Rapport de stage
Année universitaire 2020 - 2021
Laboratoire d'Informatique de Grenoble - VASCO

Génération de code à partir de modèles UML-RT ciblant des plateformes Arduino

---

Elian Loraux - 11 juin 2021

LIG :
Yves Ledru : Chef de VASCO
Nicolas Hili : Tuteur de stage

IUT :
Eric Fontenas : Responsable stage
Sara Laget-Kamel : Tutrice IUT

Déclaration de respect des droits d'auteur

Par la présente, je déclare être le seul auteur de ce rapport et assure qu'aucune autre ressource que celles indiquées n'ont été utilisées pour la réalisation de ce travail. Tout emprunt (citation ou référence) littéral ou non à des documents publiés ou inédits est référencé comme tel.
Je suis informé qu'en cas de flagrant délit de fraude, les sanctions prévues dans le règlement des études en cas de fraude aux examens par application du décret 92-657 du 13 juillet 1992 peuvent s'appliquer. Elles seront décidées par la commission disciplinaire de l'UGA.

À Saint Martin d'Hères, le 30 mai 2021.

Signature

Remerciements

Je tiens à exprimer ma reconnaissance à mes tuteurs de stage, Nicolas Hili côté LIG et Sara Laget-Kamel côté IUT.

Carole Pasini, Frédéric Gargaud et Gary Grange ont toute ma gratitude pour leurs conseils avisés et leurs recommandations.

Je souhaite remercier également Laurent Chauvineau pour son aide apportée dans la mise en place de l'outil LaTeX (outils de mise en forme PDF) dont je me suis servi pour rédiger ce rapport.

Pour finir, je remercie le département informatique de l'IUT2 ainsi que le LIG pour leur formation et leurs enseignements.

Sommaire

1 | Abstract

Porting a UML-RT library on Arduino platform
Elian Loraux
Abstract : Nowadays, embedded systems are increasingly used, and they are found in almost all informatic systems. Within the Grenoble computer science laboratory and in the context of a research project on model execution in an embedded system, I had to port an UML-RT library on Arduino platform. Arduino is a programmable prototyping board that tends to be representative of the industrial standards. So, I needed to define UML-RT and its component. To do so, I had to familiarize myself with the UML-RT language and its Papyrus-RT tool which can generate C++ code to create testing models. The most important and hardest task was to modify the UML-RT library so it could execute on Arduino. To succeed, I had to pass many steps. I started by compiling the library with the library compile tool which consisted in modifying the target. Then, I worked hard to link the library to the main program by trying several different tools and technics. After testing several Makefiles (compiling scripts), I used the command line interface Arduino tool which allowed me to fix bugs and to finish, I used the Arduino IDE. The next step was to fix many bugs within the library following modifications to compile the main program. After facing a hardware capacity restriction, I was able to test my alterations and a long phase of currently unfinished debugging had started. To conclude, I will present the potential practical applications and what is actually achievable.
Keywords : UML-RT, Arduino, LIG, embedded system.

2 | Résumé du rapport

Au sein du Laboratoire d'Informatique de Grenoble et dans le cadre d'un projet de recherche sur l'exécution de modèle sur des systèmes embarqués, j'ai dû porter la bibliothèque UML-RT sur une plateforme Arduino. Arduino est un type de carte de prototypage programmable qui tend à être représentatif des cartes utilisées dans les systèmes embarqués. Il me fallait donc définir ce qu'est UML-RT et ce qui gravite autour. Pour cela, j'ai pris en main le langage UML-RT ainsi que son outil Papyrus-RT capable de générer du code C++ dans l'objectif de réaliser un modèle de test. La tâche la plus importante et difficile à accomplir est de modifier la bibliothèque pour l'exécuter sous Arduino. Pour arriver à cela, je suis passé par plusieurs étapes. J'ai commencé par compiler la bibliothèque grâce à l'outil fourni en le modifiant pour coller à la plateforme. Ensuite, j'ai beaucoup travaillé pour lier la bibliothèque au programme principal en essayant plusieurs outils et techniques différents. Dans l'ordre, j'ai testé l'outil en ligne de commande Arduino, puis plusieurs Makefiles (script de compilation) pour finir avec l'IDE Arduino après avoir résolu un bug grâce aux techniques précédentes. Dans l'étape suivante, il a fallu résoudre beaucoup de problèmes liés à la bibliothèque pour pouvoir compiler le programme principal. Après une limitation matérielle, j'ai pu tester mes modifications et s'est ensuivi une longue phase de débogage à l'heure actuelle inachevée. Pour conclure, je présenterai les applications possibles et ce qui est réellement réalisable actuellement.

3 | Introduction

3.1 Laboratoire d'Informatique de Grenoble

Le Laboratoire d'informatique de Grenoble, aussi abrégé LIG, a été fondé le 1 janvier 2007 et est actuellement dirigé par Noël DE PALMA, Sihem AMER YAHIA, Patrick REIGNIER - adjoint à la direction - et Pascale POULET - assistante de Direction -. La recherche au LIG est mené sur trois sites : le campus, Minatec et Montbonnot. Il est partenaire du CNRS (Centre National de la Recherche Scientifique), le l'INRA (Institut National de la Recherche Agronomique), de Grenoble INP (Institut National Polytechnique) et de l'UGA (Université Grenoble Alpes).

Le LIG rassemble près de cinq cents chercheurs, enseignants-chercheurs, doctorants et personnels en support à la recherche comprenant plus de quarante-cinq nationalités. Les travaux au sein du laboratoire sont répartis en cinq axes de recherche :

- Génie des Logiciels et des Systèmes d'Information ;
- Méthodes Formelles, Modèles et Langages ;
- Systèmes Intelligents pour les Données, les Connaissances et les Humains ;
- Systèmes Interactifs et Cognitifs ;
- Systèmes Répartis, Calcul Parallèle et Réseaux.

Chaque axe a son objectif et est composé de plusieurs équipes de recherche.

L'axe **Génie des Logiciels et des Systèmes d'Information** a pour objectif de proposer des méthodes et des outils logiciels pour aider à gérer le cycle complet du développement d'un logiciel. Il est organisé en trois équipes : CTRL-A, SIGMA et VASCO dont je fais partie.

Les équipes de l'axe **Méthodes Formelles, Modèles et Langages** étudient les concepts, formalismes, techniques et outils permettant la description, l'analyse et le raisonnement sur des systèmes complexes, afin d'obtenir des systèmes à la fois plus riches en fonctionnalités, plus robustes, plus sûrs et plus efficaces. CAPP, CONVECS, SPADES et TYREX sont les quatre équipes de l'axe.

L'axe **Systèmes Intelligents pour les Données, les Connaissances et les Humains** regroupe six équipes autour d'une thématique commune, celle de la production de connaissances sémantiquement riches à partir de données brutes, potentiellement massives, hétérogènes, imparfaites et/ou peu structurées. APTIKAL, GETALP, MOEX, MRIM, SLIDE et STEAMER sont les équipes qui le composent.

L'axe **Systèmes Interactifs et Cognitifs** étudient les interactions entre les Hommes et les Machines, que ce soit grâce à des interfaces, des robots ou de l'Intelligence artificielle*. Il est composé d'IIHM, MARVIN, METAH et M-PSI.

Les équipes de l'axe **Systèmes Répartis, Calcul Parallèle et Réseaux** travaillent sur le traitement de données dans des proportions massives. Il est composé de CORSE, DATAMOVE, DRAKKAR, ERODS et POLARIS.

Les recherches de l'équipe VASCO "VAlidation de Systèmes Composants et Objets logiciels" portent sur la modélisation, la vérification et la validation des systèmes. Sous la responsabilité d'Yves LEDRU, elle compte six enseignants-chercheurs, deux doctorants et douze stagiaires. L'équipe VASCO a la conviction que les outils de génie logiciel doivent être articulés autour des modèles. Son activité principale est la validation, tant des logiciels que des modèles, avec un intérêt particulier pour la validation de la sécurité des systèmes informatiques.

3.2 Sujet de stage

Le sujet du stage est de générer du code à partir de modèles UML-RT ciblant des plateformes Arduino* (carte de prototypage). L'objectif est de pouvoir utiliser des modèles pour contrôler des système embarqués* (système électronique et informatique autonome). L'utilité d'Arduino* est de proposer une carte programmable de prototypage pour des systèmes embarqués. Elle tend à être représentative des cartes utilisées en industrie.
Pour porter le code généré à l'aide de Papyrus-RT plusieurs étapes sont à franchir :

- Me familiariser avec UML-RT et créer un modèle de test ;
- Compiler la bibliothèque ciblant Arduino* ;
- Porter le code principal sous Arduino* ;
- Tester le système avec plusieurs modèles.

Une fois ces objectifs atteints, selon le temps qu'il me reste, je peux réaliser plusieurs tâches comme la mise au propre de la bibliothèque modifiée, l'exécution de modèle UML-RT pour un rover ou pour un robot d'auto équilibrage contrôlé par Arduino*.

Mon stage entre dans le cadre du projet de recherche "in-Place DSL Execution for Embedded Robotic Systems" (iPEERS) porté par l'équipe VASCO et en collaboration avec l'université Queen's au Canada. L'objectif est d'exécuter le modèle plutôt que de générer du code pour un langage de programmation à partir dudit modèle. Le problème de générer du code à partir du modèle est que deux langages n'ont quasiment jamais la même sémantique. Il faut donc un générateur par langage sauf pour certains cas particuliers ce qui représente un coût de réalisation et de maintien très élevé.
Deux approches sont possibles :

- l'une est de définir cette sémantique et de pouvoir exécuter des modèles sur des cibles robotiques. Cependant, ce n'est pas aussi simple car les outils existants sont loin d'être adaptés pour être exécutés sur des cartes embarquées*, d'où la nécessité de chercher une solution pour des environnements plus légers ;
- l'autre approche consiste à mélanger les travaux de génération de code et d'exécution de modèles pour être capable de construire automatiquement des générateurs de code. Cela permettrait de supporter une grande partie des langages en réduisant les risques d'erreurs.

Trois stages s'articulent autour de ce projet. Ceux de Stéphane Humblot, Hugo Bantignies et le mien.

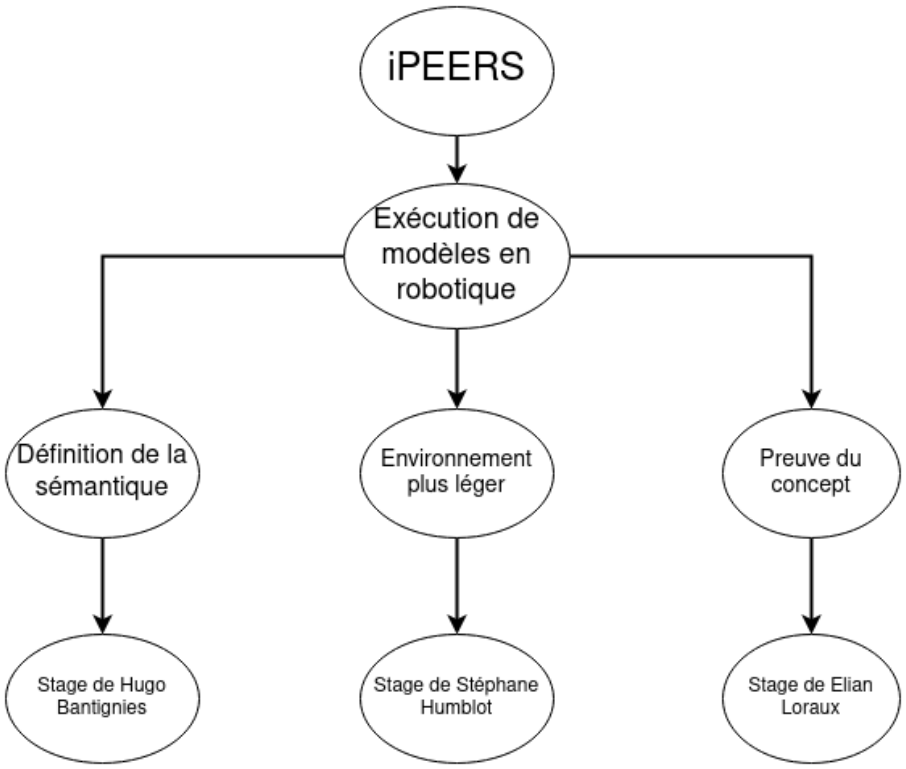

FIGURE 3.1 – iPEERS - articulation des stages.

Mon stage est d'une durée de douze semaines, du 19 avril 2021 au 9 juillet 2021. Avec la période sanitaire actuelle, je me rends au sein de l'IMAG (institut d'Informatique et Mathématiques Appliquées de Grenoble) tous les matins et suis en télétravail tous les après-midi.

3.3 Plan

La présentation de mon travail va s'axer sur trois grandes parties. En premier lieu, je traiterai les modèles UML-RT et Papyrus-RT, leurs applications, avantages et inconvénients. Ensuite, j'aborderai la bibliothèque UML-RT et les modifications que j'ai effectuées pour la porter sous Arduino*. Je finirai par le portage du code généré par Papyrus-RT sous Arduino* et les différentes techniques utilisées.

4 | Modèle UML-RT et Papyrus-RT

4.1 Modèle UML-RT

4.1.1 Modèle

"The entire history of software engineering is about raising the level of abstraction" [1] (Grady Booch, l'un des fondateurs de l'UML). Ce qui signifie que depuis le toujours, l'ingénierie logicielle charche à s'extraire le plus possible de l'environnement d'exécution. Cela se retrouve, aujourd'hui encore, avec par exemple la création du C, pourtant apparu au debut des années 70,et majoritairement utilisé pour les sytèmes d'exploitation. Le C viens "remplacer" l'assembleur en s'extrayant du jeu d'instruction spécifique à chaque processeur. Plus récemment, ce principe se répète avec l'explosion des applications web comme google doc, next cloud et bien d'autres, utilisables via navigateur internet, qui peuvent être lancées sur quasiment tous les appareils (ordinateur, smart phone, tablette, montre connectée, etc.). On note aussi en ce moment une montée en puissance de ce qu'on appelle le "no code". C'est le principe de réaliser des automatisations, voire des applications, en faisant de l'assemblage de blocs d'instruction(s) simple(s) (ex : "si... alors... sinon..."). On peut citer Scratch, IFTTT, WIX ou même Apple Shortcuts qui sont des applis de "no code" très répandues et connues dans des domaines bien différents (jeux vidéos, sites web, automatisations, etc.).

Avant d'expliquer en quoi consiste les modèles UML-RT, il est nécessaire de définir ce qu'est un modèle dans le milieu informatique : c'est une représentation schématique d'un projet ou d'un élement de projet. Il a pour but l'abstraction, l'automatisation et l'analyse.

L'abstraction : le modèle permet de communiquer avec les équipes ou de véhiculer une information plus simplement. L'expression "Une image vaut mille mots" prend tout son sens ici.

L'automatisation : l'objectif est, à partir d'une modélisation (la plus souvent graphique ou mixte graphique/code), de pouvoir générer un programme entier sans erreur. En effet, générer du code évite de nombreuses erreurs qu'on pourrait faire en écrivant du code à la main et donc de gagner du temps.

L'analyse : le fait d'avoir des schématisations abstraites permet d'analyser le projet. Nous pourrions être capables de détecter ainsi avec certains types de modèles des interblocages, des boucles infinies ou des pertes de messages. La modélisation permet aussi des analyses d'efficacité (Combien de fois cette transition est-elle franchie ? Quel est le temps de réponse pour réaliser une fonction ?).

Un modèle a son utilité s'il remplit au moins l'un des trois points précédents.

La modélisation en début de projet est devenue quasiment obligatoire. Elle permet de définir à l'avance les cas et solutions et ainsi maintenir une certaine cohérence tout au long de la réalisation du projet. La modélisation dans son sens premier peut prendre n'importe quelle forme. Cependant certaines normes se sont largement imposées comme l'UML et ses dérivés (SysML, UML-RT etc.).

1. L'histoire entière de l'ingénierie logiciel est d'augmenter le niveau d'abstraction

4.1.2 UML

L'UML (Unified Modeling Language) [2] est donc le langage de modélisation de plus répandu aujourd'hui. Il a été créé en 1997, principalement issu de l'unification (d'où le unified d'UML) des travaux de Grady Booch, James Rumbaugh et Ivar Jacobson. La version actuelle, UML 2.5, propose quatorze types de diagrammes dont sept structurels et sept comportementaux. À titre de comparaison, UML 1.3 comportait vingt-cinq types de diagrammes. L'UML est à présent un standard adopté et édité par l'Object Management Group (OMG), consortium international créé en 1989 dont le but, non-lucratif, est de standardiser et promouvoir la programmation orientée objet (POO). Cependant, l'UML reste un langage générique et n'est pas exclusif à la POO. Parmi les diagrammes principaux on retrouve :

Le **diagramme de classe** : presque indispensable pour la POO, il sert à schématiser une classe, ses méthodes, attributs et relations avec les autres classes. En reprenant l'exemple d'une école, on peut définir une classe mère *Personne* avec deux classes filles *Professeur* et *Élève*. Les deux classes filles hériteront alors des attributs et fonctions de la classe mère.
Le diagramme de classe serait donc :

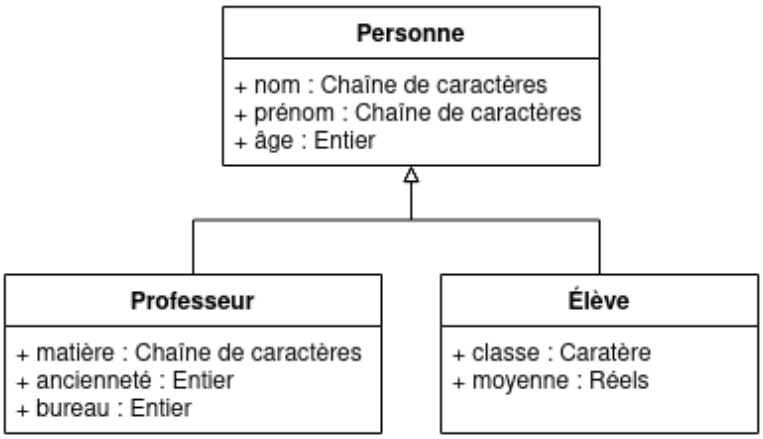

FIGURE 4.1 – Diagramme de classe - Professeur et élève.

Le **diagramme de machine à états** : il permet de mettre en évidence différents états du système et les transitions entre ceux-ci. On peut imaginer qu'un professeur a deux états. L'un "En cours" qui représente les temps où il enseigne devant ses élèves et l'autre "En pause" qui représente les temps d'intercours. Le diagramme de machine à état du professeur donne donc :

FIGURE 4.2 – Diagramme de machine à états - État du professeur.

Le diagramme de machine à états est très important pour UML-RT et générer du code via Papyrus-rt également.

2. Langage de modélisation unifié

4.1.3 UML-RT

L'UML-RT (UML-Real Time)[3] est une variante de l'UML. L'objectif est de créer des systèmes réactifs qui interagissent avec leur environnement. À sa base, un modèle UML-RT est constitué de capsules. Chacune d'elles contient une machine à états utilisée pour représenter son comportement. Les transitions entre les états sont déclenchées par des messages reçus sur les ports de la capsule.

Prenons l'exemple du premier tutoriel de Papyrus-RT réalisant un ping-pong entre deux capsules. Le but est de créer un dialogue entre le "Pinger" et le "Ponger". Le résultat de l'exécution du code généré se trouve en annexe, figure 1. Pour cela, nous avons donc deux capsules, "Pinger" et "Ponger". Le Pinger va envoyer un "ping" au Ponger qui, une fois le "ping" reçu, va renvoyer un "pong" au Pinger, et ainsi de suite une dizaine de fois.

Ici, chaque capsule n'a qu'un seul état, un état "playing", lorsque l'une d'entre elle, par exemple le Ponger, reçoit un ping, il va répondre par un pong puis se remettre en état "playing".

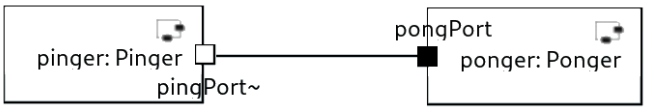

FIGURE 4.3 – Capsule Pinger et Ponger.

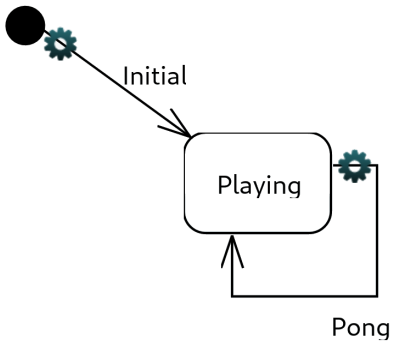

FIGURE 4.4 – Machine à états du Pinger.

On peut résumer le fonctionnement du ping-pong avec le diagramme de sequence suivant :

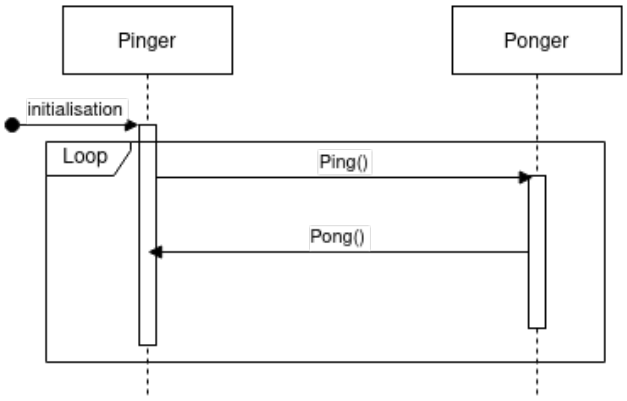

FIGURE 4.5 – Diagramme de séquence - Ping-pong.

3. UML en temps réel

4.2 Papyrus-RT

Papyrus-RT est un outil de modélisation UML-RT basé sur Papyrus/Eclipse spécifique à l'UML-RT. Il est développé principalement par Zeligsoft de 2014 à juin 2017 pour sa dernière version. Papyrus-RT a pour objectif d'être une alternative opensource [4] à RSARTE qui est aussi un outil de modélisation UML-RT, mais propriétaire à IBM. Le projet était soutenu par l'université Queen's, Ericsson et le CEA List. Il n'est actuellement plus développé principalement à cause de l'arrêt des financements. Tout son interêt est de pouvoir générer du code C++ grâce aux modèles. Un programme généré par Papyrus-RT repose sur une partie dynamique liée aux modèles et une partie statique définie par une bibliothèque. La partie dynamique appelle la bibliothèque pour fonctionner convenablement. Avant que le projet ne soit stoppé, Ericsson envisageait d'utiliser Papyrus-RT et UML-RT pour ses systèmes de télécommunication et notamment pour la 5G.

L'interface de Papyrus-RT est simple à prendre en main sans être pour autant avare de fonctionnalités.

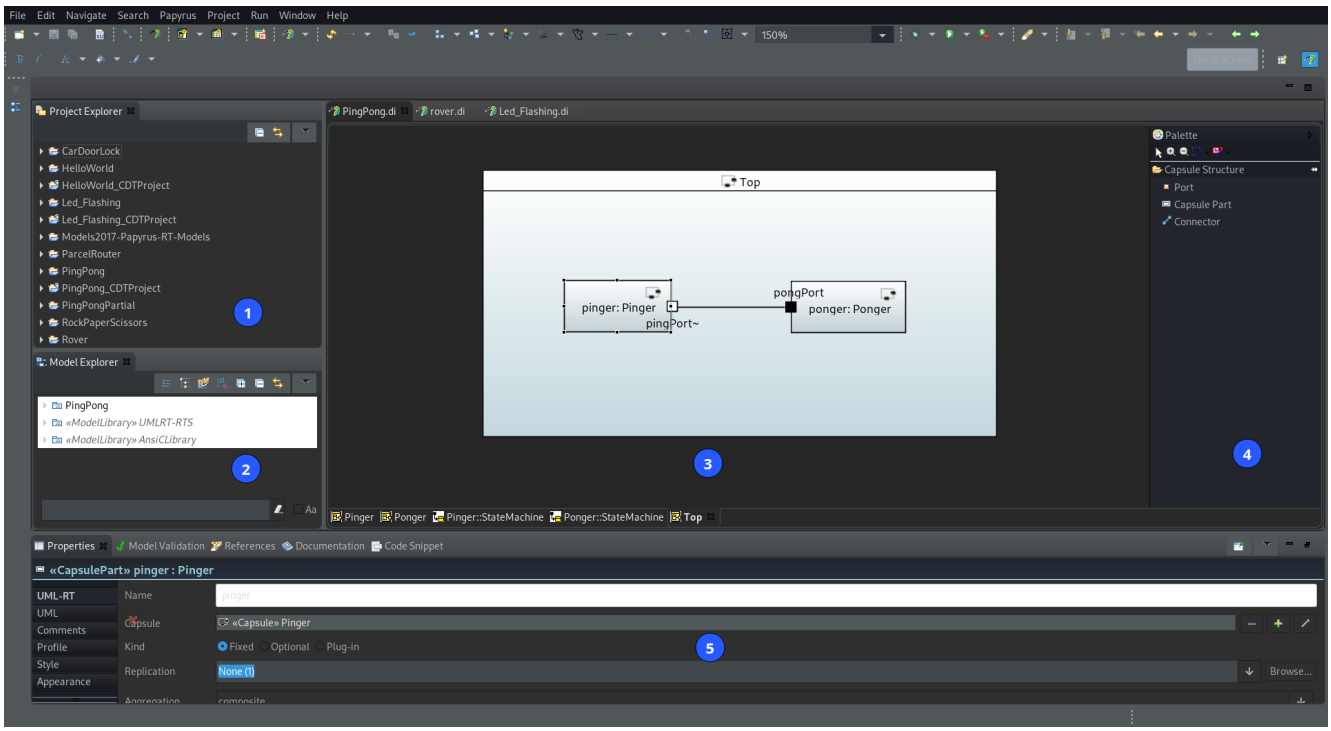

FIGURE 4.6 – Interface de Papyrus-RT.

[1] Correspond à l'explorateur de projet, ce menu permet de naviguer aisément entre les différents projets.
[2] Permet de sélectionner les différents éléments du projet. Il permet de voir et d'éditer les capsules, protocoles et autres, mais aussi de voir l'arborescence. On peut donc choisir une machine à états dans une capsule.
[3] L'espace principal permet de visualiser l'élément sélectionné et de l'éditer.
[4] La palette est là où l'on peut ajouter un port, une connexion ou une capsule à l'élément dans l'espace principal de travail [3].
[5] C'est là où l'on règle les propriétés de l'élément sélectionné. Le menu est spécifique à chaque type d'élément.

4. Source libre (le code source est en libre acces).

5 | Réalisation

5.1 Modèle de test

En tout premier lieu, j'ai dû me familiariser avec UML-RT et Papyrus-RT. Pour cela j'ai réalisé le premier tutoriel de Papyrus-RT sur le ping-pong expliqué plus haut.

L'objectif de mon travail est de réussir à porter le code généré par Papyrrus-RT pour Arduino*. Pour cela il me faut un système de test. J'ai choisi d'allumer une led avec un bouton poussoir. Le cas est intéressant car je dois gérer à la fois une entrée (le bouton) et une sortie (la led). Je me suis donc lancé dans la réalisation d'un modèle UML-RT.

Pour gérer mon système, j'ai décidé de créer deux capsules. Une de contrôle qui va détecter l'appui de bouton ou non et une d'interactions qui va s'occuper de l'allumage de la led. Cela me permet de contrôler plus facilement à la fois l'entrée et la sortie.

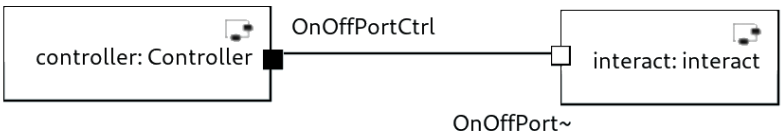

FIGURE 5.1 – Capsule pour le fonctionnement de la led.

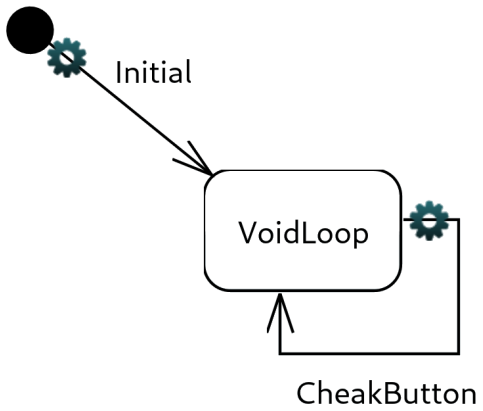

FIGURE 5.2 – Machine à états de "controller".

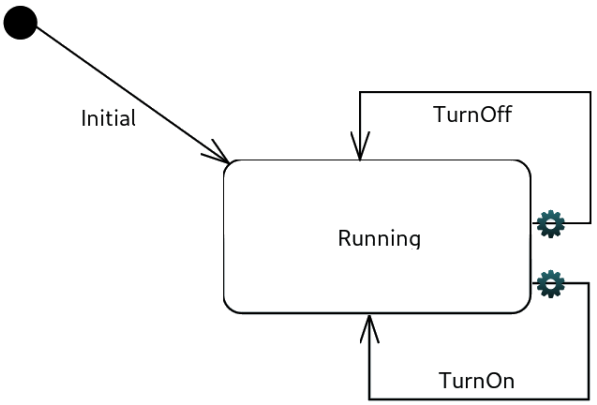

FIGURE 5.3 – Machine à états de "interact".

Le “controller“ va lire l’entrée du bouton sur la pin 3, si le bouton est enfoncé, la fonction turnOn() de “interact“ est appelée, si le bouton est relâché c’est la fonction turnOff() qui est appelée.

```
1 if (digitalRead(3) == HIGH){
2   OnOffPortCtrl.TurnOn().send();
3 }
4 else {
5   OnOffPortCtrl.TurnOff().send();
6 }
```

Programme 5.1 – fonction - CheakButton

En fonction de l’état du bouton, on passe l’état de la led à “HIGH“ ou “LOW“, s’entend allumé ou éteint.

```
1 digitalWrite(2,HIGH);
2 OnOffPort.CheakButton().send();
```

Programme 5.2 – fonction - TurnOn

```
1 digitalWrite(2,LOW);
2 OnOffPort.CheakButton().send();
```

Programme 5.3 – fonction - TurnOff

5.2 Bibliothèque UML-RT

La génération de code par Papyrus-RT s'appuie donc sur la bibliothèque UML-RT. La première chose que j'ai essayée est de l'inclure via l'IDE [1] Arduino*. J'ai donc pris les fichiers d'en-tête et de corps pour les mettre dans le répertoire de bibliothèque Arduino*. Cependant, lors de la compilation, des erreurs de type "référence indéfinie" ont été levées (que ce soit via le dossier de bibliothèque ou via le fichier "local" en donnant le chemin des fichiers). Une référence indéfinie apparaît lorsque, durant le processus d'édition de lien, la fonction n'est pas trouvée. Cela vient généralement soit du fait qu'une fonction est déclarée, mais non définie, soit du fait qu'il y a un problème avec la commande d'édition de lien. Les fonctions en question étant bien déclarées et définies, j'ai donc décidé d'essayer de lier manuellement la bibliothèque au programme principal. Pour cela, il me fallait la recompiler pour AVR-8bit.

La bibliothèque est prévue pour tourner sur un système d'exploitation avec toutes les fonctionnalités qu'on lui connaît : Thread [2] (permet de lancer simultanément plusieurs tâches), Mutex (protection de la concurrence des thread), signaux (forme de communication entre processus), sont bien spécifiques à l'OS [3] ce qui veut dire indisponibles sur des plateformes qui en sont dépourvues comme Arduino*. Un grand ménage s'est dont imposé. Le cœur de la bibliothèque est axé autour de trois dossiers, les en-têtes (.hh), les corps (.cc) et les codes spécifiques à l'OS disponibles en deux versions, une pour Linux et une pour Windows. Les codes spécifiques à l'OS sont des corps des classes gérant Thread, Mutex et signaux. La première étape a donc été de modifier le Makefile fourni pour s'adapter à Arduino*. Pour cela, j'ai dû changer le compilateur, passant de g++ à avr-g++ et le support cible passant de Linux à Arduino*. Le support cible permet au Makefile de savoir quel OS il doit utiliser donc quel code relatif à celui-ci.

J'ai dû rajouter un fichier "build tools" [4] spécifique à avr-gcc qui vient définir certaines variables comme le compilateur, l'éditeur de liens ou l'archiveur. Le fichier de "build tool" est en annexe page 1, programme 1.

Les premières erreurs que auxquelles j'ai été confronté sont des "overflow" [5] relatifs au processeur utilisé. Un ordinateur actuel a un processeur 64 bit (ou 32 bit pour les ordinateurs vraiment anciens) alors que les Arduino* utilisent des architectures de processeur 8 bit. Un bit est la plus petite information possible en informatique. Souvent représenté sous la forme de 0 ou 1, il indique un état vrai ou faux. En regroupant les bits par paquet, nous pouvons encoder des informations plus riches. Pour prendre un exemple, je cherche à avoir un "long" (un entier codé sur 4 octets soit d'une valeur max de plus de 4 milliards) tel que "long SECONDS_PER_DAY = (60 * 60 * 24)". 60 et 24 sont considérés implicitement comme un "int" [6] or, en 8 bit, un "int" est codé sur deux octet, soit une valeur maximum de 65 535. Lors de la compilation, le système va faire le calcul en "int" avant de le convertir en "long". Or, le résultat de la multiplication dépasse la valeur maximale d'un "int". Pour résoudre ce problème, soit j'entre directement le résultat de la multiplication, soit j'explicite le type "long" des 60 et du 24 en faisant "60L * 60L * 24L".

1. Integrated Development Environment (Environnement de développement)
2. Fils d'exécution
3. Operating système (Système d'exploitation)
4. outils de constructions.
5. Débordement (quand l'on va au-delà de la valeur maximale codable)
6. Integer (Entier)

Ensuite, j'ai eu des problèmes de bibliothèques standards (appelées std). Certaines std sont manquantes sous Arduino* et d'autres sont légèrement différentes. C'est le cas de la std que j'ai dû remplacer par la bibliothèque spécifique à Arduino*. J'ai aussi enlevé toutes les inclusions vers le code spécifique à l'OS et tout ce qui s'ensuit (variables, fonctions, etc.).

Après ces modifications, j'ai pu compiler la bibliothèque pour AVR 8-bit ce qui m'a donné un .a (fichier compilé de la bibliothèque à relier au code principal). C'est donc mon point suivant.

5.3 Compilation du programme principal

Après avoir compilé la bibliothèque, il me faut la lier à mon programme principal. Pour cela, j'ai deux solutions : l'outil de compilation Arduino* (arduino-builder) ou faire un Makefile avec toutes les commandes de compilation. J'ai commencé par utiliser arduino-builder utilisable en CLI [7]. Pour me servir de "arduino-builder", j'ai réalisé un petit Makefile qui me permet d'inclure des variables.

```
1 ARDUINO=/usr/share/arduino
2 CART=archlinux-arduino:avr:uno
3 UML_ARDUINO=/home/breizh/Documents/Cours/stage/Sujet/umlrt-arduino
4
5 TopMainArduinoBuilder:
6   arduino-builder -compile -verbose -hardware $(ARDUINO)/hardware -tools
        $(ARDUINO)/tools-builder -libraries $(UML_ARDUINO) -fqbn=$(CART) -
      build-path build -warnings default TopMain.ino
```

Programme 5.4 – Makefile - arduino-builder

Après avoir essayé de donner à l'option "-libraries" le chemin du répertoire de la bibliothèque, de la bibliothèque compilée (.a) et du répertoire de la bibliothèque compilée, le résultat est similaire. Dans tous les cas il y a des références indéfinies vers les fonctions de la bibliothèque.

J'ai donc changé de stratégie et commencé un Makefile pour pouvoir compiler le programme principal et surtout le relier à la bibliothèque.

Mon premier Makefile était assez simple. Il n'utilisait aucune règle ni paramètre par défaut et je n'utilisais pas la bonne option pour lier ma bibliothèque. Associées à l'édition de lien, il existe trois options :

- -I"chemin" spécifie un répertoire pour les "includes" [8] ;
- -L"chemin" spécifie un répertoire pour les bibliothèques (.a) ;
- -l"nom de la bibliothèque" spécifie une bibliothèque à lier au programme.

Avec l'option -l, si le nom de fichier contient"lib", ce préfixe est ignorée par le Makefile. Il faut donc enlever "lib" quand on utilise -l. Dans mon cas le ficher compilé de la bibliothèque s'appelle "librts" donc mon option -l serait "-lrts".

J'en suis donc arrivé au Makefile suivant :

7. Commande Line Interface (interface en ligne de commande)
8. inclusions

```
1  CXX = avr-g++
2
3  CXXFLAGS = -Wall -Wextra
4  CXXFLAGS += -std=c++17
5
6
7  # ------ les bibliotheques exterieures
8
9  AVR_ARDUINO=/usr/share/arduino/hardware/archlinux-arduino/avr
10
11 CPPFLAGS += -I$(AVR_ARDUINO)/cores/arduino
12 CPPFLAGS += -I$(AVR_ARDUINO)/variants/standard
13
14 UML_ARDUINO = /home/breizh/Documents/Cours/stage/Sujet/uml-rt-port-for-
       arduino
15
16 CPPFLAGS += -I$(UML_ARDUINO)/umlrt-arduino_lib/include
17 LDLIBS   += -L$(UML_ARDUINO)/umlrt-arduino_lib/lib
18 LDLIBS   += -lrts
19
20 # ------- structure du projet ---------------------------
21
22 MAIN = TopMain.cc
23 SOURCES = OnOff.cc interact.cc Controller.cc Top.cc TopControllers.cc
24
25 OBJ  = $(SOURCES:.cc=.o)
26 OBJ += $(MAIN:.cc=.o)
27
28 # -------- dependances --------------------------------
29
30 all: TopMain
31
32 TopMain: $(OBJ)
33     $(LINK.cc) -o $@ $^
34
35 TopMain.o:         TopMain.cc TopControllers.hh
36 TopControllers.o: TopControllers.cc Top.hh Controller.hh interact.hh
37 Top.o:            Top.cc Controller.hh interact.hh
38 Controller.o:     Controller.cc OnOff.hh
39 interact.o:       interact.cc OnOff.hh
40 OnOff.o:          OnOff.cc
41
42 # ---- divers ---------------------------------------------
43 clean:
44   $(RM) *.o
```

Programme 5.5 – Makefile - TopMain

J'utilise beaucoup de variables et règles par défaut du Makefile. On peut voir ces variables et règles grâce à la commande "make -p". Voici les principales variables et règles utilisées dans mon Makefile :

```
1 LINK.cc = $(CXX) $(CXXFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH)
2
3 %.o: \%.cc
4   $(COMPILE.cc) $(OUTPUT_OPTION) $<
5
6 COMPILE.cc = $(CXX) $(CXXFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c
7
8 $@  #cible
9
10 $^  #liste des dependances
11
12 $<  #premiere dependance
```

Programme 5.6 – Makefile - Options par défaut

Cependant, là aussi, des références vers la bibliothèque UML-RT sont présentes sur des fonctions pourtant bien définies. Après avoir avancé à tâtons sans grand résultat, j'ai décidé de faire preuve d'un peu plus de méthodologie.

5.3.1 Retour à un cas simple

Pour simplifier la résolution de bugs sur la bibliothèque UML-RT, j'ai réduit mon problème à un cas simple en créant une petite bibliothèque de gestion de led. L'objectif est d'être dans la même configuration que mon programme généré par Papyrus-RT et la bibliothèque UML-RT avec une bibliothèque bien plus simple et moins broussailleuse en .a et un programme principal dépendant de celle-ci. J'ai donc créé une classe *Led* qui va gérer sa déclaration sous Arduino* et la faire clignoter toutes les X secondes. La classe est donc telle que :

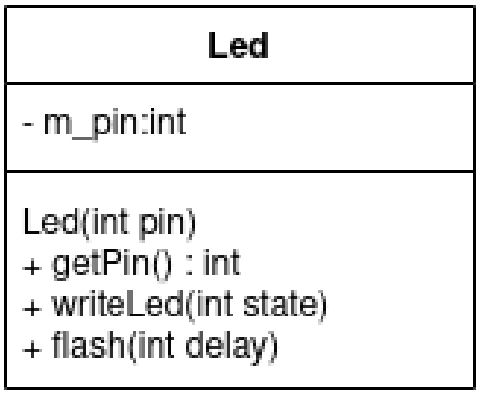

Figure 5.4 – Diagramme de classe de "Led".

```
#include "Led.h"

Led::Led(int pin){
  m_pin = pin;
  pinMode(pin,OUTPUT);
}

void Led::writeLed(int state){
  digitalWrite(getPin(),state);
}

void Led::flash(int time){
  this->writeLed(HIGH);
  delay(time);
  this->writeLed(LOW);
  delay(time);
}
```

Programme 5.7 – Classe Led.

Le constructeur déclare la led branchée sur le GPIO [9] “pin“ en sortie. La fonction “flash()“ allume la led X temps puis l'éteint pendant la même durée.

Après avoir programmé la classe, je l'ai donc compilée sous forme de bibliothèque en .a. Étant donné qu'elle appelle des fonctions Arduino*, j'ai dû lier la bibliothèque Arduino*. Pour simplifier, j'ai réalisé ce Makefile :

```
CXX=avr-g++

CXXFLAGS = -Wall -Wextra
CXXFLAGS += -std=c++17

# --------- les bibliotheques exterieurs
AVR_ARDUINO=/usr/share/arduino/hardware/archlinux-arduino/avr

CPPFLAGS += -I$(AVR_ARDUINO)/cores/arduino
CPPFLAGS += -I$(AVR_ARDUINO)/variants/standard

# ------ compilation
all: libLed.a

libLed.a: Led.o
  ar rcs $@ $<

Led.o: Led.cpp
  $(CXX) -c $< -o $@ $(CPPFLAGS)

clean:
  $(RM) *.o
```

Programme 5.8 – Makefile - bibliothèque Led

9. General Purpose Input/Output (Entrée-sortie à usage généra)

J'ai donc programmé un code principal très simple qui va construire un objet *Led* sur la pin 2 donc déclarer une sortie en GPIO 2. Ensuite, il va boucler sur la fonction "flash()" qui va allumer la led une seconde puis l'éteindre autant de temps. cela a alors pour effet de faire clignoter la Led.

```
1 #include <Led.h>
2
3 void setup(){}
4
5 Led led(2);
6
7 void loop(){
8   led.flash(1000);
9 }
```

Programme 5.9 – Programme principal .

À ce stade, je suis dans le même cas de figure qu'avec la bibliothèque UML-RT. Un programme principal qui appelle une bibliothèque externe en .a. Je reprends donc mon Makefile pour pouvoir le compiler en reliant la bibliothèque "Led". Alors, je n'ai plus de référence indéfinie vers la bibliothèque que j'ai créée, mais j'en ai vers les fonctions Arduino*. J'apprends alors que l'IDE Arduino*, à la première compilation, commence par compiler sa bibliothèque en statique dans un répertoire temporaire. Je décide donc, dans le but de tester, de copier cette bibliothèque statique dans le même répertoire que ma bibliothèque. Après tout cela, j'obtiens enfin un exécutable grâce au Makefile suivant :

```
CXX=avr-g++

CXXFLAGS = -Wall -Wextra
CXXFLAGS += -std=c++17

# --------- les bibliotheques exterieurs

AVR_ARDUINO=/usr/share/arduino/hardware/archlinux-arduino/avr

CPPFLAGS += -I$(AVR_ARDUINO)/cores/arduino
CPPFLAGS += -I$(AVR_ARDUINO)/variants/standard

LED_ARDUINO = /home/breizh/Documents/Cours/stage/Sujet/simplecase/lib

CPPFLAGS += -I$(LED_ARDUINO)
LDLIBS   += -L$(LED_ARDUINO)
LDLIBS   += -lLed
LDLIBS   += -lcore

CXXFLAGS+=-fpermissive -fno-exceptions -ffunction-sections -fdata-
    sections -fno-threadsafe-statics -Wno-error=narrowing -MMD -flto -
    mmcu=atmega328p
LDFLAGS+=-flto -fuse-linker-plugin -Wl,--gc-sections -mmcu=atmega328p

# ------- structure du projet ----------------------------

MAIN = simplecase.cpp
OBJ += $(MAIN:.cpp=.o)

# -------- dependances --------------------------------

all: simplecase

simplecase: $(OBJ)
  $(LINK.cc) -o $@ $^ $(LDLIBS)

simplecase.o: simplecase.cpp

# ---- divers ----------------------------------------------
clean:
  $(RM) *.o *.d
```

Programme 5.10 – Makefile - cas simple

Obtenir un exécutable est une bonne avancée, mais il faut maintenant le téléverser sur la carte Arduino*. Par curiosité, je retente d'inclure la bibliothèque via l'IDE et, cette fois, le programme compile puis est téléversé sur la carte. Je commence donc à chercher les différences notables entre la bibliothèque de mon cas simple (Led) et la bibliothèque UML-RT. L'une d'entre elles est l'extension de fichier : la bibliothèque UML-RT utilise .cc et .hh tandis que la mienne utilise .cpp et .h. Sachant qu'en C++ plusieurs extensions sont acceptées comme le .cc, .cpp ou .cxx, de même pour les .hh, .hpp, .h et .hxx, sans grande conviction, j'ai changé les extensions de fichiers de la

bibliothèque de mon cas simple en .cc et .hh pour coller au plus près de la bibliothèque UML-RT. Je réessaie de compiler (étant sûr et certain que cela ne vient ni du programme, ni des commandes de compilation) et cette fois, des références indéfinies vers ma bibliothèque (Led) apparaissent. J'en conclus donc que les références indéfinies viennent des extensions de fichiers.

5.3.2 Programme principal

Suite à mon test, je change toutes les extensions de fichiers par .h et .cpp ainsi que toutes les inclusions de tous les fichiers de la bibliothèque UML-RT pour pouvoir vérifier que les références indéfinies viennent bien de là. Je repasse par l'IDE Arduino* pour pouvoir tester l'inclusion UML-RT aux bibliothèques Arduino* et la plupart de mes références indéfinies disparaissent. Il reste cependant un certain nombre d'erreurs liées aux modifications que j'ai apportées à la bibliothèque. Pour des soucis de simplicité et pour éviter d'autres problèmes liés à la compilation, j'ai décidé de continuer avec l'IDE Arduino* plutôt qu'avec des Makefiles. De plus, cela me permettra de créer une bibliothèque facilement intégrable via l'IDE.

Certaines erreurs liées aux Thread, Mutex et à ce qui était dépendant d'un système d'exploitation ne sont pas apparues lors de la compilation de la bibliothèque en .a via le Makefile. Pour remédier à cela, j'ai commencé par mettre les fonctions du fichier d'en-tête des Mutex en commentaires (pour qu'elles ne soient pas exécutées) pour voir où ces fonctions étaient appelées. Après, il m'a suffi de tout retirer car le principe d'un Mutex est de protéger certaines parties du programme de la concurrence des threads. Arduino* n'ayant qu'un seul thread, il n'y a donc pas de concurrence et pas lieu de garder des protections.

Ensuite j'ai dû m'occuper des threads. L'une des premières choses que le programme principal fait est de lancer un thread qui va venir exécuter le code généré à partir du modèle. J'ai dû contourner la création du thread en gardant l'exécution du code généré. Ce fut un peu plus laborieux que prévu car dans la bibliothèque UML-RT c'est la std "pthread" qui est utilisée plutôt que simplement la std "thread" dont j'ai appris à me servir. L'idée est d'appeler la fonction exécutée par le "thread" sans créer le "thread". J'ai aussi dû modifier certaines conditions vérifiant quel thread exécutait la condition. L'avantage est que sous Arduino*, la question des threads est binaire étant donné qu'il n'y en a qu'un seul. Il n'y a donc plus besoin de vérifier si c'est le "bon" thread.

Un autre point à régler apparaît au niveau du lancement du programme principal. Un programme classique C ou C++ exécute la fonction "main()" qui retourne un entier et prend optionnellement deux arguments, argc et argv. Argv est un tableau de chaînes de caractères qui équivaut aux arguments donnés à l'exécution du programme (avec comme première valeur le nom du programme). Argc correspond au nombre d'éléments dans argv. Cependant, en Arduino* il y a deux fonctions de ce type : "setup()" et "loop()". "setup()" sert à la déclaration des éléments sur les GPIO et "loop()" sert de "main()". Cependant, loop() et main() n'ont pas le même comportement. Comme son nom l'indique, loop() est une boucle infinie. Ce qui n'est pas le cas de main() qui, lui, ne va s'exécuter qu'une seule fois. J'ai donc dans un premier temps simplement remplacé main() par loop() en rajoutant la fonction setup() vide. J'ai aussi enlevé les retours dans la fonction main() car loop() ne retourne rien (étant donné que le retour de la fonction main() sert à savoir comment s'est passée l'exécution, elle n'est pas nécessaire sur un système qui tourne en continu).

5.3.3 Type de carte

Après avoir résolu la majorité des soucis liés à la bibliothèque, un problème d'un autre type est survenu. Pour coller le plus possible à l'industrie, nous avions choisi la carte de type Arduino* la plus courante à savoir une Arduino Uno. Le problème des cartes embarquées* est que les ressources sont souvent très limitées et dans notre cas, on dépasse considérablement la capacité de la carte. En terme d'espace de stockage, le code principal et la bibliothèque prennent 2705% de la mémoire disponible. Pour régler ce souci mon tuteurs de stage et moi avions trois possibilités : soit optimiser énormément le programme et notamment la bibliothèque, soit changer de carte, soit un mélange des deux.

Voici un comparatif de plusieurs cartes possibles :

	Arduino Uno	Arduino Mega	NodeMcu 1.0	NodeMcu-32
Microcontrôleur	ATmega2560	ATmega2560	ESP8266	ESP32
Tension de fonctionnement	5V	5V	3.3 V	3.3 V
Nombre de GPIO Digitale	6	54	13	16
Nombre de GPIO Analogique	6	16	1	16
mémoire flash	32 Ko	256 Ko	4 Mo	4 Mo
mémoire Ram	2 Ko	8 Ko	128 Ko	512 Ko
Vitesse de processeur	16MHz	16 Mhz	80 MHz	160 Mhz

Le programme pur utilise 24990 octets et les variables globales utilisent 55405 octets soit au total 80395 octets. Pour n'importe quel type de carte, il y a une partie réservée pour le programme principal et une pour les variables. Ici c'est l'espace alloué aux variables qui pose problème.

Faire de l'optimisation sur la bibliothèque et sur le programme principal peut être très chronophage car cela implique soit de reprogrammer la bibliothèque de zéro en utilisant le moins de variables possibles, soit de reprendre chaque élément un par un pour le réduire au minium. Cependant l'optimisation aurait quand même l'avantage de supporter un plus grand nombre de cartes, ce qui, par extension, est plus représentatif du monde de l'industrie. L'autre possibilité est simplement de changer de carte pour en utiliser une avec plus d'espace de stockage. Pour pouvoir avoir une preuve du concept avant de se lancer dans une éventuelle optimisation et par souci de gain de temps, nous avons décidé de changer de carte.

- Pour Arduino Uno, la mémoire flash (espace de stockage) maximum est de 2048 octets ;
- Pour l'Arduino Mega elle est de 8192 octets ;
- Pour la nodeMCU 1.0 elle est fixé à 81920 octets ;
- Pour la nodeMCU-32 elle est aussi de 81920 octets.

Les deux nodeMCU feraient alors l'affaire étant donné leur mémoire flash conséquente. Cette mémoire est bien plus grande du fait que ces cartes ESP sont équipées d'un module WI-FI intégré et peuvent donc stocker un petit serveur web par exemple. Ayant une nodeMCU 1.0 à disposition, je teste le programme avec celle-ci. Les bibliothèques ESP8266 contenant déjà une structure "timeval" identique à celle de la bibliothèque UML-RT, j'ai dû l'enlever. Cependant, cette fois-ci, ce n'est plus la mémoire flash qui pose problème mais la mémoire RAM. Il m'a donc fallu utiliser la nodeMCU-32 qui a plus de mémoire RAM. J'ai dû installer dans l'IDE les modules spécifiques à l'ESP32 non intégrés par défaut pour pouvoir compiler pour ce type de carte. ESP32

n'a pas de "timeval" par défaut donc je l'ai remis. Mon programme principal compile sans erreur pour la nodeMCU-32. N'ayant pas cette carte à disposition, nous avons décidé d'en commander une pour tester. L'avantage des nodeMCU est la facilité de propotypage. Elle se connecte par micro-USB avec des pins utilisables sans soudure.

En attendant de recevoir la carte en question, j'ai décidé de mettre au propre la bibliothèque pour qu'elle soit facilement intégrable par l'IDE Arduino.

5.3.4 Bibliothèque UML-RT pour Arduino

Rendre la bibliothèque facilement intégrable par l'IDE Arduino permettra à d'autres stagiaires ou chercheurs d'utiliser mon travail facilement. D'autant plus que certaines mises à jour de l'IDE peuvent rendre certains programmes instables. De plus avec l'arrivée de l'IDE 2.0, il est important d'avoir une bibliothèque modifiable aussi simplement que possible pour pouvoir la déboguer efficacement le cas échéant. Pour cela, plusieurs choses sont à faire :

— Créer un ficher de propriétés de la bibliothèque. Les bibliothèques Arduino peuvent contenir un fichier de propriétés qui indique l'auteur, la version et un certain nombre d'informations salutaires pour l'utilisateur. N'ayant pas encore pu tester la bibliothèque sur une carte, celle-ci est donc en version 0.1 (phase de test du logiciel). J'en profite pour mettre le lien du gitlab sur lequel sont hebergés les modifications et documents concernant le port de la bibliothèque sous Arduino. Le fichier de propriétés est donc comme suit :

```
1 name=UML-RT
2 version=0.1
3 author=Zeligsoft, Elian Loraux
4 maintainer=
5 sentence=Library for UML-RT.
6 paragraph=Library for generated code by Papyrus-RT with UML-RT.
7 category=Device controle
8 url=https://gricad-gitlab.univ-grenoble-alpes.fr/hilin/uml-rt-port-
    for-arduino
9 architectures=*
```

Programme 5.11 – library.properties

Un autre avantage de créer ce fichier est qu'il est lu par l'IDE. Il permet d'afficher directement dans l'IDE le nom, le numéro de version et d'autres informations de ce type.

— Rajouter un readme [10] qui permet d'expliquer plus en détails les fonctions de la bibliothèque et tous les autres éléments qui peuvent avoir de l'importance comme le problème lié à l'espace de stockage sur les cartes.

J'en profite aussi pour mettre au propre le gitlab en intégrant le bibliothèque UML-RT sans et avec les modifications ainsi que le programme généré grâce à mon modèle de test.

10. lis moi

5.4 Débogage

J'ai pu réceptionner la carte nodeMCU-32 le 7 juin et donc tester le programme sur la carte. C'est la dernière ligne droite de la transition vers les cartes type Arduino*. Sans grande surprise, des erreurs sont survenues. Pour déboguer un gros projet, habituellement, on utilise un débogueur* (outils de débogage). L'IDE de base d'Arduino* ne proposant pas de débogueur*, je cherche une alternative. L'une des solutions est l'IDE Arduino 2.0. Un nouvel IDE Arduino est en cours de developpement et actuellement en beta test. L'IDE est bien plus moderne avec des fonctions plus avancées comme l'autocomplétion. Voici un petit comparatif :

FIGURE 5.5 – IDE Arduino 1.8.

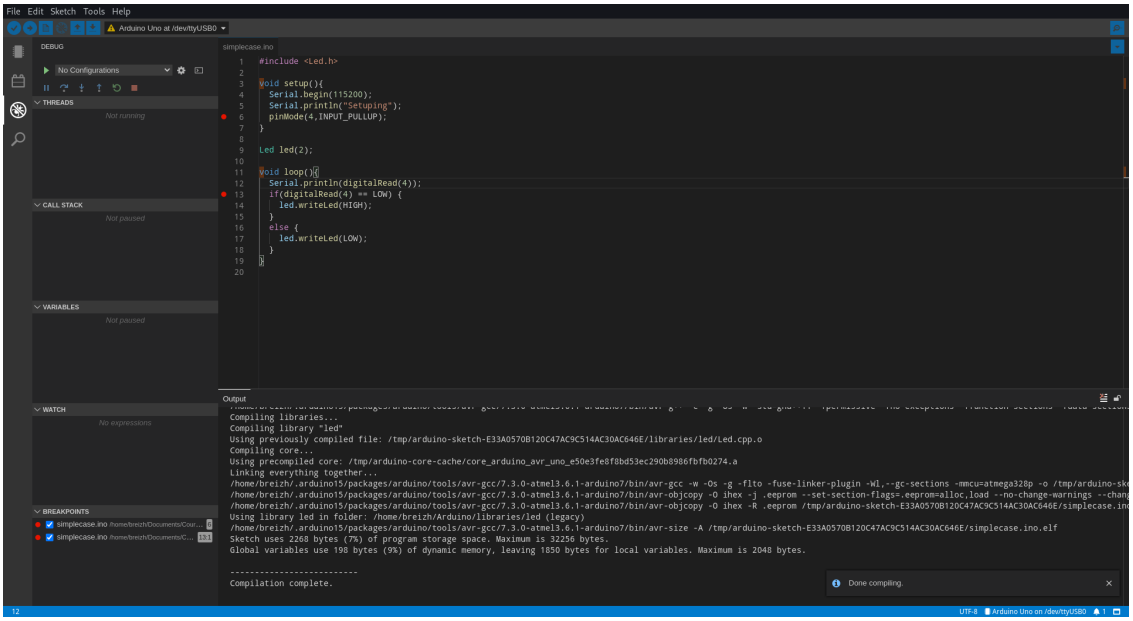

FIGURE 5.6 – IDE Arduino 2.0 Beta.

Cependant, à ce jour, le débogueur ne prend en charge que les cartes Arduino basées sur les plateformes SAMD et Mbed (famille MKR, Nano 33 IoT, Nano 33 BLE, Portenta, Zero). Je ne peux donc pas utiliser le débogueur de l'IDE 2.0 avec la nodeMCU et les autres cartes qui supportent le débogueur n'ont pas assez de mémoire pour stocker le programme et la bibliothèque.

Je dois donc déboguer manuellement en affichant des messages dans le terminal de la carte. Je commence à cibler le problème mais travaille encore actuellement à le résoudre.

6 | Applications possibles

Le domaine des systèmes embarqués étant vaste, le nombre d'applications est quasiment infini. Ils sont présents aujourd'hui dans tous les appareils du quotidien : les smartphone, les automobiles, IoT [1], dans aérospatiale, en robotique, dans l'armée et bien d'autres. Souvent, le développement de systèmes embarqués prend des années.

Si l'on arrive à exécuter directement des modèles de manière fiable sur des systèmes embarqués et quel qu'en soit le domaine, cela permettrait de réduire drastiquement le temps de développement (programmation, mais surtout débogage) et donc par extension les coûts engendrés. On pourrait alors imaginer des objets connectés moins chers pour le grand public ou permettre des économies substantielles pour les marchés spécifiques, par exemple pour les programmes spatiaux, ce qui pourrait accélérer les actions de conquête spatiale.

Actuellement, le temps de développement d'un produit dans certains domaines peut prendre plusieurs dizaines d'années. Réduire le temps de programmation permettrait de réduire ce temps de développement.

1. Internet of Things (l'Internet des Objets, les objets connecté)

7 | Conclusion

7.1 Applications réelles

Les applications à l'exécution de modèle sur des systèmes embarqués sont multiples. Cependant, à l'heure, actuelle elles ont certaines limites.

7.1.1 Carte type Arduino

Bien que les cartes de type Arduino* tendent à être représentatives du monde de l'industrie et de la production, elles sont loin de couvrir la majorité des domaines. Arduino* n'est pas la meilleure solution pour la plupart des systèmes pour plusieurs raisons :

le superflu : une carte Arduino* a un bon nombre d'éléments inutiles pour beaucoup de systèmes. Dans le cadre d'un thermomètre connecté, le système a besoin d'une GPIO numérique et avec un écran I2C (protocole de communication) la pin "SDA" et "SCA". Pour un Arduino Uno, cela fait dix-huit entrées-sorties inutilisées. Un industriel n'a aucun intérêt à rajouter du superflu dans un système embarqué* dont l'une des contraintes principales est la place qu'il occupe dans l'espace.

le prix : pour une carte Arduino Uno, il faut compter 20 € hors taxe. Cela est notamment dû à la documentation qui doit être l'une des meilleures du monde dans ce domaine. Le prix est aussi dû à la mise à disposition des outils par la société. L'environnement de développement, le forum (qui compte plus de 4 millions de postes) et arduino-builder, pour n'en citer que trois, sont au final amortis dans le prix de la carte.

7.1.2 Performance

On a pu voir au cours de la réalisation du projet que pour porter la bibliothèque UML-RT et le programme généré par Papyrus-RT, il faut des ressources non-négligeables, notamment en terme de mémoire flash et mémoire RAM. Sans faire d'optimisation dans la bibliothèque, les capacités de carte comme l'Arduino Uno ou Méga sont largement dépassées. On doit alors utiliser des cartes qui ont significativement plus de mémoire ce qui représente souvent un coût plus élevé. L'une des solutions est d'optimiser la bibliothèque pour réduire sa taille. Hormis le fait que ce soit très chronophage et source de nombreux bugs, il est quasiment impossible de faire rentrer le programme sur une Arduino Uno. En l'état, la bibliothèque et le programme principal prennent 2705% de l'espace disponible alloué aux variables. Même avec une très bonne optimisation, il faut diviser par 27 la place occupée par les variables globales.

L'une des approches possibles est de redimensionner les espaces alloués pour les variables en grignotant sur celui destiné au programme. Cela implique de contourner le dimensionnement fait pas l'IDE. Cela pourrait être combiné avec une optimisation de la bibliothèque et du programme. N'ayant pas le temps d'explorer cette piste, je ne suis pas sûr que celle-ci puisse aboutir et surtout qu'elle soit suffisante pour la majorité des cartes, mais cela reste envisageable.

7.2 Mon ressenti

Ce stage au sein du LIG fut très enrichissant pour moi dans de nombreux domaines. Il m'a permis de mettre un pied dans le monde de la recherche qui, pour ma part, est une réelle source d'inspiration. D'autant plus que le domaine de la robotique et des systèmes embarqués me passionne depuis quelques années maintenant, que ce soit en réalisant des petits systèmes domotiques ou en suivant les actualités du domaine. Des projets d'amateurs imprimés en 3D aux démonstrations spectaculaires de Boston Dynamics, en passant par les robots musiciens de Robocross Machines, c'est la recherche qui pousse ces innovations à se développer et à se démocratiser.

Tout au long du stage, j'ai eu l'occasion d'acquérir un certain nombre de connaissances supplémentaires ainsi que de compléter celles que j'avais déjà. J'ai commencé par largement étendre ma vision sur le champ d'application de la modélisation et de l'UML. Je m'étais principalement servi de l'aspect communication et conception de la modélisation sans jamais aborder l'aspect exécution de modèles. J'ai aussi énormément progressé en Makefiles. Je m'étais déjà documenté lors de mes cours de C++ pour pouvoir compiler mes tps, mais j'utilisais des Makefiles basiques sans variables ni règles par défaut. En lien avec le Makefile, j'ai exploré une partie de la documentation gcc et g++. Même si je suis loin d'être un expert tant le compilateur est vaste, je commence à être à l'aise pour les opérations de bases. Pour finir, j'ai tenu à réaliser mon rapport de stage en LaTeX, car ayant vu les avantages lors de la rédaction du rapport du projet de S3, je souhaitais pouvoir réaliser des documents en LaTeX sans trop de difficultés. Que ce soit pour générer des éléments comme le glossaire, le sommaire, le style global du document ou pour l'utilisation possible avec Git, je trouve le LaTeX pratique sous plusieurs aspects. En assumant la rédaction intégrale de mon rapport de stage, j'ai pu être confronté à la configuration du document et donc progresser significativement dans ce domaine.

Cette expérience m'a aussi beaucoup apporté professionnellement. Elle m'a permis, au travers du laboratoire, d'avoir une première approche du monde du travail en informatique. Elle m'a aussi fait découvrir pour la première fois le monde de la recherche, recherche qui englobe tout un panel de métiers souvent oubliésj que ce soit en recherche et développement, enseignants chercheurs ou bien d'autres. L'enseignement est une des voies que j'envisage, que ce soit pour les bacs technologiques ou à l'université. Enseignant chercheur pourrait être alors une alternative intéressante, car elle mêlerait cours et pratique de la matière enseignée ce qui, pour moi, est primordial pour continuer à satisfaire ma curiosité.

Pour finir, je tiens à exprimer à nouveau ma gratitude envers ceux qui m'ont permis de réaliser ce stage : Nicolas Hili, le LIG et l'IUT. À heure actuelle, cette expérience est l'une des plus enrichissantes que j'ai vécue, sans parler du plaisir que j'éprouve encore à réaliser ce projet.

Glossaire

arduino Arduino est la marque d'une plateforme de prototypage open-source qui permet aux utilisateurs de créer des objets électroniques interactifs à partir de cartes électroniques matériellement libres sur lesquelles se trouve un microcontrôleur. 4, 5, 10, 12–14, 16–21, 23, 26

cartes embarquées partie matérielle d'un système embarqué. 4, 21

débogueur Outil qui facilite la résolution de bug (historiquement un "bug" est un insecte coincé dans un composant informatique qui crée un court-circuit, il désigne aujourd'hui un un problème logiciel). 23

I2C Inter - Integrated Circuit. Protocole créé dans les années 80 par Philips, il permet la communication de plusieurs éléments en parallèle grâce à deux câbles. 26

intelligence artificielle n programme informatique qui se distingue par sa capacité à prendre des décisions. 3

système embarqué défini comme un système électronique et informatique autonome, souvent en temps réel, spécialisé dans une tâche précise. Le terme désigne aussi bien le matériel informatique que le logiciel utilisé. Ses ressources sont généralement limitées spatialement (encombrement réduit) et énergétiquement (consommation restreinte). 4, 26, 28

Bibliographie/Webographie

[1] Nicolas Hili. in-place dsl execution for embedded robotic systems(ipeers), 2020. Description du projet iPEERS.

[2] Alain Beaulieu Nicolas Hili, Juergen Dingel. Modelling and code generation for real-time embedded systems with uml-rt and papyrus-rt. pages 1–3, 2017.

[3] Francis Bordeleau. Industrial use of mbe : Ericsson experience, 2016. Possibilité d'utilisation de UML-RT et Papyrus-RT.

[4] Commission du LIG. Laboratoire d'informatique de grenoble. *Site du Lig*, 2020.

[5] Forum arduino. *Site d'Arduino*, 2005-Aujourd'hui.

[6] Forum open class room. *openclassrooms.com*.

Annexes

Trace d'exécution du programme "PingPong" généré à partir d'un modèle UML-RT.

```
~/.../PingPong_CDTProject/src >>> ./TopMain
Controller "DefaultController" running.
Starting game
ping sent!
Ponger is ready
Ping received!
pong sent!
Pong received!
ping sent!
Ping received!
pong sent!
Pong received!
ping sent!
Ping received!
pong sent!
Pong received!
ping sent!
Ping received!
pong sent!
Pong received!
ping sent!
Ping received!
pong sent!
Pong received!
ping sent!
Ping received!
pong sent!
Pong received!
ping sent!
Ping received!
Missed! Game over!
```

FIGURE 1 – Trace d'exécution du "ping-pong".

Build tool pour avr-g++

```
1
2  # tools
3  # Compiler
4  CC=avr-g++
5  # Archiver
6  AR=avr-ar
7  # Linker
8  LD=avr-g++
9
10 # compiler options
11 CC_DEF         = -D
12 CC_UNDEF       = -U
13 CC_INC         = -I
14 CC_OUT         = -o
15 CC_DEBUG       = -g -O0
16 CC_DEFINES       =
17 CC_FLAGS       = -c -Wall
18 CC_PPFLAGS       =
19 CC_INCLUDES      =
20
21 # dependencies
22 DEP_FLAGS      = -MP -MM
```

```
23 DEP_TARGET       = -MT $(subst .d,.o,$@)
24 DEP_FILE      = -MF
25
26 # linker options
27 LD_LIB         = -l
28 LD_LIBPATH       = -L
29 LD_OUT         = -o
30 LD_FLAGS       =
31 LD_PATHS       =
32 LD_LIBS        = rts pthread rt
33 LD_OBJS        =
34 LD_PRE_PROCESS     =
35 LD_POST_PROCESS    =
36
37 # archiver options
38 AR_FLAGS       = -rs
39 AR_OUT         =
40 AR_PRE_PROCESS     =
41 AR_POST_PROCESS    =
42
43 # extensions
44 CC_EXT     = .cc
45 OBJ_EXT    = .o
46 EXE_EXT    =
47 LIB_EXT    = .a
48 SHLIB_EXT    = .so
49 LIB_PRFX   = lib
50 DEP_EXT    = .d
51 DBG_FILES =
52
53 # Can optionally skip building dependencies (default is 'build
     dependencies')
54 # This can be overridden while invoking make with DEPEND=0
55 DEPEND=1
56
57 # This can be overridden while invoking make with DEBUG=1
58 DEBUG=0
59 # Add debugging compile flags
60 ifeq ($(DEBUG),1)
61 CC_FLAGS+= $(CC_DEBUG)
62 endif
```

Programme 1 – Buildtool - avr-g++