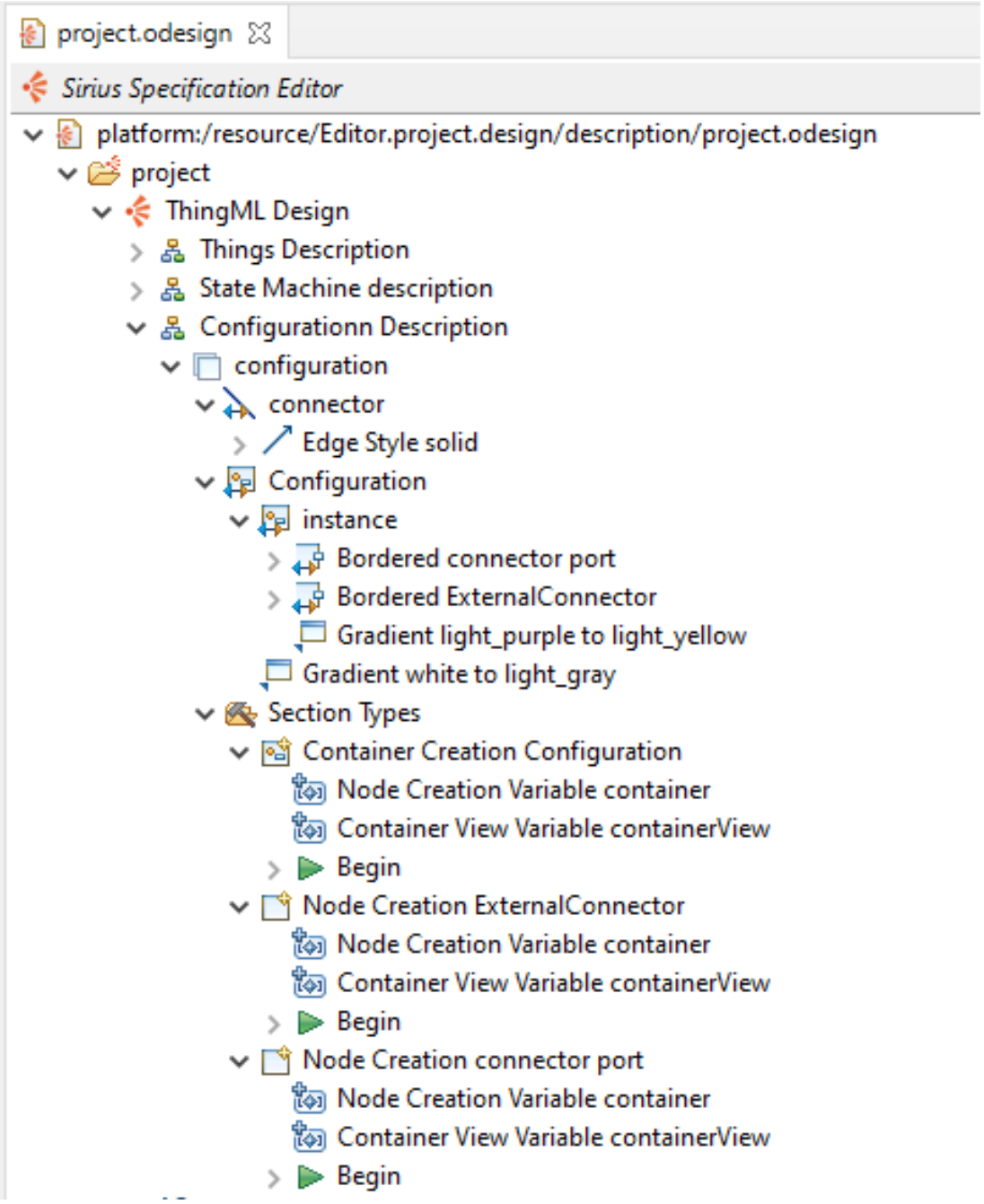

FIGURE 3.7 – La présentation de viewpoints configuration

3.5 Conclusion

Dans ce chapitre, nous avons présenté une approche basée sur l'IDM et la simulation. Puis nous avons présenté les outils utilisés dans cette approche. Enfin, nous avons montré l'éditeur hybride qui a été proposé pour faciliter la modélisation avec ThingML.

CHAPITRE 4

ÉTUDES DE CAS

4.1 Introduction

Pour simplifier la vie quotidienne, le développement des applications IoT insiste de faire des modèles intelligents sur des domaines différents. Par exemple, les feux de circulation afin de rendre le transport confortable, pratique et plus facile avec l'économie d'énergie, gagner du temps, réduire la pollution . . .etc. En plus, le détecteur de gaz pour la protection de la vie humaine et la construction d'une domotique (smart home).

Dans ce chapitre, nous avons réalisé deux modèles, le premier est pour « Traffic-light », le deuxième pour un détecteur de gaz. En expliquant la conception et les principes de fonctionnement pour chacun.

4.2 Feu de circulation

Notre premier cas d'étude dans le cadre des systèmes d'IoT, il s'intéresse particulièrement au domaine des villes intelligentes. Un système des feux de circulation joue un rôle important dans notre vie, car il régule la circulation sur les routes afin de faciliter le processus de transport. Considérons un système de feu de circulation composé de trois lumières colorées : rouge, vert et jaune. Une seule des couleurs est active dans un instant donné. Ce système fonctionne dans le temps et la couleur de feu change dans l'ordre rouge, vert, jaune, puis rouge à nouveau, et ainsi de suite. Nous utilisons des outils matériaux et logiciels pour modéliser le comportement de notre système en fonction des différents états possibles.

4.2.1 Conception

Dans la première étape, nous allons spécifier notre système avec le langage ThingML à l'aide de l'éditeur hybride développé. Le modèle de feu de circulation comprend quatre composants : les Things *LED* et *Traffic_Light*, *LEDMsgs* comme fragment de thing, et la configuration *Traffic_Light_App*. La figure 4.1 montre la présentation graphique de feu de la circulation (partie Thing) a l'aide de notre éditeur.

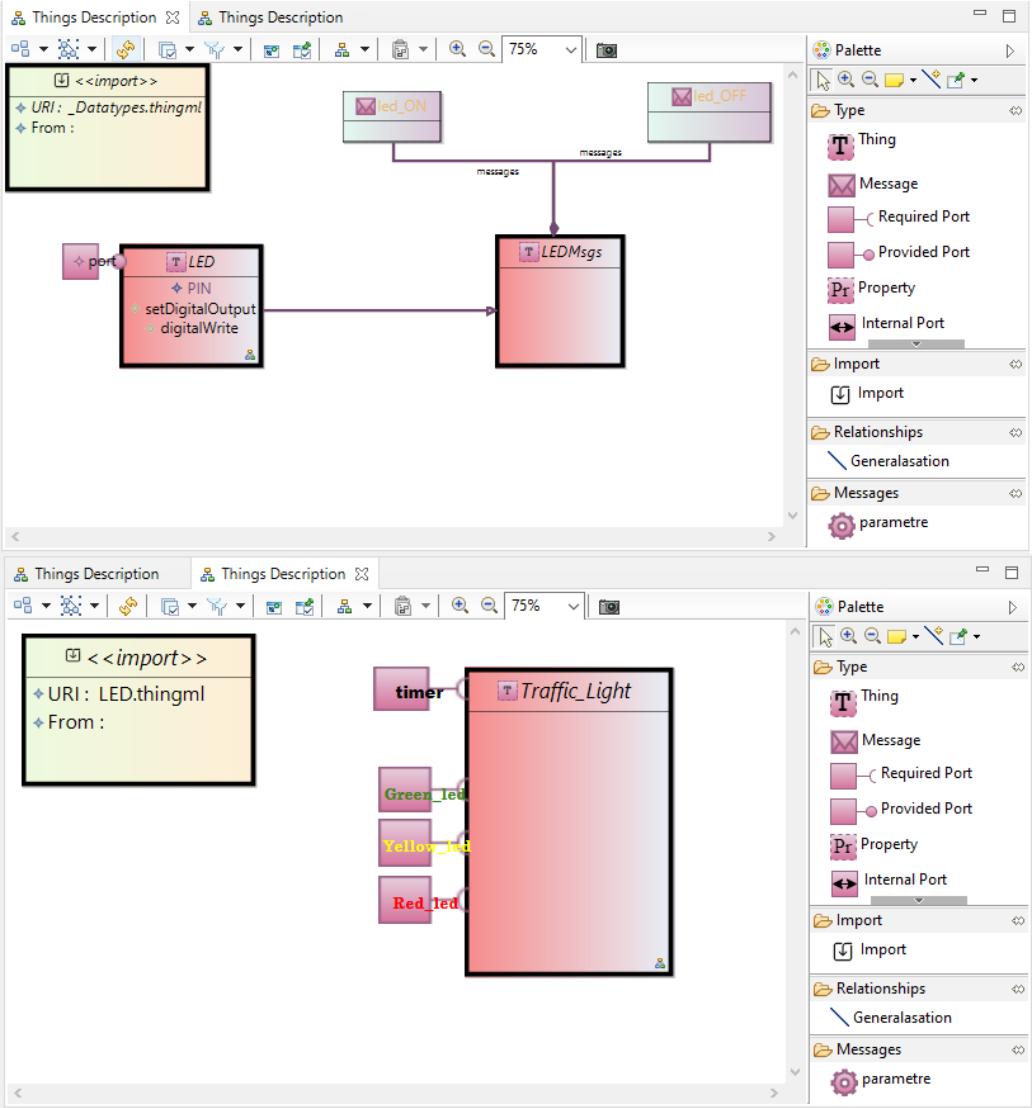

FIGURE 4.1 – Présentation graphique de feux de circulation (partie Thing)

Le Listing 4.1 décrit le fragment de thing *LEDMsgs* avec des messages *led_ON* et *led_OFF*. Les messages *led_ON* et *led_OFF* sont utilisés pour mettre en œuvre le comportement des Things *LED* et *Traffic_Light*.

Listing 4.1 – La spécification de thing fragmenté LEDMsgs

```
thing LEDMsgs
{
    message led_ON();
    message led_OFF();
}
```

Le Listing 4.2 montre la spécification ThingML de composant *LED*, y compris le fragment de thing *LEDMsgs*, un port fourni, deux fonctions et un statechart définissant leur comportement. Le port *ctrl* permet au thing *LED* d'échanger des messages avec d'autres things. Il permet de recevoir les messages *led_ON* et *led_OFF*. Le diagramme d'état *Impl_LED* qui implémente le comportement de thing LED comprend deux états : *ON* et *OFF*. Il entre initialement dans l'état *OFF*. Il passe de l'état *OFF* à l'état *ON* lorsqu'un message *led_ON* arrive sur le port *ctrl*. Dans cette transition, le thing LED exécuté la

fonction *digitalWrite* avec le paramètre *HIGH*. Le thing LED passe de l'état *ON* à l'état *OFF* lorsqu'un message *led_OFF* arrive sur le port *ctrl*. Dans cette transition, le thing LED exécute la fonction *digitalWrite* avec le paramètre *LOW*. La fonction *digitalWrite* permet d'allumer ou d'ateindre les LEDs suivant la valeur de paramètre (HIGH ou LOW).

Listing 4.2 – La spécification de Thing LED

```
thing LED includes LEDMsgs {
    property PIN: UInt8 = 10
    provided port ctrl
    {
        receives led_ON, led_OFF
    }
    function setDigitalOutput(pin: UInt8) do
        `pinMode(`&pin&`, OUTPUT);`
    end
    function digitalWrite(pin: UInt8, value : DigitalState) do
        `digitalWrite(`&pin&`, `&value&`);`
    end

    statechart Impl_LED init OFF
    {
        on entry setDigitalOutput(PIN)
        state OFF
        {
            transition -> ON
                event ctrl?led_ON
                action digitalWrite(PIN, DigitalState:HIGH)
        }
        state ON
        {
            transition -> OFF
                event ctrl?led_OFF
                action digitalWrite(PIN, DigitalState:LOW)
        }
    }
}
```

La figure 4.2 présente graphiquement le diagramme d'état de Thing LED.

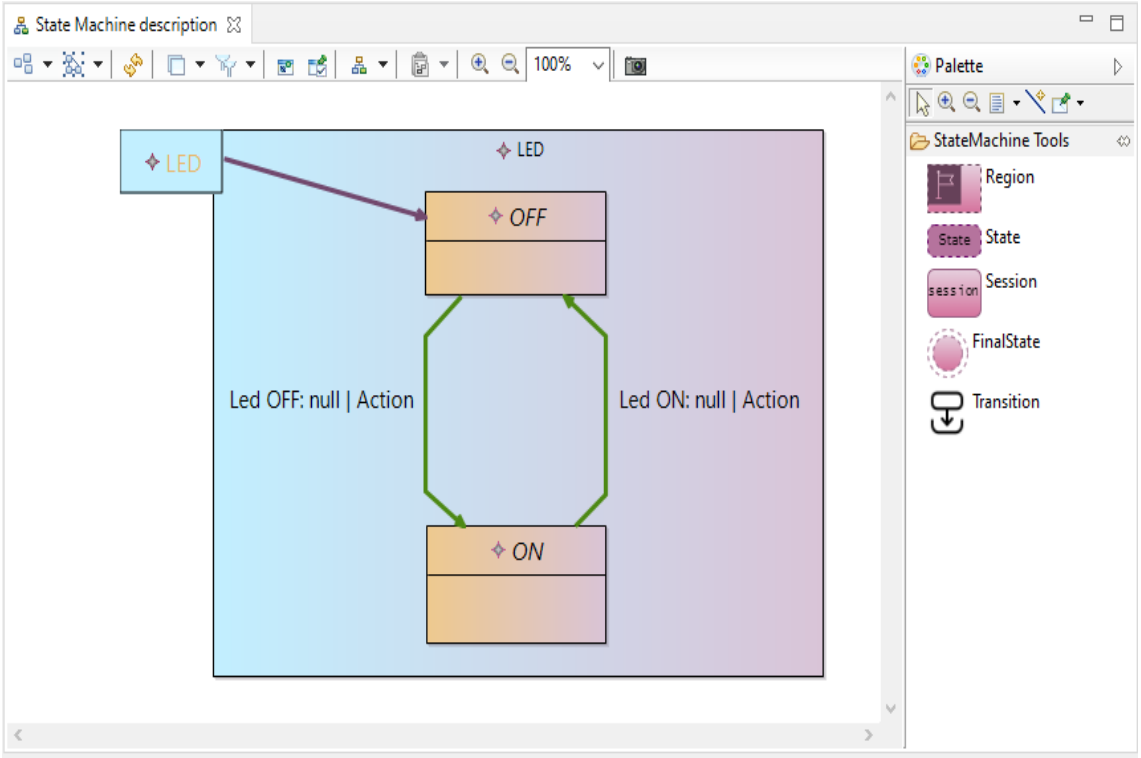

FIGURE 4.2 – Présentation graphique de diagramme d'état de Thing LED.

Cette spécification inclut deux intructions spécifiques à la plateform Arduino [43] :

— **pinMode() :** Configure la broche spécifiée pour qu'elle se comporte soit comme une entrée (INPUT), soit comme une sortie (OUTPUT).

— **digitalWrite() :** Écrivez une valeur HIGH ou LOW sur une broche numérique. Si la broche a été configurée en OUTPUT avec pinMode(), sa tension sera réglée sur la valeur correspondante : 5V pour HIGH, 0V (masse) pour LOW.

Le Listing 4.3 présente l'implémentation de la partie principale de l'application *Traffic_Light*, y compris le fragment de things *LEDMsgs* et *TimerMsgs*, quatre required ports et un statechart définissant leur comportement. Le diagramme d'état *Impl_Traffic_Light* comprend trois états :*RED*, *GREEN* et *YELLOW*. Il entre initialement dans l'état *RED*. Il passe d'un état a l'autre lorsqu'un message *timer_timeout* arrive sur le port *timer*.

Listing 4.3 – La spécification ThingML de thing Traffic_Light

```
thing Traffic_Light includes TimerMsgs, LEDMsgs {
    required port timer
    @sync_send "true"
    {
        receives timer_timeout
        sends timer_start
    }
    required port Red_led {
        sends led_ON, led_OFF
    }
    required port Green_led {
        sends led_ON, led_OFF
    }
    required port Yellow_led {
        sends led_ON, led_OFF
```

```
    }
    statechart Impl_Traffic_Ligh init RED {
        state RED {
            on entry do
                Red_led!led_ON()
                timer!timer_start(0, 500)
            end
            transition -> GREEN event timer?timer_timeout
            on exit Red_led!led_OFF() }
        state GREEN {
            on entry do
                Green_led!led_ON()
                timer!timer_start(0, 800)
            end
            transition -> YELLOW event timer?timer_timeout
            on exit Green_led!led_OFF() }
        state YELLOW {
            on entry do
                Yellow_led!led_ON()
                timer!timer_start(0, 300)
            end
            transition -> RED event timer?timer_timeout
            on exit Yellow_led!led_OFF()}
}}
```

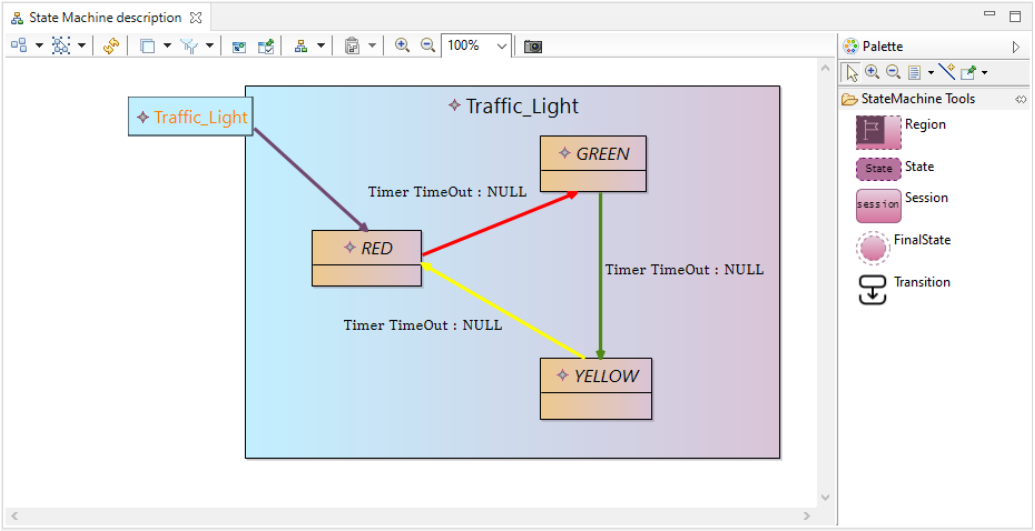

FIGURE 4.3 – Présentation graphique de diagramme d'état de Thing Traffic_Light.

On termine par la configuration, cette partie. On déclare les instances des choses, comme vous remarquez dans le code, il y a quatre instances, chaque instance a un connecteur qui gère la communication entre eux.

Listing 4.4 – partie 3 TrafficLight.thingml

```
configuration Traffic_Light_App
{ instance traffic_light : Traffic_Light
    instance red_led : LED
    set red_led.PIN = 11
    instance green_led : LED
    set green_led.PIN = 12
```

```
    instance yellow_led : LED
    set yellow_led.PIN = 13
    connector traffic_light.Red_led => red_led.ctrl
    connector traffic_light.Green_led => green_led.ctrl
    connector traffic_light.Yellow_led => yellow_led.ctrl
    connector traffic_light.timer over Timer
}
```

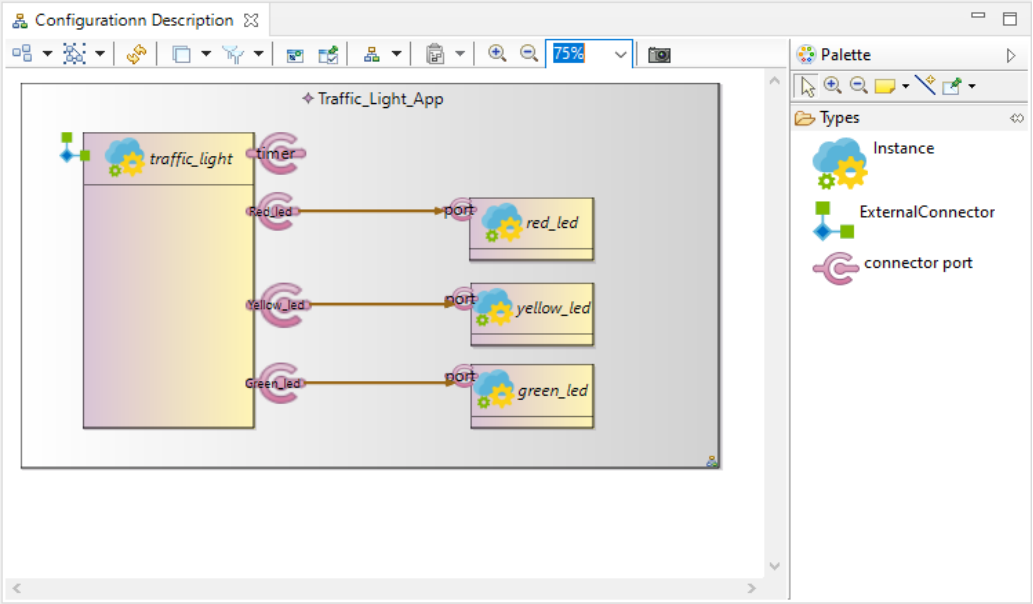

FIGURE 4.4 – Une éxtrait de présentation graphique de feu du circulation (partie configuration)

4.2.2 Génération de code

Après la modélisation du système, nous utilisons le générateur de code ThingML pour générer automatiquement le code source. Il existe deux méthodes, l'une utilisant la plateforme Eclipse ou le fichier Jar. La Figure 4.5 montre le processus de génération de code en utilisant le fichier Jar.

```
abdo@Aspire-V5-123: /mnt/wind/ThingML
abdo@Aspire-V5-123:/mnt/wind/ThingML$ java -jar ThingML-CLI.jar -c arduino -s /mnt/wind/ThingML/Traffic_Li
ght/TrafficLight.thingml -o /mnt/wind/ThingML/TT
Running ThingML -c arduino -s /mnt/wind/ThingML/Traffic_Light/TrafficLight.thingml -o /mnt/wind/ThingML/TT
Checking for EMF errors and warnings
Checking for EMF errors and warnings
Checking for EMF errors and warnings
  _____ _     _            __  __ _
 |_   _| |__ (_)_ __   __ _|  \/  | |
   | | | '_ \| | '_ \ / _\`| |\/| | |
   | | | | | | | | | (_| | |  | | |__
   |_| |_| |_|_|_| |_|\__, |_|  |_|____|
                       |___/
Generating code for configuration: Traffic_Light_App. InputDirectory is /mnt/wind/ThingML/Traffic_Light
Running C/C++ for Arduino (AVR Microcontrollers) compiler on configuration Traffic_Light_App [Sat Dec 18 1
3:32:14 CET 2021]
Checking configuration...
Checking configuration... Done! Took 796 ms.
Compilation complete [Sat Dec 18 13:32:20 CET 2021]. Took 223 ms.
  _____ _     _            __  __ _
 |_   _| |__ (_)_ __   __ _|  \/  | |
   | | | '_ \| | '_ \ / _\`| |\/| | |
   | | | | | | | | | (_| | |  | | |__
   |_| |_| |_|_|_| |_|\__, |_|  |_|____|
                       |___/
SUCCESS.
abdo@Aspire-V5-123:/mnt/wind/ThingML$
```

FIGURE 4.5 – La première méthode de génération

La deuxième est de faire la génération sur Eclipse modeling tool, comme vous pouvez le voir sur la figure 4.6 :

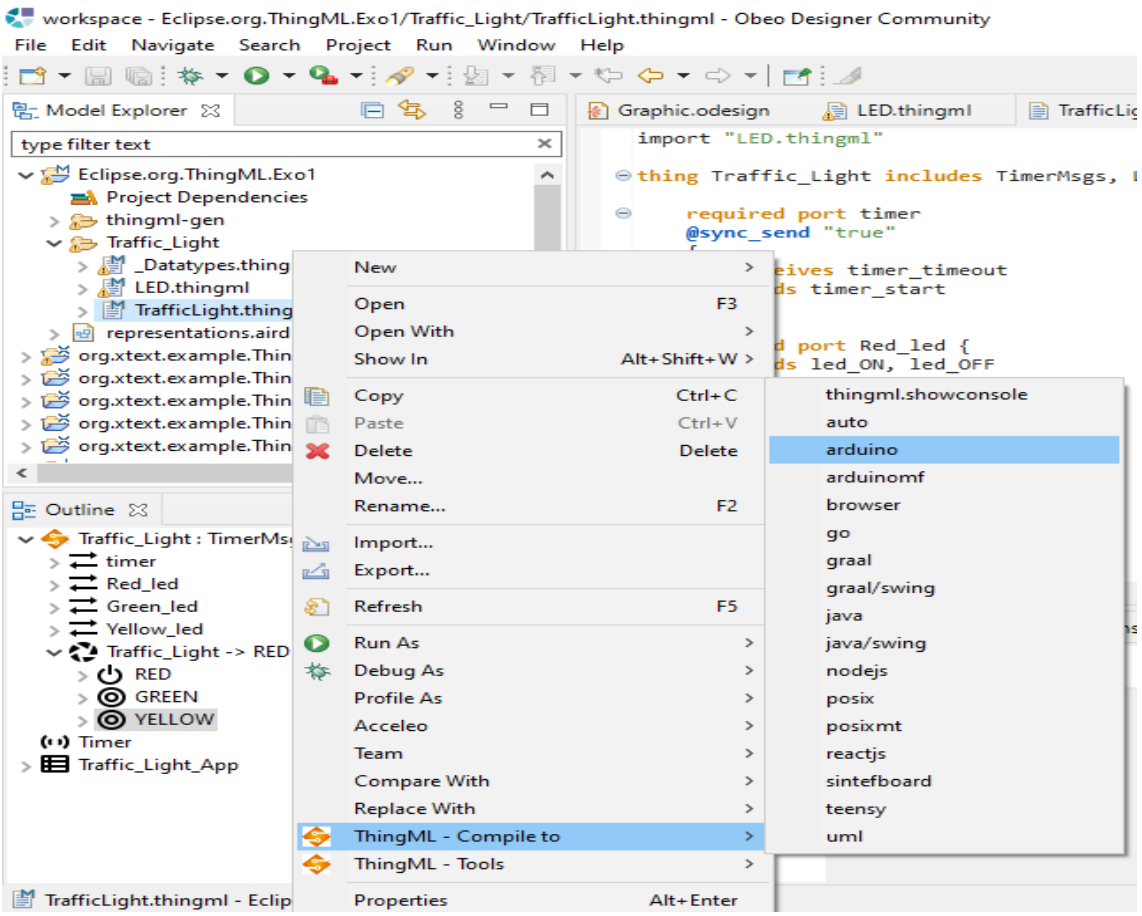

FIGURE 4.6 – La deuxième méthode de génération de code

La figure 4.7 présente le résultat de génération qui est un fichier qui contient le code source (Sketches) :

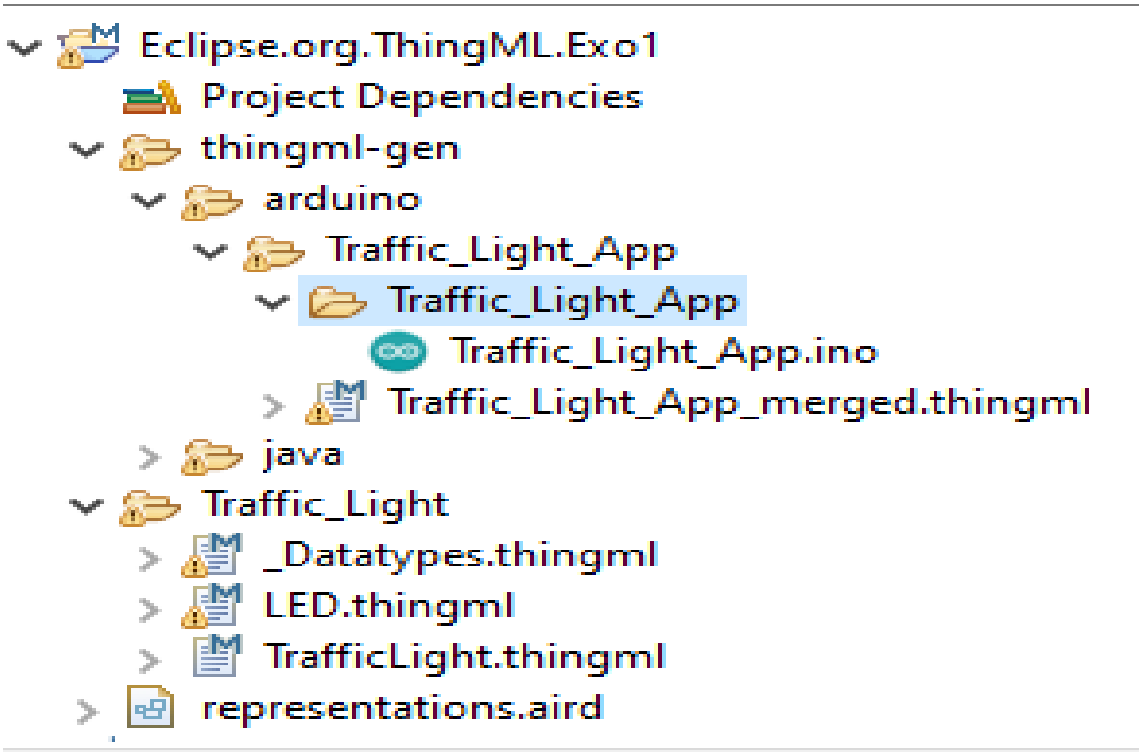

FIGURE 4.7 – la suite de génération de code de feu de circulation

4.2.3 Compilation

Dans cette étape le processus consiste à faire la compilation de code sketches en code de langage machine que l'Arduino exécute. Nous avons cliqué sur croquis > Verify/Compile dans la barre de menus. La Figure 4.8 montre les résultats que vous devriez obtenir si tout a fonctionné correctement ou pas :

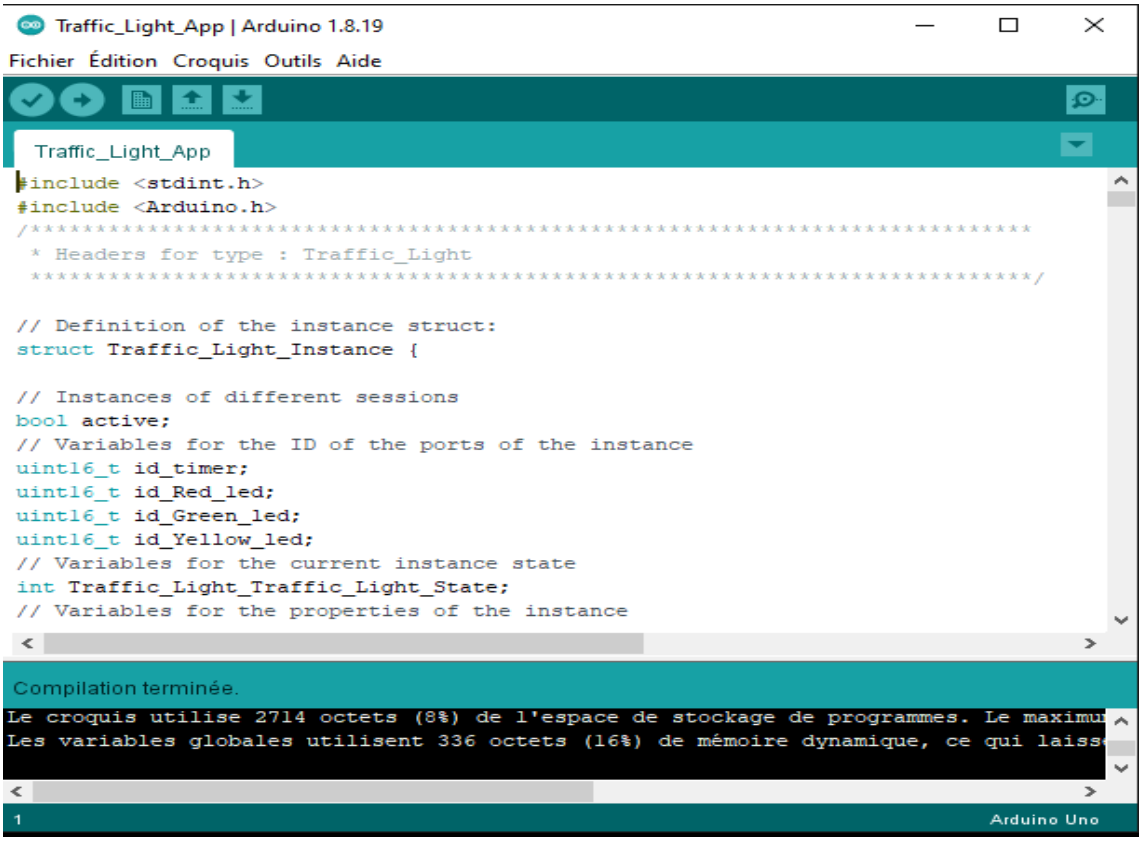

FIGURE 4.8 – Compilation de code de feu de circulation

Vous devriez voir dans la figure un message dans la zone de message indiquant que la compilation est terminée, et la fenêtre de la console devrait montrer la taille finale du code en langage machine compilé qui sera téléchargé sur Arduino, donc le code est correct.

4.2.4 Simulation :

Dans l'étape de simulation, nous construisons d'abord le circuit matériel de l'application. Le circuit matériel des feux de circulation illustré à la figure 4.9 a été créé à l'aide du programme Proteus :

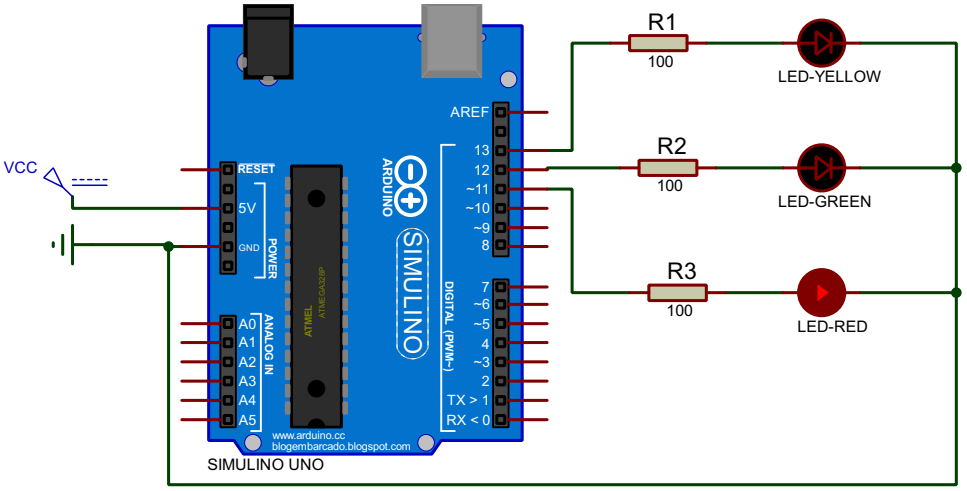

FIGURE 4.9 – Le circuit de feu de circulation

Nous définissons ensuite le fichier programme du composant Arduino par le fichier objet obtenu à l'étape de compilation. Nous double-cliquons sur le composant Arduino, et dans la fenêtre apparie, nous choisissons le fichier programme. Enfin, nous simulons en cliquant sur l'icône Run simulation dans la barre d'outils de Proteus. Nous avons effectué une série de simulations qui démontrent que le code source généré par la génération de code ThingML à partir de notre spécification est exécuté correctement sur le circuit développé dans Proteus . Le tableau 4.1 présente les résultats des simulations pour 10, 20 et 50 cycles de feux de circulation (un cycle est le passage du rouge au vert, au jaune, puis au rouge).

TABLE 4.1 – Résultats de simulation

	10 cycles	**20 cycles**	**50 cycles**
Execution time	160 s	320 s	800 s
Result	executes correctly	executes correctly	executes correctly

4.3 Détecteur de gaz

Après la maison intelligente, un détecteur de gaz rejoint le monde de la domotique. Un detecteur de gaz fait l'objet de notre deuxième cas d'étude pour voir s'il y a une fuite de gaz inattendue à distance tout en utilisant une application afin de recevoir des informations sur la valeur de gaz capter par le système et qui va nous surveiller la vie humaine en permanence.
Un détecteur de gaz composé de deux lumières colorées : rouge et vert. Une seule des couleurs est active selon le taux de fuite de gaz. Ce système fonctionne dans le temps et le couleur de feu change dans l'ordre rouge, vert, puis rouge à nouveau, et ainsi de suite. Nous avons comporté les différentes étapes nécessaires pour la réalisation de ce système, une présentation textuelle et graphique avec les logiciels utilisés pour obtenir une solution performante.

4.3.1 Conception :

Dans cette partie, nous avons répété les étapes précédentes sur le deuxième système, soit la représentation textuelle ou bien graphique, ou la génération de code sans oublier l'étape de simulation.

L'implémentation de l'exemple détecteur de gaz comprend plusieurs composants : les Things *LED*, *GasSensor* et *GasDetector*, *LEDMsgs* et *GasSensorMsgs* comme fragment de thing, et la configuration *Gas_Detector_App*. Les figures 4.10, 4.11 et 4.12 montrent la présentation graphique de détecteur de gaz (partie Thing) à l'aide de notre éditeur.

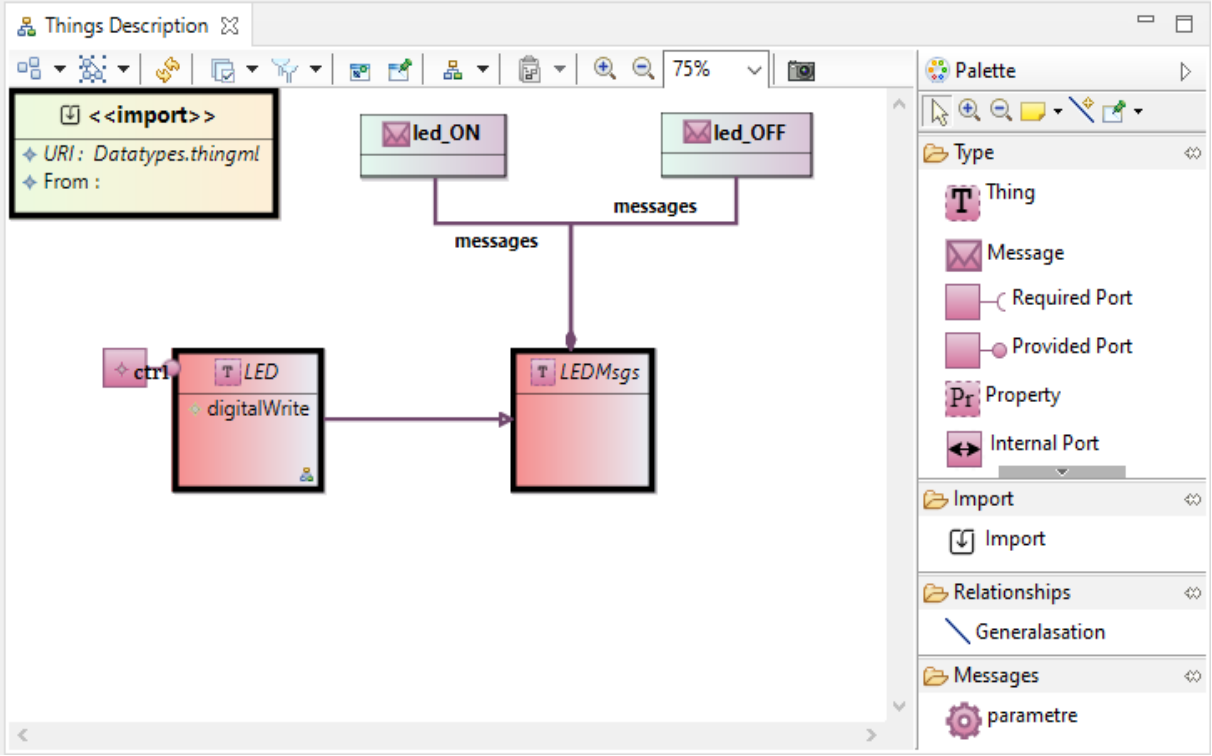

FIGURE 4.10 – un extrait de présentation graphique du détecteur de gaz (partie Thing)

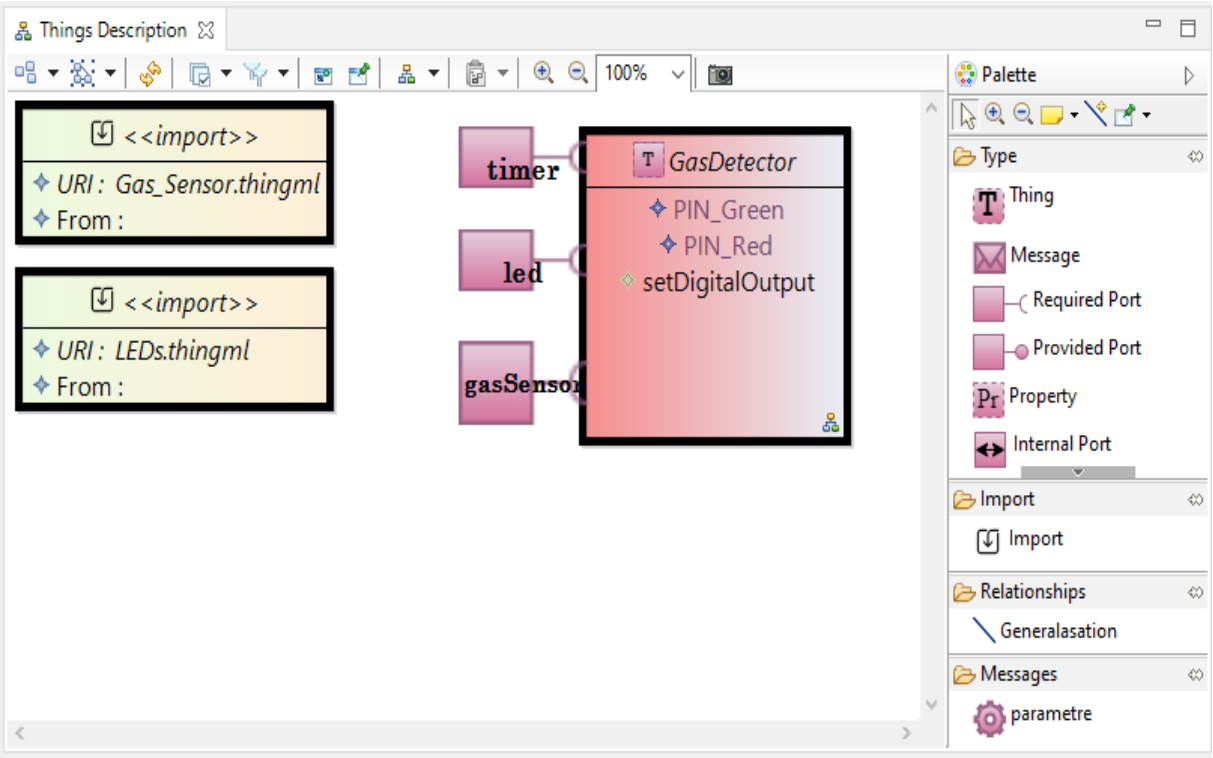

FIGURE 4.11 – un extrait de présentation graphique du détecteur de gaz (partie Thing)

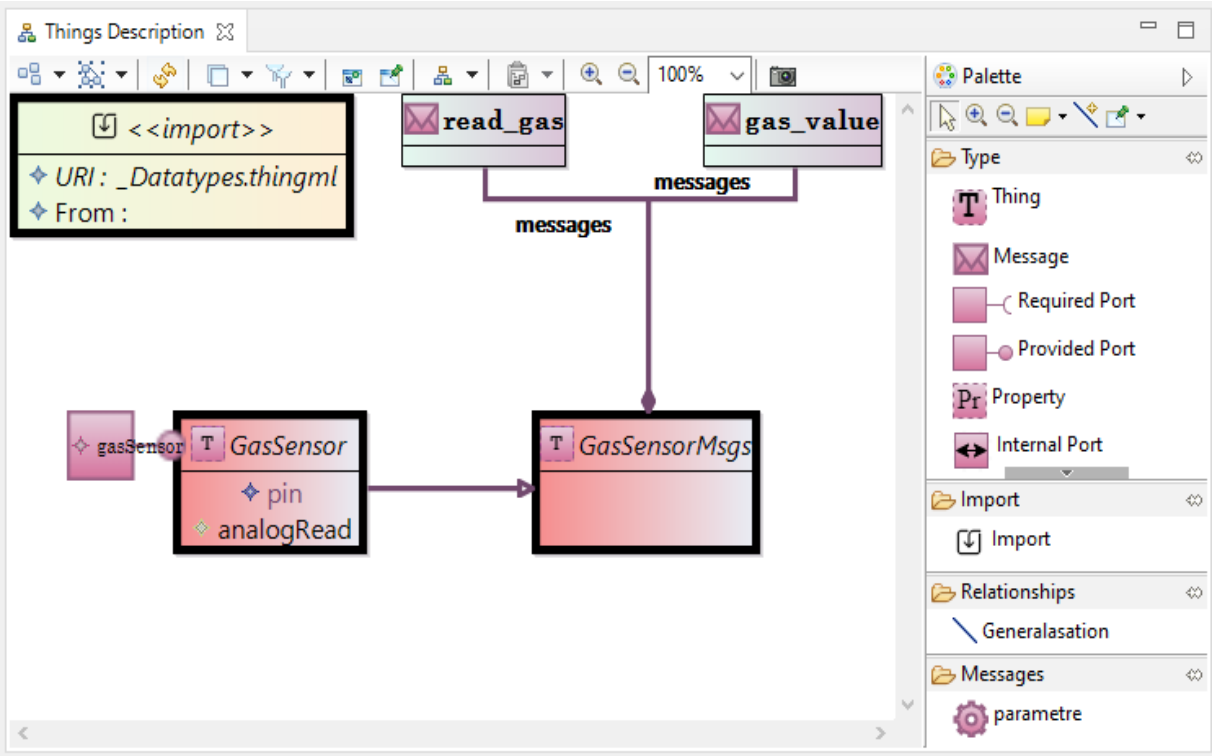

FIGURE 4.12 – un extrait de présentation graphique du détecteur de gaz (partie Thing)

Le Listing 4.5 décrit le fragment de thing *LEDMsgs* avec des messages *led_ON* et *led_OFF*. Les messages *led_ON* et *led_OFF* sont des messages avec les paramètres utilisés pour mettre en œuvre le comportement des Things *LED* et *GasDetector*.

Listing 4.5 – La spécification de thing fragmenté LEDMsgs

```
thing LEDMsgs {
        message led_ON(pin : UInt8);
        message led_OFF(pin : UInt8);
}
```

Le Listing 4.6 montre la spécification ThingML de composant *LED*,contient le fragment de thing *LEDMsgs*, un port fourni, la fonction *digitalWrite* permet d'allumer ou d'ateindre les LEDs suivant la valeur de paramètre (HIGH ou LOW), et à la fin on a principalement un état machine définissant leur comportement. Le port *ctrl* permet de recevoir les messages *led_ON* et *led_OFF* au thing *LED* à un autre thing . Le diagramme d'état *LedImpl* qui implémente le comportement de thing LED comprend un seul état :*READY*. Lorsqu'un message *led_ON* arrive sur le port *ctrl*. Dans cette transition, le thing LED execute la fonction *digitalWrite* avec le paramatre *HIGH*. Le thing LED passe de l'état *ON* à l'état *OFF* lorsqu'un message *led_OFF* arrive sur le port *ctrl*. Dans cette transition, le thing LED execute la fonction *digitalWrite* avec le paramètre *LOW*.

Listing 4.6 – la spécification ThingML de composant *LED*

```
thing LED includes LEDMsgs {
        provided port ctrl {
                receives led_ON, led_OFF
        }

        function digitalWrite(pin: UInt8, value : DigitalState) do
                `digitalWrite(`&pin&`, `&value&`);`
        end

        statechart LedImpl init READY {
                state READY {
                        internal event e : ctrl?led_ON
                                action digitalWrite(e.pin, DigitalState:HIGH)
                        internal event e : ctrl?led_OFF
                                action digitalWrite(e.pin, DigitalState:LOW)
                }
        }
}
```

-voici la représentation graphique de diagramme d'état de Thing LED selon notre éditeur graphique :

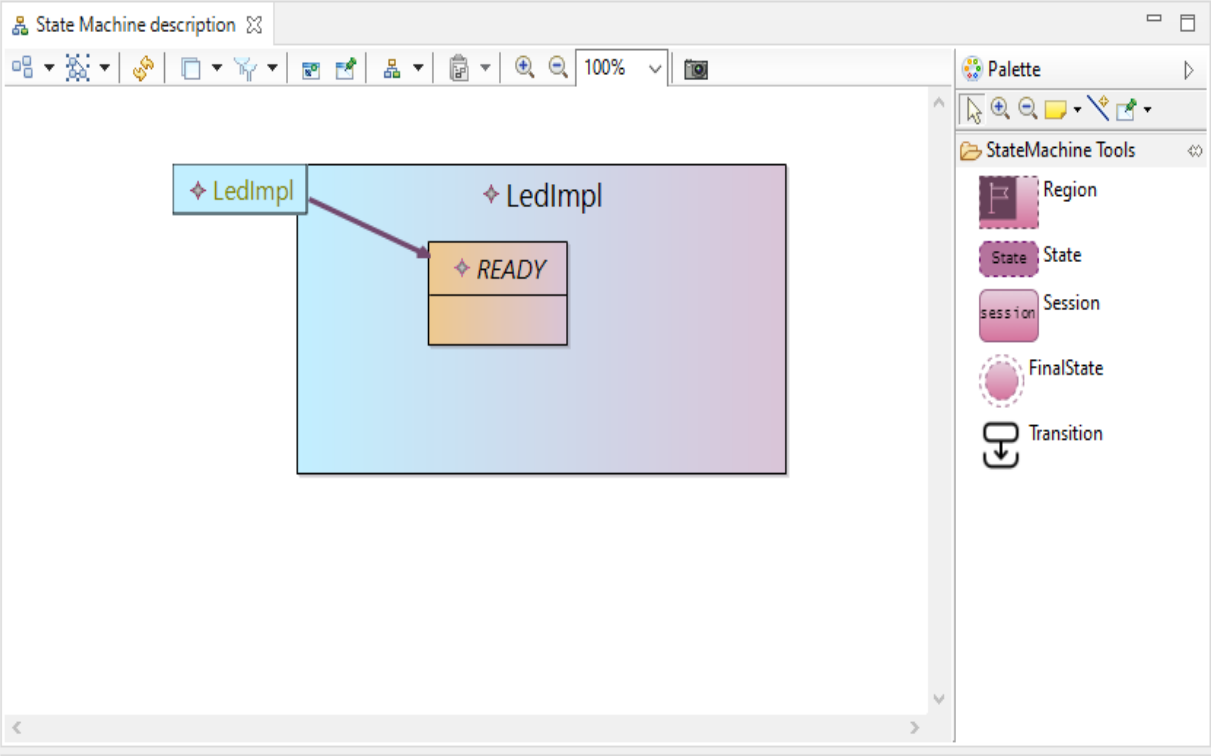

FIGURE 4.13 – un extrait de présentation graphique du détecteur de gaz (partie état machine"ledMsgs")

Le Listing 4.7 décrit le fragment de thing *GasSensorMsgs* avec des messages *read_gas* et *gas_value*. Ils sont utilisés pour mettre en œuvre le comportement des Things *GasSensorMsgs* et *GasDetector*.

Listing 4.7 – La spécification de thing fragmenté GasSensorMsgs

```
thing fragment GasSensorMsgs
{ message read_gas () ;
       message gas_value (val : Integer) ;
}
```

Le Listing 4.8 présente l'implémentation de la deuxième partie de cette application, y compris le fragment de thing *GasSensor*, un port fourni, une fonctions prédéfinies par la carte Arduino *analogRead* et un statechart définissant leur comportement. Le port *gasSensor* permet de recevoir un message *read_gas* et d'envoyer un message *gas_value*. Le diagramme d'état *GasSensorImpl* qui implémente le comportement de thing *GasSensor* comprend un seul état :*Running*. Lorsqu'un message *read_gas* arrive sur le port *gasSensor*. Dans cette transition, le thing *GasSensor* execute la fonction *analogRead* avec le paramètre *PIN* qui permet de lire la valeur de gaz capter et de l'envoyer.

Listing 4.8 – GasSensor.thingml

```
import "_Datatypes.thingml"
thing GasSensor includes GasSensorMsgs
{
    readonly property pin : AnalogPin = AnalogPin:A_3

    provided port gasSensor
    {
```

```
        receives read_gas
        sends gas_value
    }

    function analogRead(pin : AnalogPin) : Integer do
        var val : Integer
        val = ‘analogRead(‘&pin&‘);‘
        return val
     end

    statechart GasSensorImpl init Running
    {
        state Running
        {
                internal event gasSensor?read_gas
                        action gasSensor!gas_value (analogRead(pin))
        }
    }
}
}
```

-Voici la représentation graphique d'état machine de la deuxième partie :

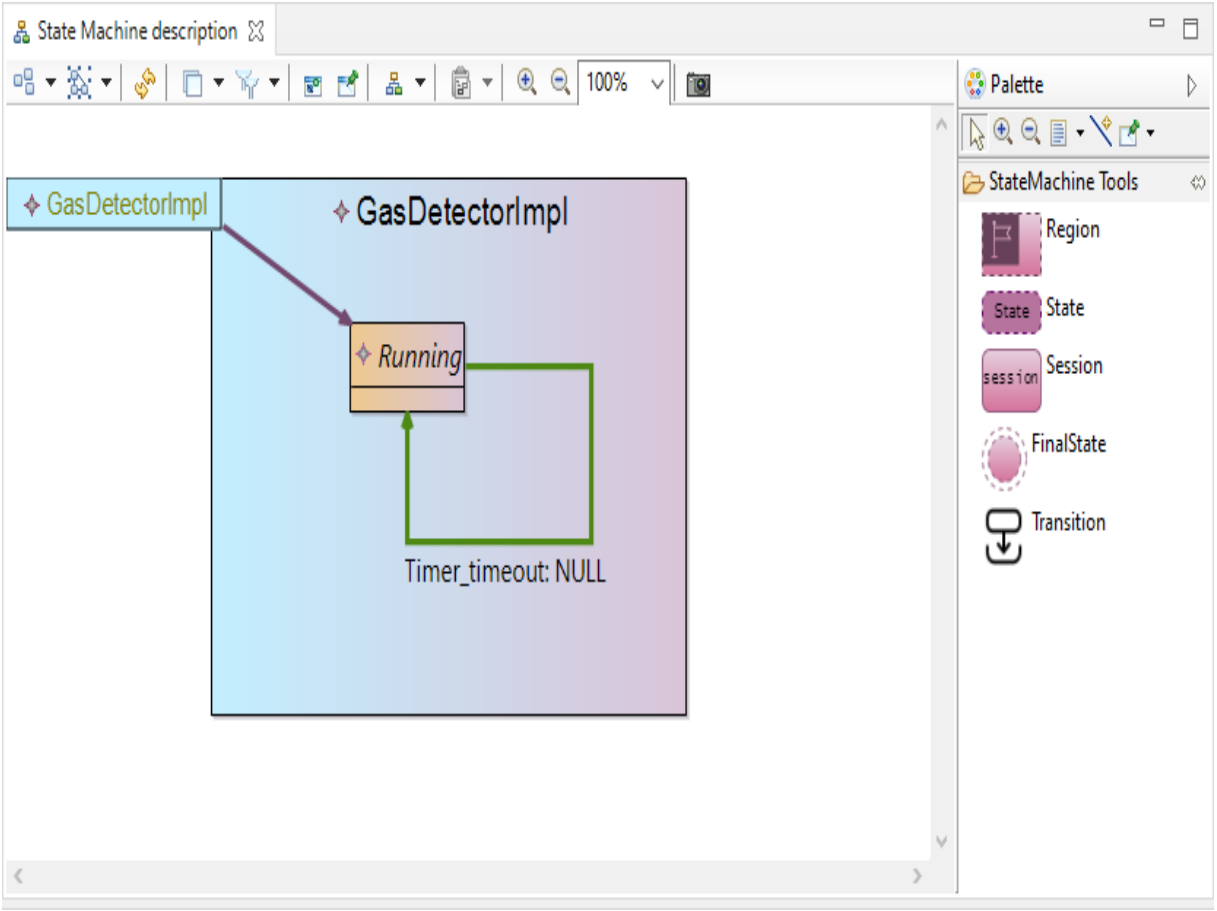

FIGURE 4.14 – un extrait de présentation graphique du détecteur de gaz (partie état machine " GasSensor ")

Cette spécification inclut une instruction spécifique à la plateform Arduino [43] :

— **"AnalogRead()" :** Lit la valeur de la broche analogique spécifiée. Les cartes Arduino contiennent un convertisseur analogique-numérique 10 bits multicanal. Cela signifie qu'il cartographiera les tensions d'entrée entre 0 et la tension de fonctionnement (5 V ou 3,3 V) en valeurs entières.cette fonction a un paramètre "PIN"(le nom de la broche d'entrée analogique à lire).ET un retour de la lecture analogique sur la broche. Avec le type de données : entier.

Le Listing 4.9 présente l'implémentation de la partie principale de l'application détecteur de gaz , y compris le fragment de things TimerMsgs, LEDMsgs et GasSensorMsgs, deux variables locales, trois required ports et un statechart définissant leur comportement. Le diagramme d'état *GasDetectorImpl* comprend un état : Running. Il entre initialement dans l'état de lumière verte, car le message gas_value est donné la valeur 0, il passe à l'état rouge lorsqu'un message gas_value arrive sur le port gasSensor avec la valeur 1 et aussi un message timer timeout arrive sur le port timer.

Listing 4.9 – GasDetector.thingml

```
import "LEDs.thingml"
import "Gas_Sensor.thingml"

thing GasDetector includes TimerMsgs, LEDMsgs, GasSensorMsgs {

    property PIN_Green : UInt8 = 2
    property PIN_Red : UInt8 = 3

        required port timer
        @sync_send "true"
        {
                receives timer_timeout
                sends timer_start
        }

        required port led {
                sends led_ON, led_OFF
        }

    required port gasSensor
    {
                sends read_gas
                receives gas_value
        }

        function setDigitalOutput(pin: UInt8) do
                `pinMode(`&pin&`, OUTPUT);`
        end
statechart GasDetectorImpl init Running {
                on entry do
                   `Serial.begin(0);`
                   setDigitalOutput(PIN_Green)
                   //Led!led_ON(PIN_Green)
                   setDigitalOutput(PIN_Red)
                   //Led!led_OFF(PIN_Red)
                end

                state Running {
                        on entry do
```

```
                        gasSensor!read_gas()
                            timer!timer_start(0, 10)
                    end
                    internal event e : gasSensor?gas_value
                            guard (e.val ==0)
                            action do
                                ‘Serial.println(‘&e.val&‘);‘
                                led!led_ON(PIN_Green)
                                led!led_OFF(PIN_Red)
                            end
                    internal event e : gasSensor?gas_value
                            guard (e.val ==1)
                            action do
                                ‘Serial.println(‘&e.val&‘);‘
                               // ‘Serial.println(analogRead(17));‘
                                led!led_OFF(PIN_Green)
                                led!led_ON(PIN_Red)
                            end
                transition -> Running event timer?timer_timeout
            }
        }
}
protocol Timer;
```

-La figure 4.15 représente graphiquement le code "GasDetector" la partie d'état machine :

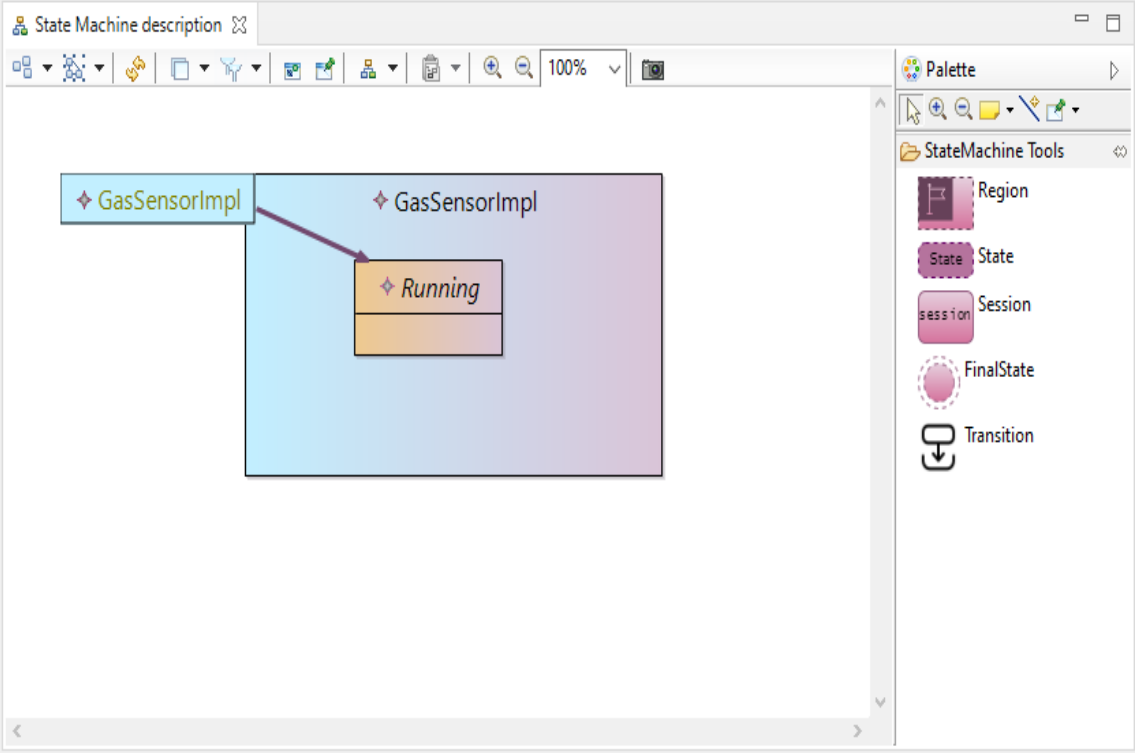

FIGURE 4.15 – un extrait de présentation graphique de détecteur du gaz (partie état machine " GasDetector ")

Finalement, on passe à la partie de configuration. Exactement comme nous l'avons vu dans le modèle précédent, on déclare les instances des choses, il y a trois instances, chaque

instance a un connecteur qui gère la communication entre eux.

Listing 4.10 – Dernière partie GasDetector.thingml

```
configuration Gas_Detector_App
{
        instance gas_detector : GasDetector
        instance led_set : LED
        instance gas_sensor : GasSensor
        connector gas_detector.led => led_set.ctrl
        connector gas_detector.gasSensor => gas_sensor.gasSensor
        connector gas_detector.timer over Timer
}
```

La figure 4.16 représente la partie de la configuration :

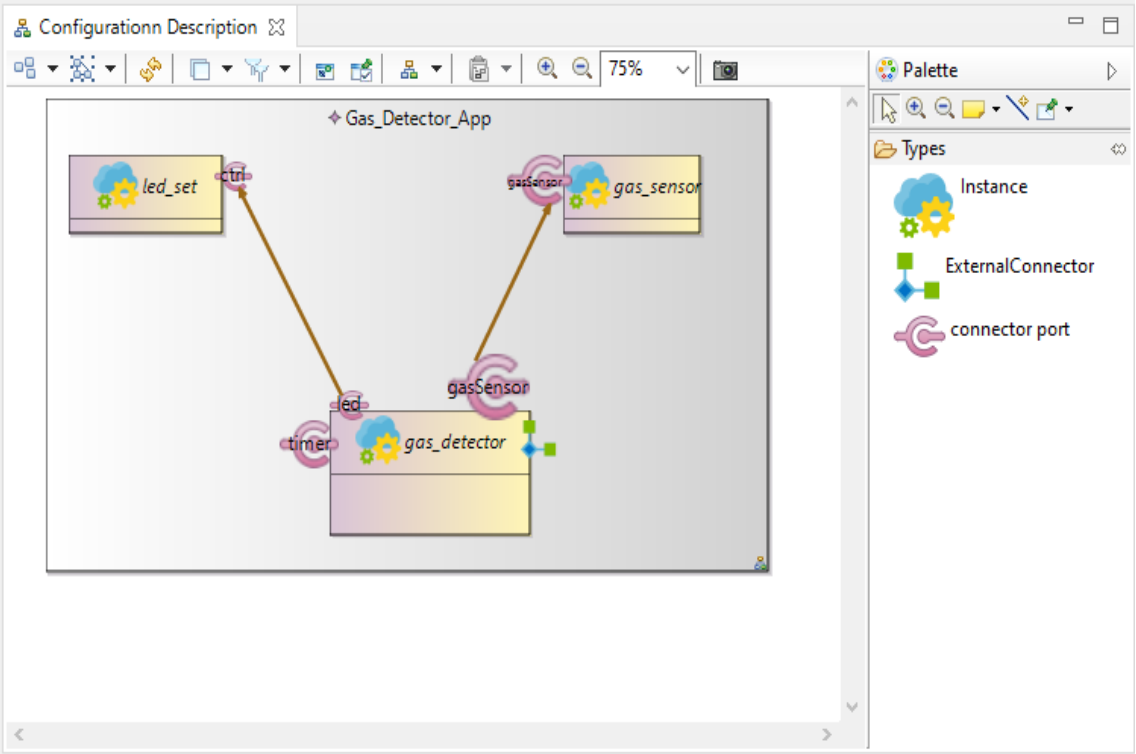

FIGURE 4.16 – un extrait de présentation graphique de détecteur du gaz (partie configuration "GasDetector")

4.3.2 Génération de code

Dans cette section, nous avons répété les mêmes étapes de génération de code ThingML que nous avons suivi précédemment :

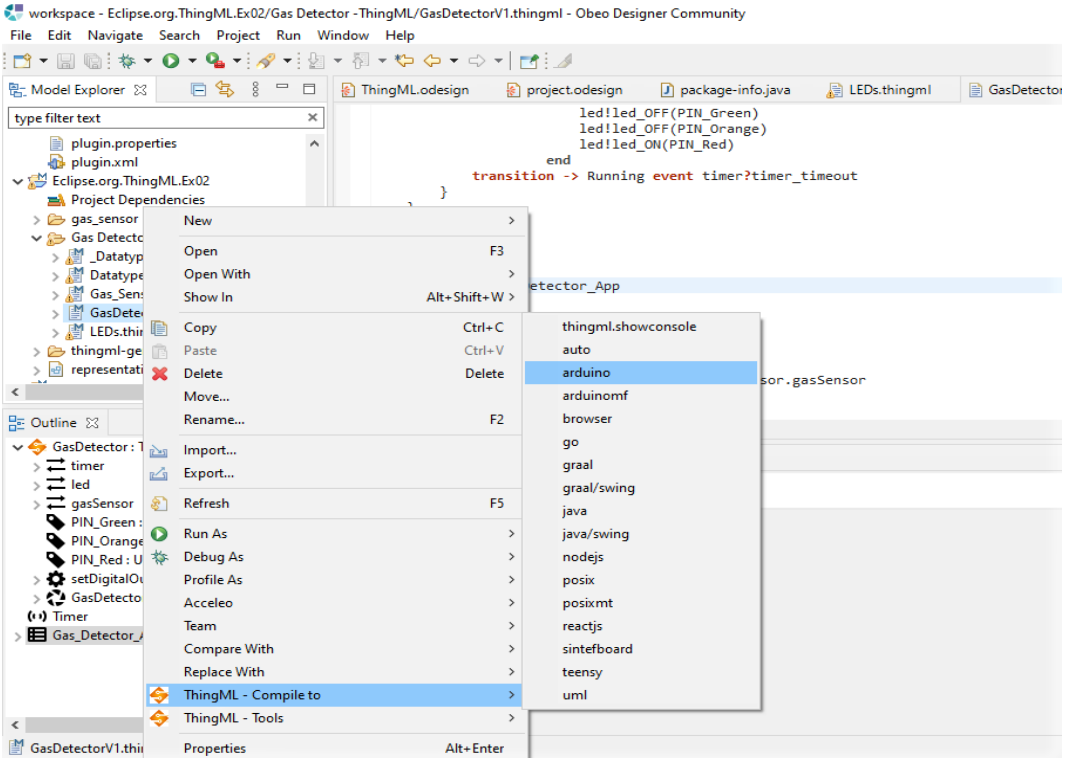

FIGURE 4.17 – Génération de code de détecteur de gaz

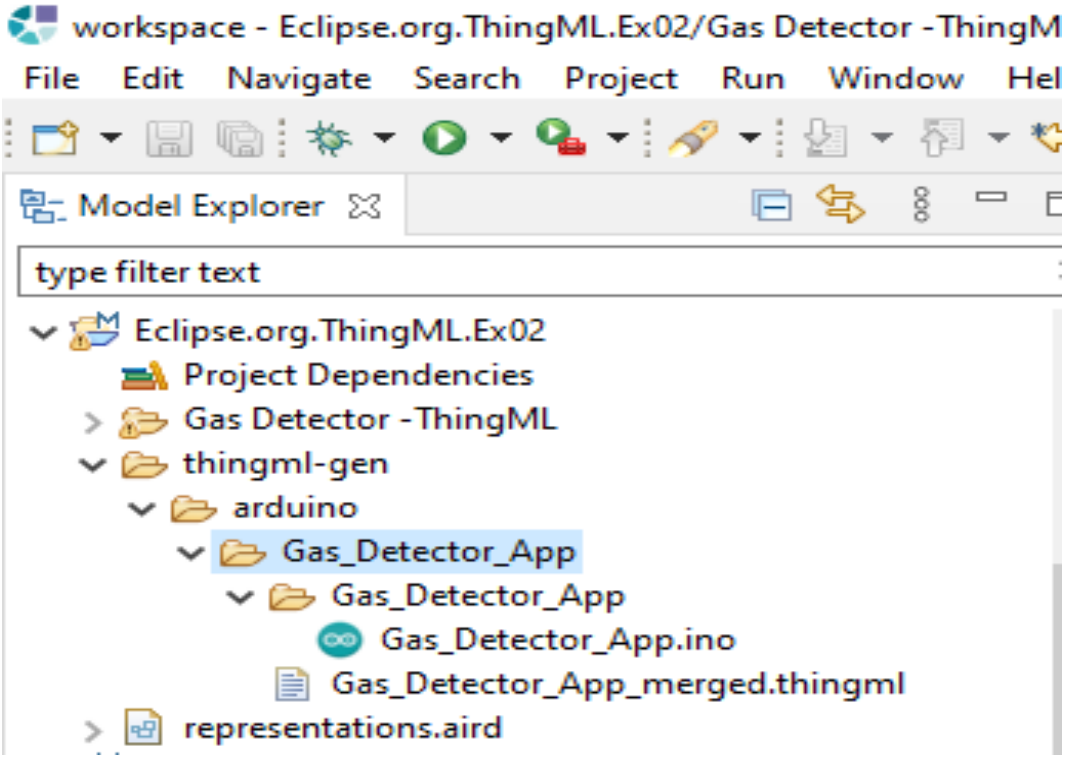

FIGURE 4.18 – la suite de génération de code de détecteur de gaz

4.3.3 Compilation :

Dans cette étape , nous avons répété le même algorithme pour faire la compilation de code sketches de détecteur de gaz par un simple clique sur croquis > Verify/Compile dans

la barre de menus. La Figure 4.19 montre les résultats que vous devriez obtenir si tout a fonctionné correctement ou pas :

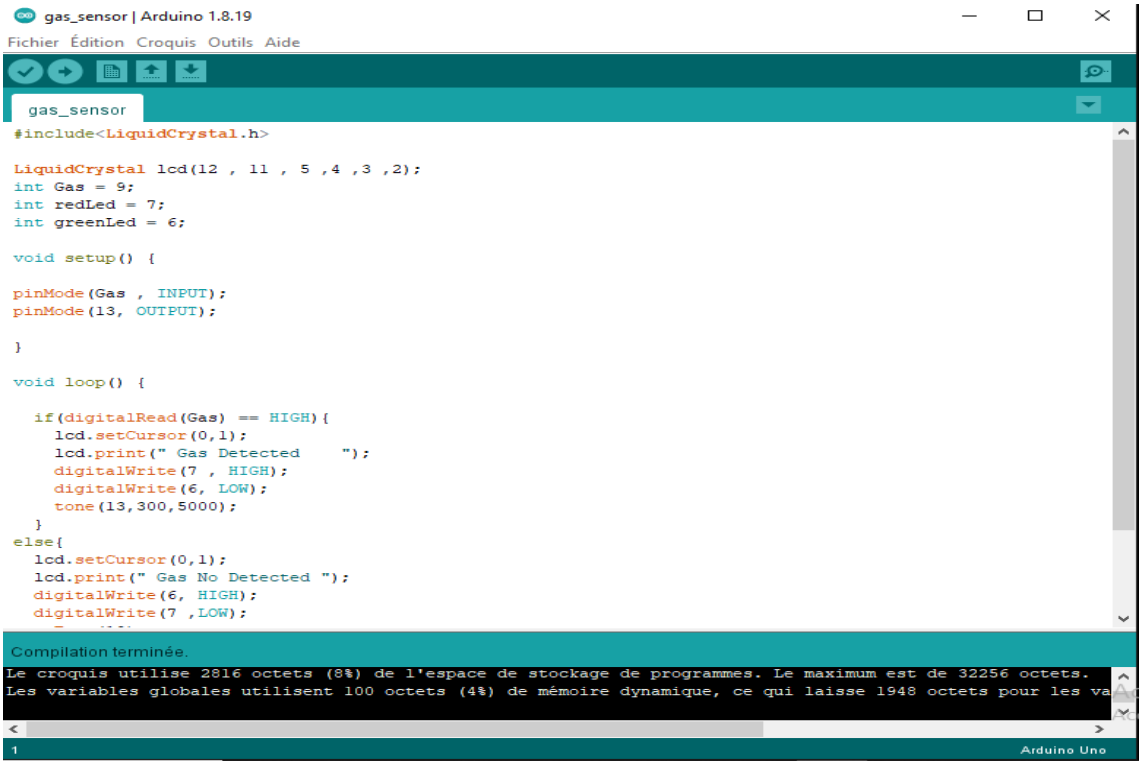

FIGURE 4.19 – Compilation de code de détecteur de gaz

Comme vous remarquez , il n'y a pas de messages d'erreurs alors le code ne contient pas des fautes de frappe qui font échouer le processus de compilation.

4.3.4 Simulation :

La figure 4.20 illustre schématiquement le circuit correspondant et le résultat de simulation sous Proteus en cas de fuite de Gaz :

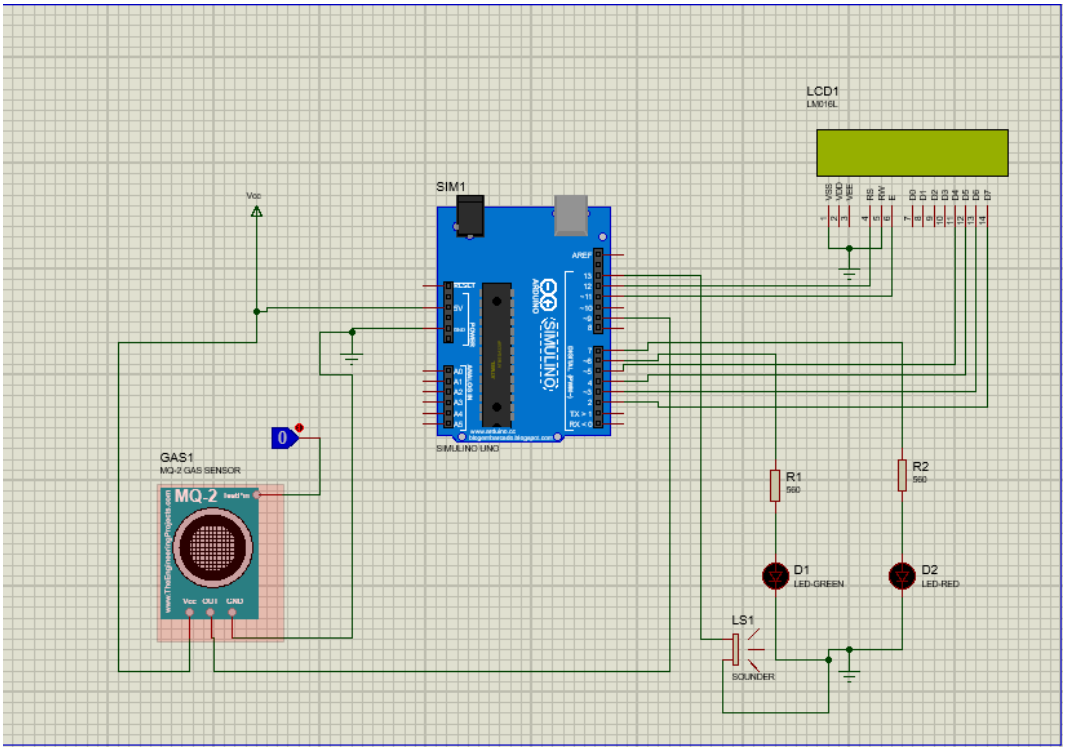

FIGURE 4.20 – le circuit de détecteur de gaz

Nous avons obtenu un fichier objet à l'étape de compilation ce dernier faire une définition de fichier programme du composant Arduino. Nous double-cliquons sur le composant Arduino , et nous choisissons le fichier programme . Enfin, nous cliquons sur l'icône Run simulation dans la barre d'outils de Proteus pour faire la simulation .Nous avons effectué une série de simulations qui démontrent que le code source généré par la génération de code ThingML à partir de notre spécification est exécuté correctement sur le circuit développé dans Proteus . S'il ya une détection d'une fuite de gaz : Le buzzer est allumé rouge et l'afficheur LCD affiche (gaz detected).

4.4 Conclusion

Dans ce chapitre, nous avons entamé notre réalisation pratique sur les deux exemples " feu de circulation" et "détecteur de gaz". Pour cela nous avons essayé de présenter les procédures, les procédure ainsi que les outils utilisés afin d'arriver à notre but.

CONCLUSION GÉNÉRALE

Dans ce mémoire, nous avons mis en œuvre des systèmes embarqués qui répondent à des fonctions principales soit de la domotique ou bien de villes intelligentes à savoir la détection d'une fuite de gaz et le contrôle de la circulation.
Nous avons modélisé de ces applications IoT en se basant sur l'approche ThingML, pour la proposition d'un éditeur graphique-textuel. Nous avons choisi la plateforme Eclipse Modeling Tools en utilisant les technologies EMF, Xtext et siruis intégrés dans cette dernière pour le développement d'un environnement graphique. Nous avons généré le travail sur la palteforme Arduino pour simuler et obtenir des résultats.
Ce projet nous a permis de faire le lien entre l'étude théorique d'une application IOT et sa réalisation pratique dans le but de valider nos connaissances théoriques par la pratique, nous avons appris les compétences suivantes :

- La compréhension de l'approche ThingML et apprendre sa programmation.
- L'utilisation des outils informatiques et électroniques.
- acquérir les bases de la méta modélisation.
- La réalisation pratique de circuit électronique sur un simulateur.

Finalement, nous sommes unanimes pour dire que ce projet nous a permis de nous amuser grâce à la manipulation des logiciels , tout en acquérant de meilleures connaissances des applications IOT, ce qui pourrait nous être fortement utile pour notre vie professionnelle future.Le seul point « négatif », serait sûrement le manque de matériel pour pouvoir encore approfondir ce travail, donc nous souhaitons vivement d'appliquer la simulation des modèles obtenus en réalité avec des résultats visibles et tangibles.

BIBLIOGRAPHIE

[1] Strategyanalytics, Global Connected and IoT Device Forecast Update, [Online]. `https://www.strategyanalytics.com/access-services/devices/connected-home/consumer-electronics/reports/report-detail/global-connected-and-iot-device-forecast-update/` [Accessed Jul-2022].

[2] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things : A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorials*, 17(4) :2347–2376, 2015.

[3] Eleonora Borgia. The internet of things vision : Key features, applications and open issues. *Computer Communications*, 54 :1–31, 2014.

[4] Bruno Costa, Paulo F Pires, and Flávia C Delicato. Towards the adoption of omg standards in the development of soa-based iot systems. *Journal of Systems and Software*, 169 :110720, 2020.

[5] Federico Ciccozzi, Ivica Crnkovic, Davide Di Ruscio, Ivano Malavolta, Patrizio Pelliccione, and Romina Spalazzese. Model-driven engineering for mission-critical iot systems. *IEEE software*, 34(1) :46–53, 2017.

[6] Brice Morin, Nicolas Harrand, and Franck Fleurey. Model-based software engineering to tame the iot jungle. *IEEE Software*, 34(1) :30–36, 2017.

[7] Nicolas Harrand, Franck Fleurey, Brice Morin, and Knut Eilif 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*, pages 125–135, 2016.

[8] Lorenzo Addazi and Federico Ciccozzi. Blended graphical and textual modelling for uml profiles : A proof-of-concept implementation and experiment. *Journal of Systems and Software*, 175 :110912, 2021.

[9] Proteus, [Online]. Available : `https://www.labcenter.com/` [Accessed juin 2022].

[10] Lu Tan and Neng Wang. Future internet : The internet of things. In *2010 3rd international conference on advanced computer theory and engineering (ICACTE)*, volume 5, pages V5–376. IEEE, 2010.

[11] Lu Tan and Neng Wang. Future internet : The internet of things. In *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, volume 5, pages V5–376–V5–380, 2010.

[12] Walid Hadjadj and Meriem Zaiter. L'utilisation de n-version de programmation pour la prise en charge des fautes dans un environnement iot. 2018.

[13] Nallapaneni Manoj Kumar, Karthik Atluri, and Sriteja Palaparthi. Internet of things (iot) in photovoltaic systems. In *2018 National Power Engineering Conference (NPEC)*, pages 1–4. IEEE, 2018.

[14] Mohamed Tahar Hammi. *Sécurisation de l'Internet des objets*. PhD thesis, Université Paris-Saclay (ComUE), 2018.

[15] Priya Suresh, J Vijay Daniel, Velusamy Parthasarathy, and RH Aswathy. A state of the art review on the internet of things (iot) history, technology and fields of deployment. In *2014 International conference on science engineering and management research (ICSEMR)*, pages 1–8. IEEE, 2014.

[16] M. MOHAMED MAHMOUD Othman M. HAOUA Zakaria. Vers des bâtiments intelligents pour l'élevage de volailles. Université Saad Dahlab – Blida, 2019.

[17] Frédeéric Lemoine. *Internet des Objets centré service autocontrôlé*. PhD thesis, Conservatoire national des arts et metiers-CNAM, 2019.

[18] Mr Hidjeb Ali. implémentation d'un protocole d'élection d'un serveur d'authentification dans l'interner des objets. promotion. Université Abderrahmane Mira de Bejaïa, 2017.

[19] militairz. 2019.

[20] Zineddine Kouahla. Plateforme de développement pour l'internet des objets (ido) avec un apprentissage automatique. 2019.

[21] Mayuri A Bhabad and Sudhir T Bagade. Internet of things : architecture, security issues and countermeasures. *International Journal of Computer Applications*, 125(14), 2015.

[22] Shruti G Hegde Soumyalatha. Study of iot : understanding iot architecture, applications, issues and challenges. In *1st International Conference on Innovations in Computing & Net-working (ICICN16), CSE, RRCE. International Journal of Advanced Networking & Applications*, number 478, 2016.

[23] Salim Mahamat Charfadine. *Gestion dynamique et évolutive de régles de sécurité pour l'Internet des Objets*. PhD thesis, Université de Reims Champagne-Ardenne, 2019.

[24] Prachi Jain1 Kancha Jha2 Sanjivani Patwa. Architecture of internet of things (iot).

[25] Bruno Costa, Paulo F Pires, Flávia C Delicato, Wei Li, and Albert Y Zomaya. Design and analysis of IoT applications : A model-driven approach. In *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/-CyberSciTech)*, pages 392–399. IEEE, 2016.

[26] Kleanthis Thramboulidis and Foivos Christoulakis. UML4IoT—A UML-based approach to exploit IoT in cyber-physical manufacturing systems. *Computers in Industry*, 82 :259 – 272, 2016.

[27] Federico Ciccozzi and Romina Spalazzese. MDE4IoT : Supporting the Internet of Things with Model-Driven Engineering. In *International Symposium on Intelligent and Distributed Computing*, pages 67–76. Springer, 2016.

[28] S. Xu, W. Miao, T. Kunz, T. Wei, and M. Chen. Quantitative Analysis of Variation-Aware Internet of Things Designs Using Statistical Model Checking. In *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 274–285, August 2016.

[29] Mohammad Sharaf, Mai Abusair, Rami Eleiwi, Yara Shana'a, Ithar Saleh, and Henry Muccini. Modeling and Code Generation Framework for IoT. In Pau Fonseca i Casas, Maria-Ribera Sancho, and Edel Sherratt, editors, *System Analysis and Modeling. Languages, Methods, and Tools for Industry 4.0*, Lecture Notes in Computer Science, pages 99–115, Cham, 2019. Springer International Publishing.

[30] Imad Berrouyne, Mehdi Adda, Jean-Marie Mottu, Jean-Claude Royer, and Massimo Tisi. CyprIoT : framework for modelling and controlling network-based IoT applications. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 832–841, 2019.

[31] Felicien Ihirwe, Davide Di Ruscio, Silvia Mazzini, and Alfonso Pierantonio. Towards a modeling and analysis environment for industrial IoT systems. *arXiv preprint arXiv :2105.14136*, 2021.

[32] REZALI milila TITI lydia. *Un Environnement Graphique pour la Conception du Comportement de l'Internet des Objets : une Approche basée sur les Processus Métier*. PhD thesis, université Mohamed Seddik Ben Yahia ,JIJEL, 2019-2020.

[33] Ithar Saleh, Rami Ilaiwi, and Yara Shanaa. Mapping from caps software architecture modeling language (saml) to thingml language using acceleo code generator. 2019.

[34] Xtext.[Online]. `https://www.eclipse.org/Xtext/` [Accessed juin 2022].

[35] Eclipse Modelling Framework (EMF), [Online]. Available : `https://www.eclipse.org/modeling/emf/` [Accessed June-2022].

[36] ThingML open-source project.[Online]. `https://github.com/TelluIoT/ThingML` [Accessed juin 2022].

[37] Abdelouahab Fortas, Elhillali Kerkouche, and Allaoua Chaoui. Formal verification of iot applications using rewriting logic : An mde-based approach. *Science of Computer Programming*, page 102859, 2022.

[38] Franck Fleurey and Brice Morin. Thingml : A generative approach to engineer heterogeneous and distributed systems. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 185–188. IEEE, 2017.

[39] HEADS-project, [Online]. Available : `https://github.com/HEADS-project` [Accessed juin 2022].

[40] PingPong, [Online]. Available : `https://github.com/HEADS-project/training/tree/master/1.ThingML_Basics/2.PingPong/` [Accessed juin 2022].

[41] Brice Morin, Nicolas Harrand, and Franck Fleurey. Model-based software engineering to tame the iot jungle. *IEEE Software*, 34(1) :30–36, 2017.

[42] Sirius.[Online]. `https://www.obeosoft.com/fr/produits/eclipse-sirius` [Accessed juin 2022].

[43] Arduino.[Online]. `https://www.arduino.cc/reference/en/language/functions` [Accessed juin 2022].

[44] Xavier Dolques. *Génération de Transformations de Modèles : une approche basée sur les treillis de Galois*. PhD thesis, Université Montpellier II-Sciences et Techniques du Languedoc, 2010.

[45] ATTOU Ismail KAMBOUCHE Sofiane. *Conception et réalisation d'un système d'agriculture intelligent*. PhD thesis, Centre Universitaire Belhadj Bouchaib d'Ain-Temouchent Institut de Technologie Département de Génie Mécanique, 2017/2018.

[46] Justin Cooper and Dimitris Kolovos. Engineering hybrid graphical-textual languages with sirius and xtext : Requirements and challenges. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 322–325. IEEE, 2019.