Ingénierie dirigée par les modèles : outils de modélisation, génération de code

Mohamed Daoudi

Université de Lorraine
Faculté des Sciences et Technologies

Master Ingénierie Electrique Electronique et Informatique Industrielle

Spécialité «Electronique Embarquée et Microsystèmes »

Année universitaire 2015/2016

Ingénierie dirigée par les modèles : Outils de modélisation, Génération de code

Mémoire présenté par « DAOUDI Mohamed »

Soutenu le 16/09/2016

Stage effectué au sein de l'entreprise Somfy
50 Avenue du Nouveau Monde, 74300 Cluses, France

Tuteur industriel : VORMS Johan

Tuteur universitaire : WEBER Serge

Bâtiment 1er cycle - Boulevard des Aiguillettes - B.P. 70239 - 54506 VANDŒUVRE-LES-NANCY CEDEX
Téléphone : 03.83.68.41.03 - Télécopie : 03.83.68.41.53

Remerciements

Je tiens tout particulièrement à remercier mon directeur de stage Johan Vorms, de m'avoir encadré et conseillé tout au long de ce stage. Il m'a donné l'opportunité d'effectuer un travail qui correspond fortement à mon domaine d'étude, et a su créer un contexte idéal pour ce travail. En plus de son soutien et du savoir qu'il m'a transmis, je tiens vivement à le remercier pour sa disponibilité et son encouragement.

Un grand remerciement à l'équipe pédagogique du master Electrique Electronique et Informatique Industrielle de l'université de Lorraine pour m'avoir apporté l'aide théorique et pratique nécessaire au bon déroulement du stage.

Je suis également extrêmement reconnaissant envers Julien Pistono, Jean-Rémy Gaxotte et Arnaud Desbiolles pour leurs conseils avisés, leurs remarques pertinentes et l'exemplarité de leurs rigueurs scientifique.

Un grand remerciement est adressé à tous mes collègues, membres de l'équipe logiciel au sein de Somfy pour l'ambiance très favorable qu'ils ont su créer autour de moi, et pour m'avoir intégré rapidement et facilement dans leur équipe.

Je remercie aussi toutes les personnes avec lesquelles j'ai eu le plaisir de collaborer et qui ont pu par la même occasion m'aider durant toute la durée de mon stage.

Enfin, je voudrais aussi remercier ma famille et mes amis, pour leur soutien inconditionnel.

Introduction

Dans le cadre de ma formation de master spécialité électronique embarquée et informatique industrielle j'ai réalisé un stage de fin d'études chez SOMFY, entre les mois d'avril 2016 et septembre 2016. Mon stage s'est déroulé au sein de l'activité Home&Building de Somfy où j'ai intégré le service logiciel de la direction technique. Pendant ce stage, j'ai eu l'opportunité de travailler sur différentes missions qui m'ont permis d'apprendre le métier d'ingénieur et de m'épanouir aussi bien au plan professionnel que personnel.

L'équipe d'ingénieurs logiciels se caractérise par le développement d'un certain nombre de logiciels appelé « briques» qui sont utilisés par les différentes entités de somfy. Ces briques sont actuellement développées en langage C_orienté_objet, et depuis peu certaines personnes compte tenu des contraintes de plus en plus complexes ont suggéré de faire le développement en C++. Ainsi, la mission principale de mon stage a été de faire un comparatif entre les deux langages C_oriente_Objet et C++.

Tout projet au sein de Somfy se base sur l'ingénierie dirigée par les modèles (IDM). Pour la phase de modélisation, les développeurs de Somfy utilisent le même outil depuis dix ans. Dans le but de faire évoluer l'outil de modélisation pour une version plus récente ou un changement d'outil pour un autre. Ma deuxième mission a été de faire une évaluation des outils de modélisation existants sur le marché, pour ce faire nous avons listé des critères que l'outil devra respecter pour répondre aux besoins des développeurs.

Pour finir, la dernière partie aura pour but de m'intéresser à la personnalisation ou customisation des générateurs de codes en C et C++ à partir des outils de modélisation choisis. Chaque outil intègre son propre générateur de code, le but est d'évaluer le niveau de customisation et d'optimisation de la génération pour s'approcher de ce qui est actuellement fait au sein de Somfy. Aujourd'hui la génération de codes est orientée vers la délivrance d'un résultat compact afin d'apporter une simplicité au niveau de la compréhension.

Après une présentation de l'entreprise Somfy, ce rapport de stage vous présentera les différents aspects de mes missions.

Sommaire

Table des illustrations

I. Présentation de Somfy

1. Somfy : historique *:*

C'est en 1960 que louis Carpano et Charles Pons fondent la société Carpano&Pons de mécanique, micromécanique et décolletage de précision à cluses en Haute-Savoie. Cette société est la plus dynamique dans une région réputée dans ce domaine pour la fabrication de pièces d'horlogerie suisse. Mais à la suite d'une restructuration de Carpano&Pons, une filiale de ce groupe est créée : la Société d'outillage et de Mécanique de Faucigny qui changera de nom pour une meilleure ouverture à l'internationale et deviendra Somfy.

En 1966 la mécanique de précision est un secteur en crise. La décision est alors prise de s'affranchir de la position précaire de simple fournisseur et l'entreprise part alors à la recherche d'un secteur où elle pourrait produire et commercialiser des produits finis.
A cette même époque qu'il y a eu le premier contact entre la société MARCHE ROCHE (fabricant lyonnais de volets roulants a commande manuelle) et le groupe CARPANO & PONS qui accepta alors l'étude et la fabrication d'un Lève Store (LS) motorise pour des applications domestiques, constitue essentiellement d'un motoréducteur équipé d'une fin de course, monté a l'intérieur d'un tube de 50 mm de diamètre. En 1967, SOMFY fabrique le premier lève-store électrique, puis les gammes LS50 (en 1969) et LS60 (en 1975), permettant a l'entreprise de conquérir une vraie spécialisation et du coup une indépendance technologique et stratégique.

En janvier 1984, la société DAMART SA rachète SOMFY au groupe CARPANO & PONS et crée en juin 1984 la holding SOMFY International SA. SOMFY accélère alors sa croissance, passant du statut de PME de la vallée de l'Arve a celui d'une structure internationale multi-entreprises, avec la création de plusieurs filiales en Europe et dans le monde et la maitrise d'un métier sur toute sa filière, de l'assemblage des produits a leur distribution.

Parallèlement, SOMFY assure son expansion horizontale, au rythme des opportunités. Le but est de diversifier la gamme de produits, afin de couvrir au plus le marche de l'ouverture (portes de garage, protection industrielle) et d'écarter ainsi des concurrents, même potentiels.
C'est ainsi que sont contrôlés, par rachat ou actionnariat majoritaire, Spirel Bobinage, Simu, Manaras, Monseigneur, Asa, FAAC, Simbac et plus récemment Galax, Mingardi, Siminor et le secteur "motorisation porte de garage" de Bosch. En 2001, la société Domis, spécialisée dans l'assemblage et la distribution de systèmes d'alarmes, intègre le groupe et dans la même période, le rapprochement de Simbac avec la société espagnole Gaviota a donné naissance au leader européen sur le marche des composants mécaniques pour stores et volets roulants.

L'orientation actuelle est de poursuivre la diversification du groupe : après avoir assis ses positions sur les différents marches, le groupe SOMFY compte également s'orienter vers le marche de la 'Home Automation' (équivalent au terme 'Domotique'), en créant des partenariats avec des sociétés spécialisées dans des secteurs tels que les alarmes, la ventilation, la gestion de l'éclairage, le chauffage, la climatisation, etc., afin de créer par la suite une interopérabilité entre ces systèmes.

2. Groupe Somfy

2.1 Généralité

En janvier 2008, Somfy a souhaité donner une meilleure visibilité à sa double stratégie de croissance en se structurant en deux branches distinctes :

- Somfy activités qui se consacre au métier historique du groupe à savoir l'automatisation des ouvertures de la maison et du bâtiment. Fort de sa culture de l'innovation, Somfy Activités crée et développe des solutions qui contribuent à l'amélioration des cadres de vie des habitants en répondant à leurs attentes de confort, de sécurité, d'économie d'énergie et d'autonomie des personnes.
- Somfy participations gère les investissements et participation du Groupe dans des entreprises qui ne relèvent pas du cœur de métier de Somfy. Les équipes de Somfy participations mobilisent les savoir-faire acquis au sein du Groupe pour accompagner le management et le développement des entreprises dont elle est actionnaire. Somfy détient, par exemple, 40 % du capital de Ciat à Culoz (Ain), leader européen de la pompe à chaleur et des équipements de climatisation, 34 % de l'italien FAAC, premier fabricant européen de systèmes et d'automatismes pour portails et portes de garage…

Somfy international est une branche d'activité à part entière du groupe DAMART. Elle regroupe toutes les filiales spécialisées dans le domaine de l'ouverture. Sa vocation est de concevoir, fabriquer et distribuer des moteurs et automatismes pour les protections solaires, les fermetures et la décoration d'intérieur, les fermetures commerciales, les portes de garage et les portails afin de répondre aux besoins de confort et de sécurité de l'utilisateur.

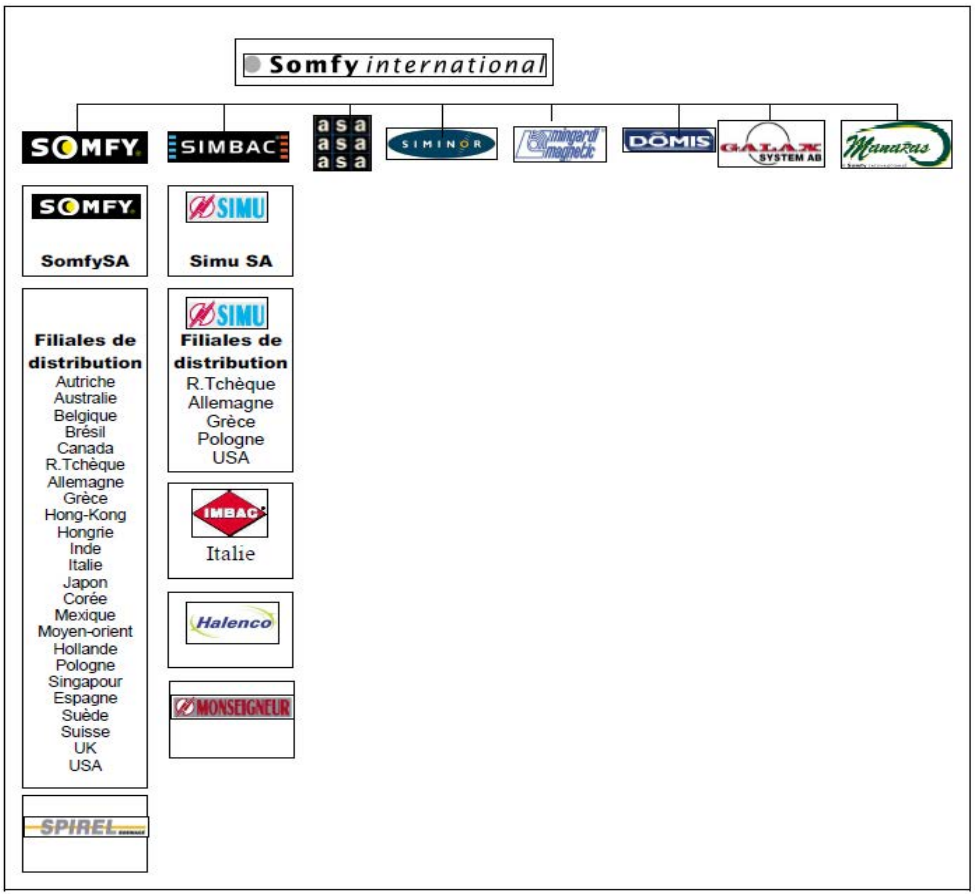


Figure 1 : Somfy international

Le groupe Somfy a un effectif total au 31 décembre 2015 de 6079 personnes (hors intérimaires) avec la répartition suivante :

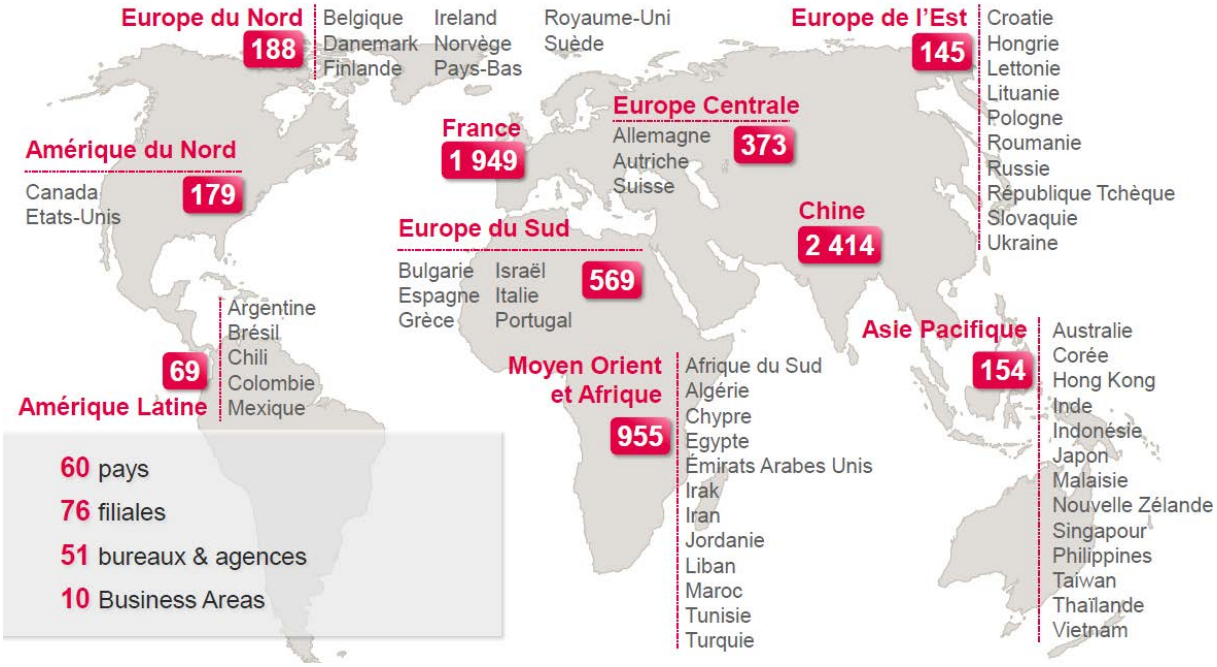


Figure 2 : Effectifs total du groupe Somfy

2.2 Chiffre Clés :

- **Chiffre d'affaire**
  - Somfy a réalisé un chiffre d'affaires de 1061,1 M€ sur l'année 2015, avec une hausse de 8,1% par rapport à l'année 2014. 76% du chiffre d'affaires est réalisé à l'étranger et 24% en France.

En milliers d'euros	31/12/15	31/12/14	Variation N/N-1
France	254 060	245 694	3,4 %
Allemagne	165 153	153 162	7,8 %
Europe du Nord	103 865	95 706	8,5 %
Europe de l'Est et Centrale	104 756	95 009	10,3 %
Europe du Sud, Moyen-Orient et Afrique	190 160	174 342	9,1 %
Asie-Pacifique	115 176	96 933	18,8 %
Amériques	127 979	120 883	5,9 %
SOMFY CONSOLIDÉ	**1 061 149**	**981 731**	**8,1 %**

Figure 3 : Chiffre d'affaire Somfy 2014/2015

- **Recherche et développement :**
  - 40 brevets déposés en 2015 et un total de 1 849 depuis la création de la société
  - 10% du chiffre d'affaire dédié à l'innovation

- **Production/Logistique**
  - 70 000 moteurs par jour
  - 158 000 000 moteurs produits depuis 1980
  - 7 sites de production dont cluses et Gray en France, Bologne et Schio en Italie, Sitem en Tunisie, LianDa et Dooya en Chine
  - Centre Logistique de 27 000 m² en France, et plus de 50 entrepôts logistique dans le monde
  - 32 000 clients professionnels
- **Effectif**
  - 6 079 personnes hors intérimaires
  - Moyenne d'âge : 40 ans
  - Plus de 50 nationalités
- **Qualité**
  - 5 ans de garantie totale sur les moteurs et automatismes Somfy
  - Plus de 600 homologations dans le monde

2.3 Marchés et produits

Aujourd'hui Somfy est l'un des leaders mondiaux de la motorisation, de l'automatisation des ouvertures de l'habitat et du bâtiment et des systèmes d'alarme. Le groupe affirme sa présence pour la maison et le bâtiment, Somfy conçoit des solutions d'automatisation faciles d'installation et d'utilisation.

- **Pour la maison**

1 Automatisation et protection des stores extérieurs
2 Motorisation des volets roulants, pour une meilleure sécurité et des économies d'énergie.
3 Motorisation des stores d'intérieur, pour une meilleure gestion de la lumière et de la décoration.
4 Automatisation des portes de garage.
5 Grâce à une interface web ou un smartphone, le consommateur pilote les applications Somfy et les autres équipements de la maison interconnectés via la technologie radio

- **Pour le bâtiment**

6 Solutions complètes d'automatisation pour les façades des bâtiments : fenêtres et stores, afin d'améliorer le confort visuel et thermique, tout en réduisant les coûts énergétiques.
7 Les utilisateurs peuvent également contrôler leur environnement à l'aide de télécommandes individuelles.

Figure 4 : Offre produit Groupe Somfy

Nous pouvons considérer trois grandes familles de produits Somfy :

- Les *actionneurs*, improprement appelés moteurs, sont la plupart du temps constitués d'un moteur asynchrone, d'un condensateur, d'un frein, d'un réducteur (pour obtenir en sortie une vitesse de rotation acceptable et augmenter le couple) et d'un module de réglage de fins de course. Dans la plupart des applications, ils sont de forme tubulaire et sont introduits dans le tube d'enroulement du produit (store volet roulant, grille, rideau métallique…)

Voir annexe 1 – Exemple d'actionneur tubulaire Somfy

- Les *boîtiers d'automatisation* et *de commande électronique*, ou systèmes de commandes, assurent un ensemble très large de fonctions destinées à apporter à l'utilisateur confort, sécurité et maîtrise de l'énergie (interrupteurs, horloges journalières et hebdomadaires, automatisation vent-soleil, commandes à distance par des applications Smartphone …)

Voir annexe 2 – Exemple de commande électronique Somfy

- Les *accessoires* ou *périphérique* qui permettent l'installation, l'adaptation ou l'intégration des actionneurs et boîtiers de commande dans les systèmes de fermeture, de protection solaire et de fermetures industrielles et commerciales.

Voir annexe 3 – Exemple d'accessoire Somfy

Figure 5 : Produits de Somfy

II. Etat de l'art

1. Contexte :

Les systèmes embarqués sont désormais utilisés dans plusieurs domaines d'application tels que les actionneurs, les boîtiers d'automatisme ou les accessoires au sein de l'entreprise Somfy. Devant assurer de plus en plus de fonctionnalités, ceux-ci deviennent de plus en plus complexes. A cette complexité s'ajoute une contrainte liée au développement rapide des systèmes embarqués dans les marchés. Cette contrainte oblige les industriels à fournir des produits toujours plus innovants et à un prix bas.
Pour gérer ce problème, l'ingénierie dirigée par les modèles (IDM) [1] permet aux développeurs de systèmes embarqués de se concentrer désormais sur l'élaboration de modèles abstraits, plutôt que sur des concepts liés à l'algorithmique et la programmation. Pour ce faire, les développeurs se basant sur l'ingénierie dirigée par les modèles se basent sur une approche qui composée de 3 étapes : la modélisation UML (Unified Modeling Language) [2] [3], la génération de code et la compilation du code généré pour obtenir l'exécutable de l'application (Code final). La génération de code à partir d'un modèle est réalisée automatiquement grâce notamment au générateur de code inclus dans les modeleurs ou un générateur de code développé par l'entreprise elle-même, cette étape étant automatique, elle permet un gain de temps et améliore considérablement la qualité de code du produit.

2. Problématique :

L'ingénierie dirigée par les modèles permet certes une génération de code automatique et rapide, mais présente un inconvénient dû au niveau d'abstraction qui est différent entre les langages de modélisation et les langages de programmation et en conséquence n'offrent pas les mêmes concepts. Par cette problématique, il est très difficile de traduire tous les concepts du langage de modélisation vers des concepts du langage de programmation. De ce fait lors de la génération de code, il y a souvent des pertes d'informations liées à la sémantique du langage de modélisation. En conséquence, les développeurs au sein de l'entreprise Somfy modifient souvent le code C,C++ généré en changeant, supprimant ou ajoutant des instructions qui, une fois compilées permettent d'avoir un code exécutable plus optimisé pour un meilleur gain au niveau de la vitesse du code ou l'empreinte mémoire.

3. Solution:

Concernant la problématique au niveau de la génération de code, Somfy a décidé de développer son propre générateur de code à partir des modèles UML modélisé sous l'outil Rhapsody [4]. Le générateur de code Somfy (GDC_Somfy) comme appelé permet de générer du code en C grâce à une liaison entre l'outil et le générateur de code. Concernant la génération de code en C++, Somfy utilise UML Inside qui est un générateur de code payant compatible avec Rhapsody.

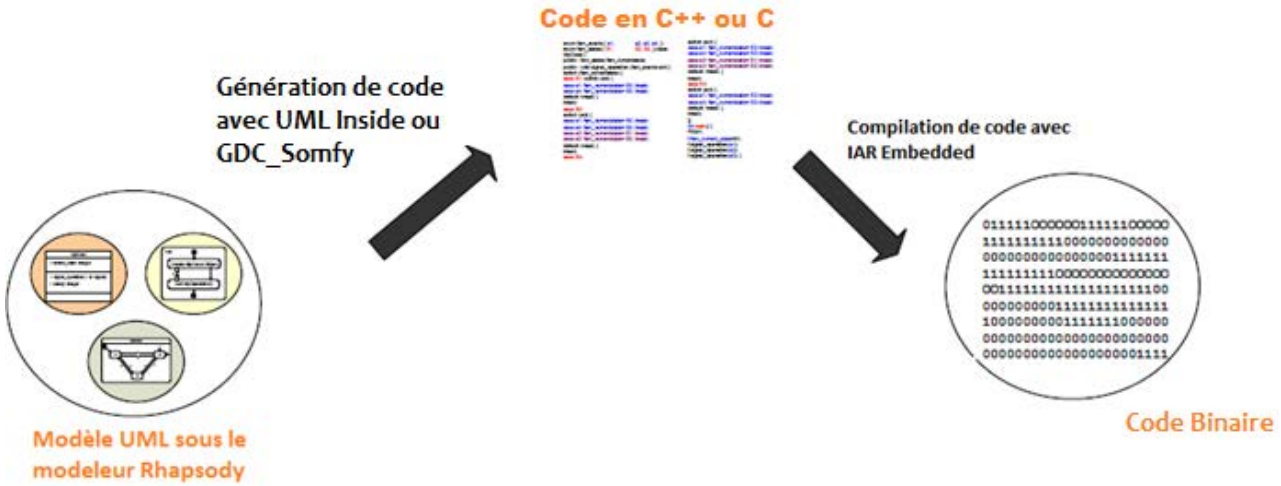


Figure 6 : Démarche génération de code chez Somfy

III. Comparatif des deux langages de programmation C_orienté objet et C++

1. Objectif :

Au sein de l'entreprise Somfy, tous les projets se basent sur l'ingénierie dirigée par les modèles. Et comme vu auparavant, nous pouvons générer du code en C ou C++ avec l'utilisation des deux générateurs de code. Actuellement la majorité des projets est réalisée en langage C, mais suite à des contraintes de plus en plus complexes et des tailles restreintes des mémoires des microcontrôleurs utilisés, la question de savoir le quel des deux langages de programmation présente les meilleures performances se pose.
À mon arrivée, la modélisation d'un grand projet était en cours. À la fin de cette modélisation, la décision de savoir si ce projet sera généré en langage C ou C++ n'était pas encore prise. C'est dans le but d'apporter une réponse que ce premier travail me fut confié. L'objectif était de faire un comparatif entre ces deux langages de programmation à partir d'un modèle UML et évaluer les informations fournies afin de prendre une décision.
Après avoir généré le code, une configuration manuelle est obligatoire afin de réaliser ce comparatif entre la méthode utilisée par la business group controls pour le C++ et celle utilisée par le service Connected Solutions Activity pour le langage C.
Les critères de comparaisons traités dans ce document seront basés sur le résultat de la génération de code et porteront sur les critères suivants :

- Empreinte mémoire (RAM, ROM)
- Vitesse d'exécution

Cette étude permettra par la suite de faire le choix entre l'implémentation C_orienté objet et C++ en sachant que SOMFY réalise toutes les conceptions via une approche objet avec UML (Unified Modeling Langage).

2. Première étape : la modélisation

Le langage UML (Unified Modeling Language) [2] [3] est un langage de modélisation graphique à base de pictogrammes. Il est apparu dans le monde du génie logiciel, dans le cadre de la conception orientée objet. Souvent utilisé par les développeurs dans les projets logiciels, il peut être utilisé dans toutes sortes de systèmes en dehors du domaine de l'informatique.
Pour réaliser un comparatif cohérent, nous choisissons de réaliser un diagramme de classes statique UML générique, prenant en compte les principales interactions utilisées dans les modèles Somfy. Ce diagramme est constitué de 7 classes dont 4 comprenant une machine à état, ce choix peut se justifier par le fait que les projets chez Somfy sont basés sur une approche de modélisation prépondérante de machines à états. Pour que ce comparatif soit le plus représentatif possible, nous avons deux sortes d'héritages. L'héritage simple (héritage entre la classe A et B) et un héritage multiple (entre les classes B1,B2 et B) figure 2

Les machines à états des classes A, B1 et B2. La machine à états de B est un simple héritage de la machine à état de A. Les classes C, D et E ne possèdent pas de machine à états.

Voir annexe 4 pour les machines à états

Chaque classe possède également un attribut et une méthode qui lui est propre (MethodeA, AttributeA pour la classe A par exemple…).

Le générateur de code UML Inside utilisé pour la génération de langage C++ ne gère pas la surcharge des états de la machine à états, c'est-à-dire qu'on ne peut pas déclarer deux ou plus de fois une machine en gardant le même nom des états. La solution qui a été trouvée est d'inclure dans la machine à états de la classe A les états des machines à états des classes B1 et B2. En revanche, la surcharge des conditions et des transitions s'effectuera dans la machine à états de B1 et B2.

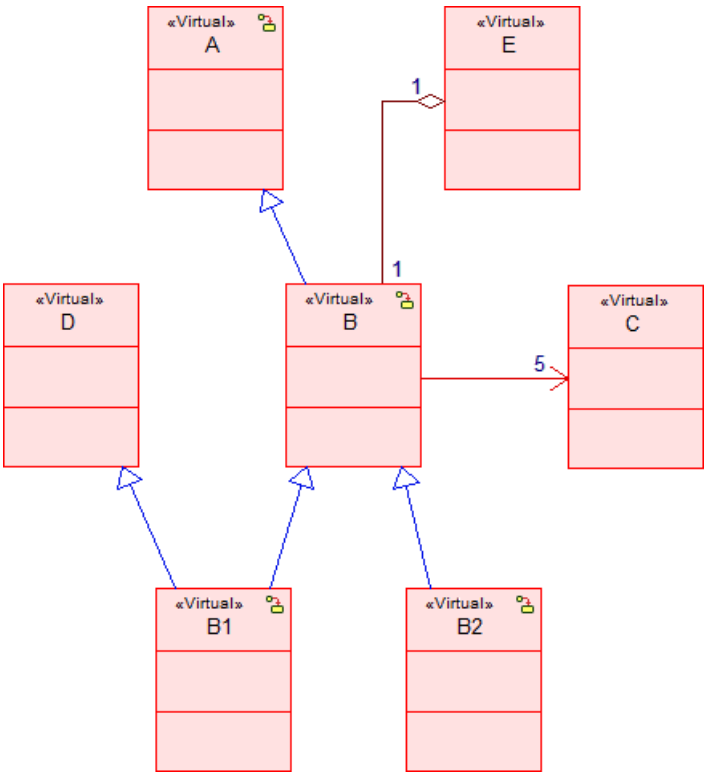


Figure 7 : Modèle UML de base pour le comparatif

3. Deuxième étape : Génération de code :

La génération de code à partir des modèles UML exécutable a pour cible les langages de programmation de forme textuelle, elle est le plus souvent basée sur l'approche M2T (Model to Text Transformation) [5]. Plusieurs générateurs de code sont basés sur cette approche, on peut citer parmi eux Acceleo [6] qui est un plugin sur eclipse. Actuellement avec l'évolution des outils de modélisation, certains modeleurs permettent la génération de code automatique avec un générateur de code intégré directement au modeleur. Le plus souvent, le code généré n'est pas optimisé c'est pourquoi les entreprises préfèrent développer leurs propres générateur de code. C'est le cas de Somfy qui s'intéresse seulement à la génération de code à partir des modèles comportementaux d'UML.

3.1 : Génération de code de modèle UML pour le comparatif

Dans ce comparatif, le diagramme statique sera implémenté alternativement avec les 2 langages (C (Object SOMFY) et C++) et en utilisant 2 méthodes d'implémentation différentes :

- Héritage multiple selon la méthode Somfy en langage C avec un gestionnaire de mémoire RamPool: C'est la façon dont Somfy implémente un héritage en utilisant un langage qui n'est pas orienté objet.
- Héritage multiple selon la méthode une entité de somfy Busniss Group controls avec un gestionnaire de mémoire GenPool (Libraire GenLib développée au sein de la BG controls) qui permet de faire une allocation pseudo-dynamique des objets avec le langage C++

Après avoir généré le code avec la méthode vu précédemment, une étape de modification et configuration manuelle est obligatoire afin de réaliser un comparatif entre la méthode utilisée par la Business group controls pour le c++ avec le gestionnaire mémoire GenPool et celle utilisée par le service Connected Solutions Activity pour le langage C_orienté objet. Pour pourvoir comprendre ces deux méthode nous allons décrire par la suite ce que représente les deux gestionnaire de mémoire

➢ GenPool :

Pour la gestion de la mémoire RAM, la Business Group controls a développée au sein de leur service la librairie Genlib. Dans cette librairie nous trouvons le GenPool qui permet une allocation de mémoire dynamique avec gestion de la RAM mutualisé. Cette méthode présente l'avantage de ne pas fragmenter la RAM lors des allocations pour des objets de grande taille.

L'utilisation de GenPool est dédiée aux développements où la RAM est limitée. Il évite la fragmentation de la RAM pour une meilleure allocation des objets. Il permet aussi à l'utilisateur de gérer l'allocation dynamique en C ++.

Lorsqu'on utilise le GenPool, le constructeur « new » et le destructeur « delete » sont surchargés pour gérer leurs comportements par la RAM disponible pour une allocation dynamique qui a été statiquement allouée au GenPool, c'est pourquoi on appelle cette allocation pseudo-dynamique.

Pour ce faire, le GenPool divise sa mémoire disponible en plusieurs partie (appelé pools de chunks) de différentes tailles (classées par ordres croissants). Lorsqu'on essaye d'allouer dynamiquement de la mémoire pour un objet d'une certaine taille, On parcourt tous les pools jusqu'à en trouver un de libre et de taille égale ou supérieure à celle de l'objet, c'est à ce moment que l'allocation s'effectue.

De cette façon l'allocation d'un objet sera toujours effectuée dans le plus petit chunk disponible dans la mémoire, sachant que l'utilisateur connait la taille de chaque objet et que c'est lui qui définit les chunks, ceci permet d'éviter la fragmentation de la mémoire qui des fois conduit à des cas ou l'allocation d'un grand objet s'effectue mais pas dans un seul endroit unique de la RAM.

Voir annexe 5 pour la configuration de GenPool et un exemple

➢ RamPool

Ce gestionnaire de mémoire est destiné à gérer tous les ensembles d'objets utilisé chez Somfy dans les développements embarqués écris en langage C. En effet, instancier des objets en langage C ne va pas de soi. Il est nécessaire de gérer à l'aide de tableaux les instances disponibles et celles utilisés. RamPool a été développé pour fournir un moyen de centraliser et d'uniformiser la gestion des ensembles d'instances de tous les objets codés en C chez Somfy.

L'instanciation des objets s'effectue dans un tableau que le développeur déclare, ce tableau n'est pas déclaré sous forme de case dans l'ordre croissant comme cela est fait dans le GenPool. Donc avec cette méthode nous pouvons avoir une fragmentation de la mémoire dans le cas d'une allocation d'un objet de grande taille. Dans le cas d'une instanciation d'un objet, nous parcourons le tableau à la recherche d'un espace suffisant pour allouer l'objet, une méthode permet de savoir s'il est possible de créer une instance. Elle renvoie FALSE s'il reste de la place dans le tableau et TRUE si le tableau est plein.

Voir annexe 6 pour la configuration de RamPool et un exemple

4. Banc de test

4.1 Généralité

Dans un premier temps nous allons faire un comparatif entre l'utilisation de l'héritage selon la méthode Somfy avec l'utilisation d'un langage qui n'est pas orienté objet (LangageC) avec un gestionnaire de mémoire RamPool et l'utilisation de l'héritage avec un gestionnaire de mémoire GenPool pour le langage C++. Ce premier comparatif à pour but de savoir lequel des deux langages de programmation (C et C++) présente le plus d'avantages.
Dans une deuxième partie nous allons nous intéresser seulement à l'héritage en langage C++, avec deux gestionnaires de mémoire différents (RamPool et GenPool). GenPool étant dédié seulement au langage C++ et RamPool peut être utilisé dans les langages C ou C++, ce comparatif aura donc pour but de savoir le quel des deux gestionnaires de mémoire est plus avantageux à utiliser.

4.2 Matériel

Les exécutables seront générés avec le logiciel IAR Embedded et ils seront testés sur un microcontrôleur STM32L152 de chez STMicroelectronics implanté sur une platine de développement STM32L-Discovery MB963 B.

Les pins de sorties utilisées pour mesurer les différents temps d'exécution seront les pins 6 et 7 du port B. Ces pins correspondent aux deux LED, bleue (Pin PB6) et verte (Pin PB7), présentes sur la platine. Ces temps seront mesurés avec un oscilloscope KEYSIGHT InfiniiVision MS0-X 3024T.

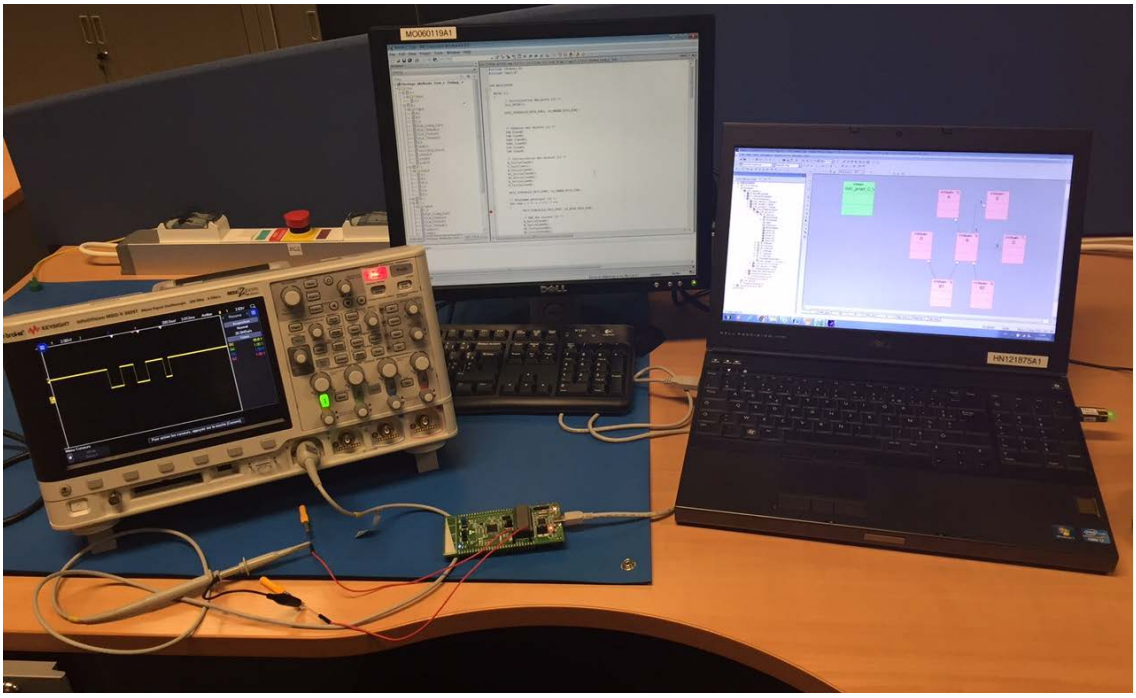


Figure 8 : Photo du banc de test

5. Comparatif entre langage C++ avec GenPool et langage C avec RamPool

5.1 Résultat taille de code

Pour plus d'informations concernant les tailles des codes, on a annexé les différents fichiers .map à la fin de ce document. ***Voir annexe 7***

5.1.1 : Taille prise dans la ROM (Read-Only Memory)

Le tableau ci-dessous nous permet de comparer l'implémentation en langage C_orienté_objet selon la méthode Somfy avec le gestionnaire de mémoire RamPool, et le langage C++ avec l'utilisation de GenPool de la Business Group Controls.

	Programme en langage C++ Avec GenPool	Programme en langage C avec RamPool	ECART (%)
Taille avec optimisation Vitesse+taille (Octets)	**2 234**	**2 424**	**8**
Taille sans aucune optimisation (Octets)	**2 904**	**2 848**	**2**

Pour les différents projets au sein de Somfy, le choix de compiler un projet sans aucune optimisation n'est pas souvent rencontré. Les projets ont beaucoup de contraintes, et de plus les microcontrôleurs utilisés ont une mémoire limitée (RAM et flash) ce qui oblige un choix d'optimisation de vitesse ou de taille, et parfois même un mixte des deux.

5.1.2 : Taille prise dans la RAM (Random Access Memory)

	Programme en langage C++ Avec GenPool	Programme en langage C avec RamPool	ECART (%)
Taille (Octets)	**1 116**	**1 252**	**12**

En taille de RAM, le langage C++ est plus volumineux de 12% que le langage C.
Le choix de l'optimisation n'est pas important ici, car cette différence de 12% reste indépendant de la méthode d'optimisation.

5.1.3 : Empreinte mémoire de chaque objet

La Taille des fichiers générés par les compilateurs IAR pour une optimisation maximale (Vitesse+taille) :

- La taille des fichiers ***.o*** se trouve dans le fichier .map ***voir annexe 7***

	Langage C++ GenPool (Octets)	Langage C RamPool (Octets)	ECART (%)
Objet A	**126**	**148**	**17,4**
Objet B	**238**	**388**	**63**
Objet B1	**116**	**248**	**113**
Objet B2	**138**	**288**	**108**
Objet C	**10**	**116**	**1060**
Objet D	**12**	**40**	**233**
Objet E	**18**	**112**	**477**
Total des objets	**658**	**1340**	**102**
GenPool	**532**	**X**	**X**
Main	**860**	**856**	**0.4**

Pour ce qui est de l'empreinte mémoire, l'héritage en langage C est plus volumineux de **102%** que l'héritage en C++.
Nous n'avons pas pris en compte les tailles de GenPool et le Main car leurs tailles est statique. Quelque soit le nombre d'objets que le projet contient, la taille de GenPool reste inchangée. Donc inclure sa taille dans notre comparatif risque de fausser totalement nos résultats. Concernant le fichier main sa comparaison reste négligeable entre les deux langages de programmation.

Concernant la grande différence pour les deux objets C et E entre le langage C et C++, ceci s'explique par l'ajout manuel des lignes du code pour réaliser le langage C_orienté_objet comme cela est fait chez somfy, avec gestionnaire de mémoire RamPool.

5.2 : Résultat vitesse d'exécution du code :

	Langage C++ GenPool	**Langage C RamPool**	**ECART (%)**
Création et initialisation [1]	**832 µs**	**1130 µs**	**36**
Destruction [2]	**512 µs**	**75 µs**	**580**
RAZ [3]	**74 µs**	**29 µs**	**155**
Machine à états de A [4]	**94 µs**	**108 µs**	**15**
Machine à états de B [5]	**88 µs**	**114 µs**	**29,5**
Machine à états de B1 [6]	**112 µs**	**136 µs**	**21,4**
Machine à états de B2 [7]	**136 µs**	**160 µs**	**18**
Méthodes [8]	**44 µs**	**79 µs**	**80**
Boucle principale [9]	**548 µs**	**622 µs**	**13,5**
Temps d'exécution du code = [1] + 100 x [9] + [2]	**56,1 ms**	**63,4 ms**	**13**

En temps d'exécution le langage C++ est plus rapide que le langage C avec le RamPool de 13 %.

Pour détruire une instanciation d'un objet et libérer l'emplacement mémoire alloué, les deux gestionnaires de mémoire utilisent deux méthodes différentes. Le GenPool réalise une allocation pseudo-dynamique contrairement au RamPool qui fait une allocation statique. La libération de mémoire avec le GenPool se fait avec la fonction Genfree() qui est appelée à travers le destructeur de l'objet, alors que le RamPool libère la mémoire seulement avec le destructeur de l'objet sans faire appel à aucune autre méthode. Ce qui explique la grande différence de vitesse entre les deux méthodes pour la destruction des objets.

L'utilisation de Genpool est plus rapide que le Rampool concernant l'allocation de mémoire des objets de petite taille. Dans notre cas nous avons les tailles suivantes pour nos objets : A/E = 8, B = 36, B1 = 44, B2 = 40, D/C = 4 en octet, ce qui peut expliquer la rapidité de GenPool par rapport au RamPool.

Pour le GenPool les chunks sont déclarés dans un ordre croissant, cependant lorsqu'un objet est de taille conséquente, il faudra donc parcourir tout les chunks de petite taille jusqu'à atteindre un chunk qui peut contenir cet objet. Ce qui rend le GenPool plus long pour l'allocation de mémoire pour des objets de grande taille.

Certains produits commercialisés par Somfy sont initialisés qu'une seule fois durant leurs vies de produits, donc le critère de comparaison 'initialisation et création' de l'objet n'est pas un critère prépondérant dans notre comparatif.
Ce critère devra être prise en compte dans certains cas et peut devenir un facteur déterminant pour d'autres produits tels que ceux de l'entité Business Group Controls (Produits de commande des stores par exemple), avec des applicatifs qui créent et détruisent les objets en fonction de leurs besoins.

5.3 : Synthèse des résultats

Comme le reprend cette synthèse, nous pouvons en conclure que le langage C++ est plus avantageux que le langage C pour les projets au sein de l'entreprise Somfy.

V : Avantage / = : Equivalent / X : Désavantage

Fonctionnalité	C++ GenPool	Langage C RamPool	Commentaire
Héritage	V	X	Héritage complexe en C
Modification du modèle en cas d'héritage	V	X	Modification plus complexe en C
Taille ROM	V	X	C++ prend un peu moins de place
Taille RAM	V	X	C prend un peu plus de place
Le temps d'exécution du code	V	X	L'exécution du code est plus rapide en C++.
Initialisation et création des objets	V	X	Le temps en C est 1,5 plus long qu'en C++.
Destruction des objets	X	V	C++ plus long que le C
RAZ des MAE et attributs	X	V	
Taille du fichier binaire	V	X	Pour la méthode SEM_T le binaire prend plus de place
L'initialisation des classes mères à partir de l'initialisation des classes filles	V	X	En C il faut le faire manuellement alors qu'en C++ c'est généré automatiquement
Modifications manuelle ajoutées au code généré	V	X	Moins d'ajout de ligne de code pour le C++ que pour le C, en C toutes les méthodes et attributs sont à mettre dans un **Define**, et l'initialisation des classes mères à partir de l'initialisation des classes filles…)
Surcharge des machines à états des classes mères	V	V	En C++ la surcharge des machines à état de la classe mère par la classe fille peut être évitée grâce à une option lors de la génération du code. En C de base elles ne sont pas surchargées.
La maintenance et la mise à jour des compilateurs	V	X	Les compilateurs C++ ont une maintenance et des mises à jour plus fréquentes (correction de bug, ajout de nouvelles fonctions pour respecter les dernières normes…).
Appel des méthodes	V	X	L'appel des méthodes est effectué plus rapidement en C++.
Exécution des machines à états	V	X	En C++ l'exécution des MAE est plus rapide de 15 à 30%
Instanciation d'un objet	V	X	En C on a besoin d'avoir une variable d'état pour savoir si on a bien instancié un objet ou non.

6. Comparatif entre Langage C++ avec GenPool et Langage C++ avec RamPool

Dans cette deuxième partie nous allons nous intéresser à un deuxième comparatif entre deux méthodes d'allocation de mémoire pour le langage C++. Il s'agit de faire une comparaison d'un héritage multiple avec un gestionnaire de mémoire RamPool et un gestionnaire de mémoire GenPool tout en utilisant le même langage de programmation à savoir le C++.
Ce comparatif sera réalisé sur la base de même modèle UML vu précédemment *(figure7)*.

6.1 : Taille de code

6.1.1 : Taille prise dans la ROM

	Programme en langage C++ Avec GenPool	Programme en langage C++ avec RamPool	ECART (%)
Taille avec optimisation maximale (Octets)	**2 234**	**2 228**	**0**
Taille sans aucune optimisation (Octets)	**2 904**	**2 612**	**11**

Concernant la taille de la ROM, nous pouvons voir dans un premier temps que le C++ avec le gestionnaire de mémoire RamPool est plus avantageux que le langage C++ avec le gestionnaire GenPool. Mais comme indiqué auparavant, la taille de GenPool (environ 530 octets) est statique et ne change pas en fonction de nombre d'objets à instancier. Contrairement au RamPool qui lui a chaque objet à instancier, il faut ajouter des lignes de code qui ont une taille d'environ 70 octets.

```
STM32L152-EVAL.map
*******************************************************************
*** ENTRY LIST
***

Entry                       Address    Size  Type      Object
-----                       -------    ----  ----      ------
B::RAZ_B()                0x08000513    0xe  Code  Gb  B.o [1]
B::~B()                   0x08000521    0x2  Code  Gb  B.o [1]
B::~B() [subobject]       0x0800050f    0x4  Code  Gb  B.o [1]
C::C()                    0x08000415    0x4  Code  Gb  C.o [1]
C::C_Init()               0x0800041f   0x2c  Code  Gb  C.o [1]
C::C_InitClass()          0x0800044b   0x1a  Code  Gb  C.o [1]
C::~C()                   0x0800046d    0x4  Code  Gb  C.o [1]
```

Figure 9 : Taille Code pour le gestionnaire RamPool

Pour faire de l'allocation de mémoire en utilisant RamPool, il faut créer deux méthodes dans l'objet à allouer. Ces deux méthodes prennent une taille de 70 octets. Le cas d'un projet ayant un nombre apportant d'objet à instancier, la taille de la ROM avec un gestionnaire RamPool sera conséquente.

```
Entry                          Address    Size   Type       Object
-----                          -------    ----   ----       ------
D::D()                       0x0800081d     0x2  Code   Gb  D.o [1]
D::D() [subobject]           0x0800081b     0x2  Code   Gb  D.o [1]
D::RAZ_D()                   0x08000825     0x2  Code   Gb  D.o [1]
E::E(B *)                    0x08000853     0x2  Code   Gb  E.o [1]
E::~E()                      0x0800085d     0x2  Code   Gb  E.o [1]
GPIO_Init(GPIO_TypeDef *, GPIO_InitTypeDef *)
                             0x08000323    0x70  Code   Gb  main.o [1]
GenGlobalPool::InitGenGlobalPool()
                             0x080003f5    0x5e  Code   Gb  GenPool.o [1]
GenGlobalPoolInit            0x08000551     0x4  Code   Gb  GenPool.o [1]
GenPool::GenPool(unsigned long, unsigned long, unsigned long)
                             0x0800039d    0x58  Code   Gb  GenPool.o [1]
GenUtility::GenFree(void *)
                             0x080004cb    0x86  Code   Gb  GenPool.o [1]
GenUtility::GenMalloc(unsigned int)
                             0x08000457    0x74  Code   Gb  GenPool.o [1]
GlobalPool                   0x20000000    0x5c  Data   Gb  GenPool.o [1]
```

Figure 10 : Taille de code pour un gestionnaire GenPool

Comme le montre *la figure10*, la taille prise par le GenPool dans un projet est d'environ 530 octets. Cette taille est statique, ce qui permet d'instancier plus d'objets sans augmenter la taille finale du code.

Si nous faisons une différence globale entre le GenPool et le RamPool au niveau de la taille prise dans la ROM, on peut en conclure qu'à partir de 7 objets à allouer cette différence sera égale. (7 objets * 70 octets pour le Rampool = 490 soit environ la taille statique de GenPool)

En conclusion, on peut dire qu'à partir de 8 objets à instancier, GenPool deviendra plus avantageux, et cet avantage augmente avec le nombre d'instanciation à effectuer.

6.1.2 : Taille prise dans la RAM

	Programme en langage C++ Avec GenPool	Programme en langage C++ avec RamPool	ECART (%)
Taille (Octets)	**1 116**	**1 596**	**43**

Pour la RAM, le langage C++ avec un gestionnaire GenPool prend 43% de taille en moins que le langage C++ avec RamPool.

6.1.3 : Empreinte mémoire pour chaque objet :

	Langage C++ GenPool (Octets)	Langage C++ RamPool (Octets)	ECART (%)
Objet A	**126**	**126**	**0**
Objet B	**238**	**256**	**7,5**
Objet B1	**116**	**116**	**0**
Objet B2	**138**	**138**	**0**
Objet C	**10**	**148**	**1700**
Objet D	**12**	**12**	**0**
Objet E	**18**	**148**	**1480**
Total des objets	**658**	**944**	**43**
GenPool	**532**	**X**	**X**
Main	**860**	**772**	**11**

En ce qui concerne l'empreinte mémoire, le langage C++ avec héritage multiple avec gestionnaire RamPool à une empreinte moins conséquente de **43%** que l'héritage multiple avec gestionnaire GenPool.

Pour la grande différence qu'on peut remarquer pour les objets C et E, elle peut être expliquée par l'ajout de deuxièmes constructeurs pour initialiser le RamPool et le deuxième destructeur qui détruit les instanciations présentent dans le RamPool et sa remise à vide.

6.2 : Résultat vitesse d'exécution :

	Langage C++ GenPool	Langage C++ RamPool	Différence C++_GenPool/C++_RamPool
Création et initialisation [1]	**832 µs**	**970 µs**	**16,5**
Destruction [2]	**512 µs**	**164 µs**	**212**
RAZ [3]	**74 µs**	**74 µs**	**0**
Machine à états de A [4]	**94 µs**	**94 µs**	**0**
Machine à états de B [5]	**88 µs**	**88 µs**	**0**
Machine à états de B1 [6]	**112 µs**	**112 µs**	**0**
Machine à états de B2 [7]	**136 µs**	**136 µs**	**0**
Méthodes [8]	**44 µs**	**44 µs**	**0**
Boucle principale [9]	**548 µs**	**548 µs**	**0**
Temps d'exécution du code = [1] + 100 x [9] + [2]	**56,1 ms**	**55,93ms**	**0**

Concernant la différence qu'il y a pour la création et l'initialisation, elle peut être expliquée par le fait qu'on ajoute un constructeur en plus pour les classes C et E afin d'initialiser le RamPool (initialiser le tableau qui dans lequel on instancie les objets).

Lorsqu'on détruit un objet avec le GenPool, il fait appel au destructeur de l'objet et ensuite à la fonction Genfree() qui libère l'espace mémoire. Alors que le RamPool fait appel seulement au destructeur de l'objet ce qui explique la différence.

Les machines à états, appels des méthodes, et la remise à zéro n'ont pas de différence car nous les avons codés dans le même langage.

6.3 : Synthèse des résultats :

V : Avantage / = : Equivalent / X : Désavantage

Fonctionnalité	C++ (GenPool)	C++ (RamPool)	Commentaire
Héritage	V	V	Héritage possible car même langage de programmation pour les deux
Modification du modèle en cas d'héritage	V	V	Modification facile car les deux en C++
Taille ROM	X	V	C++ RamPool prend entre 3,1% et 11,6% de place en moins
Taille RAM	X	V	C++ RamPool prend moins de place au niveau de la mémoire
Le temps d'exécution du code	=	=	Equivalent
Initialisation et création des objets	V	X	C++ RamPool prend plus de temps car il a besoin d'initialiser le RamPool donc un constructeur en plus.
Destruction des objets	X	V	C++ GenPool plus long que le C car il doit faire appel a Genfree() en plus de destructeur.
RAZ des MAE et attributs	=	=	Equivalent
Taille du fichier binaire	V	X	Pour la méthode SEM_T le binaire prend plus de place
Code manuelle ajouté	V	X	On ajoute un peu plus de lignes du code pour réaliser le gestionnaire de mémoire RamPool que le GenPool.
Appel des méthodes	=	=	Equivalent
Exécution des machines à états	=	=	Equivalent
Instanciation d'un objet	V	X	En C++ gestion mémoire RamPool on a besoin d'avoir une variable d'état pour savoir si on a bien instancié un objet ou non.

6.4 : Conclusion :

Dans cette partie, un comparatif entre deux langages de programmation (C et C++) a été réalisé sur la base d'un modèle UML et en particulier les machines à états. Le comparatif s'est donc porté sur la comparaison de la taille du code produit à partir du modèle UML, ainsi que sur sa rapidité d'exécution. Les résultats vus dans cette partie, révèlent que le langage de programmation C++ est plus avantageux en terme de mémoire et de rapidité que le langage C orienté objet.

IV : Evaluation des modeleurs UML

1. Objectif :

L'objectif de ce chapitre est l'évaluation de quatre modeleurs UML choisis parmi tous les modeleurs existants. Les modeleurs évalués sont choisis sur une base de critères définis en fonction des besoins rencontrés lors du développement des projets au sein de Somfy.

Après une étude de recherche préliminaire regroupant tous les principaux modeleurs UML existants, il a fallu faire un choix pour en évaluer que quatre. Le choix est fait sur ces quelques critères qui sont fondamentaux pour les développeurs :

- Ils permettent de générer du code en C/C++ à partir des modèles UML
- Ils proposent la fonction de rétro-conception (lorsqu'on modifie le code généré cette fonction permet de synchroniser le modèle UML avec ce code)
- Ils permettent de créer tous les diagrammes UML(diagramme de classe, séquence, machine d'états…)

Les critères précédents ont permis de réduire suffisamment la liste pour permettre une évaluation plus précise des fonctionnalités de seulement quatre candidats.
Les outils de modélisation que nous considérons dans cette évaluation sont : Rhapsody 8.1.4 d'IBM, Enterprise Architec [7] de Sparx Systems, MagicDraw [8] de No Magic et enfin le plugin sur eclipse Papyrus UML [9].
Actuellement le modeleur UML utilisé au sein de Somfy est Rhapsody version 6.2 de chez IBM, dans le but de faire évaluer son outil de modélisation, cette étude réalisée va permettre de savoir s'il existe un autre outil plus récent qui pourrait répondre aux critères des projets de Somfy.

2. Critères

Les critères d'évaluation ont été définis par le besoin des développeurs, après plusieurs réunions avec eux, une première liste à vu le jour, cette liste est compétée en fonction des informations recueillies avec d'autres développeurs pour donner lieu à cette liste finale :
La fiche d'évaluation devra comporter ces différents critères :

- Plate forme Windows, Linux, Plateforme ?
- Open source ou non ?
- Coût de licence et maintenance
- Langages de programmation générés
- Langage de rétro-conception (modifie automatiquement le modèle UML à partir d'une modification du code)
- Simplicité de l'environnement de travail
- Gestion d'exigence
- Types de diagrammes pris en compte (Classe, machine à états, diagrammes de séquence, cas d'utilisateurs…)
- Animation et test des modèles UML (Animation des machines à états, diagrammes de séquences…)
- Rapidité de génération de code
- Générer de la documentation automatiquement à partir des modèles
- Importation et exportation des modèles sous format XMI (Format qui permet d'exporter ou importer un projet d'un outil à un autre)
- Importer par référence des modèles pour les partager sous SVN
- Qualité de la documentation support et forum présent
- Inclure ses propres commentaires lors de la phase de génération de code avec l'outil Doxygen.
- Inclure les diagrammes UML utilisé dans un projet lors de la génération de documentation.
- Génération de code scriptable (permet d'optimiser, customiser la génération de code)
- Générer de code à partir des diagrammes de séquence

Certains critères sont plus importants que d'autres et il est fortement conseillé de les pondérer. Pour cela, voir *figure11* où nous avons défini une matrice critère/critère qui met en évidence l'importance d'un critère par rapport à un autre. Un système de notation de chaque critère est mis en place. La notation porte sur la facilité de chaque modeleur à répondre au critère, la qualité de la réponse et enfin est-elle la réponse attendue par les développeurs au sein de Somfy.

	Plateforme /OS	Open source	Coût de la licence et sa maintenance	langages générés	langage de rétro-génération	IDE	Gestion d'exigences	Type de diagramme	Animation et test des modèles	Rapidité génération code	générer de la documentation	Import/Export XMI	Importer par référence des modèles pour les partager sous SVN	type de documentation /support/forum	Doxygen dans la génération de code	Diagramme dans Doxygen	Génération de code scriptable	Génération de code à partir des diagrammes de séquence	Somme
Plateforme/OS		0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Open Source	1		1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	3
Coût de la licence et maintenance	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Langage générés	1	1	1		1	1	1	1	1	1	1	1	1	1	1	1	1	1	17
Langage de rétro-génération	1	1	1	0		1	1	1	1	1	1	1	1	1	1	1	1	1	16
IDE	1	0	1	0	0		0	0	0	0	0	0	0	0	0	0	0	0	2
gestion d'exigences	1	1	1	0	0	1		0	0	1	0	0	1	1	1	1	1	0	10
type de diagramme	1	1	1	0	0	1	1		1	1	1	1	1	1	1	1	1	1	15
Animation,simulation et test des modèles	1	1	1	0	0	1	1	0		1	1	1	1	1	1	1	1	1	14
Rapidité génération code	1	1	1	0	0	1	0	0	0		0	0	0	0	1	1	1	0	7
Générer de la documentation	1	1	1	0	0	1	1	0	0	1		1	0	1	1	1	1	1	12
Import/export XMI	1	1	1	0	0	1	1	0	0	1	0		1	1	1	1	1	1	12
Importer par référence des modèles pour le partager sous SVN	1	1	1	0	0	1	0	0	0	1	1	0		1	1	1	0	1	10
type documentation support/forum	1	1	1	0	0	1	0	0	0	1	0	0	0		0	0	0	0	5
Doxygen dans la génération de code	1	1	1	0	0	1	0	0	0	1	1	1	1	1		0	0	0	9
Diagramme dans Doxygen	1	1	1	0	0	1	0	0	0	0	0	0	0	1	1		0	0	6
génération de code scriptable	1	1	1	0	0	1	0	0	0	0	0	0	1	1	1	1		0	8
Générer de code à partir des diagrammes de séquence	1	1	1	0	0	1	1	0	0	1	0	0	0	1	1	1	1		10

Figure 11 : Classement par ordre d'importance des critères

3- Evaluation :

Les étapes reprenant le travail d'évaluation des outils de modélisation UML sont indiquées *figure12*

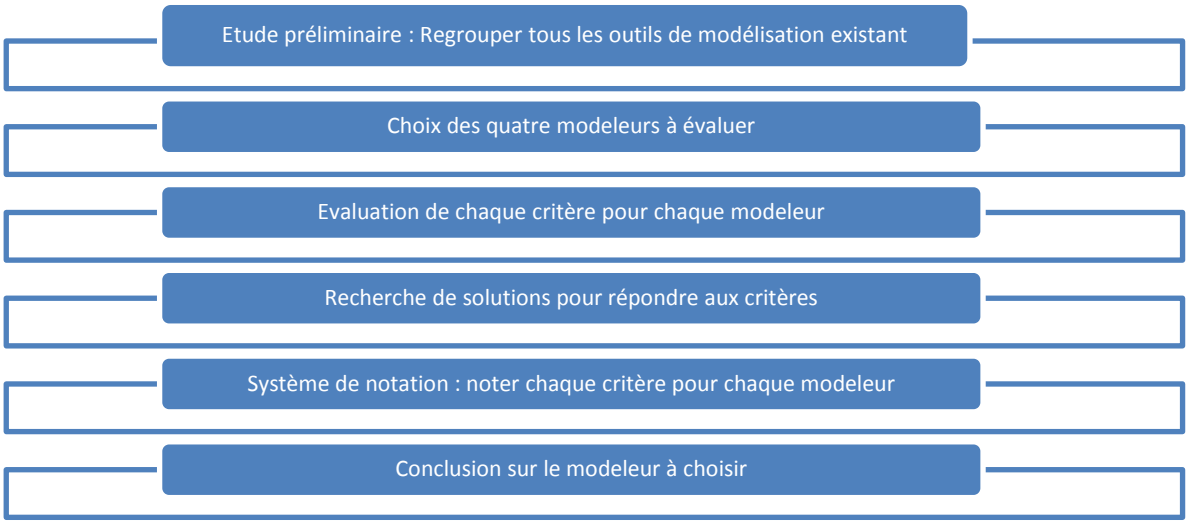


Figure 12 : Etapes pour l'évaluation des modeleurs

Après la phase d'étude préliminaire, qui consistait à définir les critères et le choix des outils de modélisation à évaluer, nous avons commencé par évaluer l'outil d'IBM Rhapsody version 8.1.4. Pour cela mon responsable de stage m'a planifié plusieurs réunions avec un expert de Rhapsody de chez IBM. Ces différentes réunions avaient pour but de m'initier à cette version, me montrer comment générer les langages de programmation C et C++ à partir d'un modèle UML. Dans un deuxième temps, pour pouvoir optimiser la génération de codes, il m'a montré les différentes solutions d'optimisation que Rhapsody proposait. Mais ce sujet sera détaillé dans la dernière partie de mon rapport.

3.1 : Evaluation de Rhapsody

Critères	Rhapsody 8.1.4	Notation
Créateur	*Telegic racheté par IBM*	*X*
Plateforme /OS	*Windows*	*X*
Open source	*Non*	*5*
Coût de la licence	*7000$ + 800 pour la maintenance*	*3*
langages générés	*C/C++*	*8*
langage de rétro-génération	*c/c++*	*8*
Intégrable dans	*Eclipse, VxWorks Workbench*	*8*
Gestion des exigences	*Rhapsody s'intègre aux systèmes de gestion des exigences et de la traçabilité tel que Reqtify déjà utilisé pour la version 6.2 utilisée actuellement*	*6*
Type de diagrammes	*Inclut tout type de diagramme UML ainsi que des diagrammes en SysML*	*10*
Animation et test des modèles	*L'animation des modèles et leurs debugs est intégré dans Rhapsody. Animation des machines d'états, diagramme de séquence…*	*8*
Rapidité génération du code	*Génération de code en C/C++ rapide et le reverse également. Code généré non optimisé mais possibilité de l'optimiser avec différentes méthodes (Properties, stéréotype, simplifier)*	*7*
Générer de la documentation automatiquement	*Génération sous format : HTML, PowerPoint, Word, RTF File, Text File.* *La génération est rapide et inclut tout type de diagramme*	*7*
Import/export des modèles en format XMI	*Dans la version 8.1.4 la fonctionnalité est bien proposée. Pour un projet volumineux, l'importation et exportation prend un certain temps (Info partagé par un collègue, car problème avec rhapsody sur ma machine, la fonctionnalité n'est pas proposée et donc en conséquence je n'ai pas pu tester...)*	*9*
Importation des projets pour les partages sous SVN	*Pour Rhapsody 8,1,4 l'importation par référence des modèles se fait de la même manière que le version actuelle utilisée chez Somfy*	*7*
Support/Forum	*Support site IBM,tutorial Youtube, support personnels d'IBM et Sodius.*	*8*
Doxygen dans la génération de code	*Deux propriétés (DescriptionBeginLine et DescriptionEndLine) spécifient le préfixe pour le debut et fin de lignes de commentaires dans le code généré. Doxygen cherche donc ces préfixes pour produire la Documentation*	*9*
Diagrammes dans doxygen	*L'utilisation de Graphviz permet la génération de diagrammes dans doxygen, cette méthode est plutôt simple à mettre en œuvre et le doxygen généré inclut toutes les informations voulues de notre projet*	*7*
Génération de code scriptable	*Pour le code scriptable, Rhapsody propose ce qu'ils appellent "Simplifier". Les Simplifiers permettent de customiser la génération de code et d'ajouter d'autres properties à des Patterns par exemple. Les Simplifier doivent être codé en Java.*	*8*
Génération de code à partir des diagrammes de séquence	*Possibilité de générer le code et l'éditer pour une rétro-conception à partir des diagrammes de séquence*	*7*
Génération automatique des diagrammes de séquence à partir des machine à états	*Lorsque le modèle est simulable, on peut générer automatiquement un diagramme de séquence à partir d'une animation d'une machine d'états par exemple.* *A partir de ce diagramme la génération de code est possible et la rétro-conception également.*	*8*
		7,05/10

3.2 : Evaluation d'Enterprise Architect :

Critères	Enterprise Architect	Notation
Créateur	Sparx Systems	*X*
Plateforme /OS	Windows/Linux	*X*
Open source	Oui	*8*
Coût de la licence	849$ Ultimate Edition Floating + 228$ maintenance	*6*
langages générés	C/C++ Possibilité d'exporter le projet sous mdxml pour générer le code avec UML Inside	*8*
rétro-génération	C/C++ et d'autres langages	*8*
Intégrable dans	Visual basic 2005 et Eclipse	*8*
Gestion des exigences	différentes solutions : Import/export dans Excel des exigences avec un fichier CSV. La possibilité de créer des exigences directement dans le modèle. Pour la traçabilité, on crée un événement "Change" interne à l'exigence, et toutes les modifications seront enregistrées dans la zone de saisie Description.	*6*
Type de diagrammes	Prend en compte tout les diagrammes UML ainsi que la création des méta-Modèles	*10*
Animation et test des modèles	Animation des modèles (animation des MAE, Diagramme de séquence) intégré directement dans EA, Possibilité de faire un debug également	*8*
Rapidité génération du code	Génération de code en C/C++ rapide et la rétro-conception également. Le code généré est non optimisé et ressemble de prés à celui généré par Rhapsody, mais ayant accès au Templates sa customisation devient plus facile que les autres modeleurs.	*7*
Générer de la documentation automatiquement	Génération de la documentation facilement en différents format (Docx, PDF, RTF). La durée de génération est importante pour des projets volumineux.	*7*
Import/export des modèles en format XMI	EA offre beaucoup de possibilités concernant les importations et exportations des projets par rapport à ses concurrents: Importation et exportation en XMI et d'autres formats très facile et rapide. Possibilité d'importer les modèles Rhapsody directement en *.rpy pour les versions > 8, Possibilité d'importer et exporter des modèles entre Eclipse et EA grâce au Plugin ModelBus	*9*
Importation des projets pour les partages sous SVN	Possibilité d'importer et exporter des modèles sous SVN sous format XMI directement depuis EA, Ceci est fait par la configuration des options des paquetages	*7*
Support/Forum	le site de sparxsytems fournit beaucoup de tuto vidéo (support très utile lorsqu'on débute avec le modeleur), Et une communauté assez importante sur les différents forums.	*8*
Doxygen dans la génération de code	Implémentation des commentaires dans le code grâce à la création des modèles de commentaire Doxygen, ou la modification des Template de base. EA inclut différents Template (Classe, attributs…) qui peuvent être modifié pour ajouter les commentaires dans le code.	*9*
Diagrammes dans doxygen	Utilisation de Graphviz en association avec doxygen, ceci permet la génération des diagrammes dans documentation	*7*
Génération de code scriptable	Le générateur de code est en open source, le script qui permet de générer de code en C/C++ est modifiable par l'utilisateur. A partir de ce script nous avons la possibilité de customiser la génération de code.	*8*
Génération de code à partir des diagrammes de séquence	la génération de code à partir des diagrammes de séquences est possible ainsi que la rétro conception	*7*
Génération des diagrammes de séquence à partir des machine à états	On peut générer un diagramme de séquence à partir de la simulation d'une machine à états. Ce diagramme généré automatiquement peut être simulé, mais on ne peut pas générer de code à partir de celui-ci.	*8*
		7,7/10

3.3 : Evaluation de MagicDraw :

Critères	MagicDraw	Notation
Créateur	No magic	*X*
Plateforme /OS	Windows	*X*
Open source	Oui	*4*
Coût de la licence	1299$ la licence + 199$ de maintenance + prix de certains Plugins en plus	*5*
langages générés	Seulement C++. Mais compatible avec le générateur de code UML Inside (*.mdxml) pour le langage C	*7*
langage de rétro-génération	Java, C++. Compatible avec UML Inside (*.mdxml)	*7*
Intégrable dans	Autonome	*6*
Gestion des exigences	Utilisation de diagramme d'exigence qui est une extension de SysML. SysML permet également de suivre la traçabilité de l'exigence.	*5*
Type de diagrammes	Tous les diagrammes sont pris en compte sauf le diagramme de timing	*9*
Animation et test des modèles	Les modèles peuvent être testé et simuler en utilisant le Plugin Cameo Simulation Toolkit	*8*
Rapidité génération du code	Temps de génération de code plus long que les autres modeleurs pour le générateur de code intégré. Mais compatible avec UML Inside qui générer de code en C/C++ rapidement.	*6*
Générer de la documentation automatiquement	Génération de la documentation facile, et en différents format (Doxc, pptx, ods, xlsx). Les données générées dans la documentation sont minimales par rapport aux documents des autres modeleurs.	*5*
Import/export des modèles en format XMI	Exportation des fichiers en format uml 2.5 XMI facile et rapide. Possibilité d'exporter aussi en fichier ReqIF, eclipse UML 2 (v2.x XMI à v5.x XMI) Importation des fichiers en CSV files, UML 2.1/2.5 XMI, Enterprise Architect, et enfin eclipse UML2	*7*
Importation des projets pour les partages sous SVN	Installation de Plugin TeamWork Server qui permet d'associer MagicDraw à SVN. Ce qui permet d'importer et exporter les projets directement depuis MagicDraw.	*7*
Support/Forum	Peu de forums sur ce modeleur, mais un support sur le site en live pour les différentes questions. Pour les questions techniques, on est directement guidé vers un expert, et la réponse est très rapide.	*5*
Doxygen dans la génération de code	Utilisation des balises @see, la documentation doit être sous la forme de "@see commentaire", doxygen on détectant cette balise peut de générer l'élément commentaire dans la documentation	*6*
Diagrammes dans doxygen	De la même maniére que les autres modeleurs, les diagrammes sont générés dans doxygen en utilisant Graphviz. La démarche à suivre reste la même.	*7*
Génération de code scriptable	Le modeleur ne permet pas la génération de code scriptable, ni la customisation de son générateur de code. (Prise de contact avec le support pour la vérification de l'information).	*X*
Génération de code à partir des diagrammes de séquence	La génération de code à partir des diagrammes de séquence n'est pas une fonction présente dans ce modeleur.	*X*
Génération automatique des diagrammes de séquence à partir des machines à états	La génération de diagramme de séquence n'est pas une fonctionnalité présente sur ce modeleur	*X*
		6,26/10

3.4 : Evaluation de Plugin sous Eclipse : Papyrus UML

Critères	PapyrusUML	Notation
Créateur	CEA	*X*
Plateforme /OS	Multiplate forme (Java/Eclipse)	*X*
Open source	Oui	*7*
Coût de la licence	Eclipse Public License	*8*
langages générés	C++. Possibilité de coder et intégrer son propre générateur de code pour le langage C	*5*
langage de rétro-génération	Java (je n'ai pas trouvé de Plugin permettant de faire de reverse en C,C++)	*5*
Intégrable dans	Eclipse	*5*
Gestion des exigences	Concernant les diagrammes d'exigence et les diagrammes paramétrés, nous avons Papyrus qui supporte le SysUML. SysUML permet de formaliser les exigences, lier un modèle à une exigence lors de la phase de développement, et suivre sa traçabilité.	*5*
Type de diagrammes	Inclut tout type de diagramme UML et possibilité de créer ses propres diagrammes	*10*
Animation et test des modèles	La possibilité de faire un Debug, la génération d'un exécutable et sa simulation. On ne peut pas simuler les machines à états par exemple ou les diagrammes de séquence	*5*
Rapidité génération du code	Génération de code en C++ rapide mais ne génère rien automatique, faut créer le destructeur, constructeur… Génération de code plus complexe au niveau de la démarche que les autres modeleurs (Installation des Plugins et environnement pas facile à prendre en main)	*5*
Générer de la documentation automatiquement	De base, Eclipse permet la génération des diagrammes en PNG,SVG,GIF… Installation d'un plugin "Gendoc" pour générer de la documentation (*.docx et .odt)	*6*
Import/export des modèles en format XMI	Pour pouvoir exporter un modèle de Papyrus à un autre modeleur, la réponse donnée par les développeurs de papyrus est de renommer le fichier *.uml en *.XMI (Méthode qui s'avère hasardeuse) ModelBus un Plugin payant permet d'importer et exporter des modèles entre Papyrus et Enterprise Architec de manière facile et rapide	*6*
Importation des projets pour les partages sous SVN	Installation d'un Pulgin Subclipse SVN qui permet de relier un projet PapyrusUML à un repository SVN, une fois la configuration finie le programme exécute automatiquement un checkout du projet	*8*
Support/Forum	support site Eclipse + papyrusUML. Quelques forums pour les installations des différents plugins et leurs intégrations à Eclipse. Pas énormément de tutoriels vidéo sur les différents sites (Youtube,…)	*7*
Doxygen dans la génération de code	Pour les commentaires dans la génération de code avec Doxygen, il faut installer le Plugin Eclox qui permet l'intégration de la documentation du code sur eclipse	*6*
Diagrammes dans doxygen	le même Plugin (Eclox) se connecte directement à Doxygen et ainsi permet de générer de la documentation (HTML, LateX, XML, Man Pages, Rich Text Format). Dans Doxygen les diagrammes sont inclus grâce à GraphViz.	*6*
Génération de code scriptable	PapyrusUML étant un plugin intégrable dans Eclipse, il permet la génération de code scriptable	*5*
Génération de code à partir des diagrammes de séquence	Génération de code possible à partir des diagrammes de séquences en C++ et Java. Concernant le langage C, Eclipse ne permet pas la génération de code en C.	*4*
Génération automatique des diagrammes de séquence à partir des machines à états	La génération de diagramme de séquence n'est pas une fonctionnalité présente sur le Plugin PapyrusUML	*X*
		6,05/10

Pour pouvoir réaliser cette étude, nous avons créé différents diagrammes UML permettant d'évaluer un ou plusieurs critères. On a annexé deux exemples de diagramme utilisé sous deux outils de modélisation. Cette annexe met en évidence l'environnement de travail des deux outils (Rhapsody et entreprise Architect).

Annexe 8 : Environnement de travail des modeleurs évalué :

4. Conclusion :

Notons d'abord que parmi les outils de modélisation que nous n'avons pas retenu parmi nos quartes finalistes, plusieurs présentent des fonctionnalités importantes pour certains critères. Mais nous n'avons pas pu les évaluer, car le critère éliminatoire est la génération de codes en C et C++.

D'autre part, l'évaluation technique de nos quatre outils finalistes a mené au classement suivant :

1. **Entreprise Architect** avec une note de **7.7/10 :**

Outils de la société australienne Sparx Systems payant mais peu cher, qui propose un environnement de travail très facile en termes de prise en main. Il permet de couvrir par ses nombreuses fonctionnalités l'ensemble des étapes du cycle de conception. Permet de répondre à tous les critères attendus par les développeurs au sein de Somfy à savoir : Gestion des exigences, génération de code en C et C++, Simulation et test des modèles, génération de documentations en différents formats (word, ppt…), importation exportation sous plusieurs formats (XMI, RPY…) ….

2. **Rhapsody version 8.1.4** avec une note de **7.05/10** :

Outils de la société IBM avec une licence payante assez chère, qui propose un environnement de travail qui peut être difficile au début en termes de prise en main. Tout comme Enterprise architect, il permet de couvrir l'ensemble des étapes du cycle de conception et répond malgré certains inconvénients à presque tous les critères. La génération de codes en C et C++ est similaire à celle proposée par son concurrent Enterprise Architect, et le code généré peut être modifié dans le modèle ou avec l'Environnement de Développement Intégré associé (Rhapsody se charge de mettre à jour le modèle). Mais les méthodes de customisation et l'open source de l'outil restent plus complexe qu'entreprise architect. De plus, l'importation et exportation sous format XMI durent beaucoup plus longtemps avec cet outil. Ces inconvénients justifient l'écart de notation entre les deux outils.

3. **MagicDraw** avec une note de **6.2/10 :**

MagicDraw, outils de modélisation payant de la société No Magic offre des commandes intuitives au sein d'une interface graphique très bien conçue. Il fournit un mécanisme d'ingénierie qui intègre de nombreux produits tels que l'intégration des exigences, les tests des modèles, bases de données…
Certaines fonctionnalités ne sont pas proposées de base, il faut acheter et installer des Plugins pour pouvoir en profiter (exemple : test et animation des machines d'états). Le fait que l'outil ne permet pas de générer des diagrammes de séquence à partir des animations des machines à états, la génération de codes scriptables et la complexité de certaines fonctionnalités justifient la troisième place.

4. **PapyrusUML** avec une note de **6.05/10 :**

Outil de modélisation gratuit sous Eclipse, il présente l'avantage d'exportation de modèle, éditer les diagrammes et de créer ses propres diagrammes en dehors de ceux de base. Etant un Plugin sous Eclipse, il permet d'exporter et importer des projets réalisés sous Enterprise Architec ou Magicdraw. Ce qui peut justifier sa dernière place est la complexité de génération de codes. Il ne permet pas la génération de codes en C, mais offre la possibilité d'en développer un et de l'intégré à Eclipse. Le manque de génération de code à partir des diagrammes de séquence, et l'ensemble des plugins que l'utilisateur doit intégrer à Eclipse pour répondre à certains critères sont un ensemble qui justifie la dernière place de cet outil.

Après l'évaluation technique sur tous les plans, nous avons donc identifié en Enterprise Architect un outil de modélisation qui pourrait convenir au développement de projet au sein de l'entreprise Somfy.

V: Evaluation des générateurs de codes

1. Objectif :

Après avoir évalué les modeleurs sur tous les plans techniques en fonction des critères définis, nous allons maintenant nous intéresser aux générateurs de codes que ces modeleurs intègrent. Avec l'évolution rapide des standards UML, les modeleurs UML essayent de suivre la cadence et assurent désormais outre la conception des modèles UML la génération automatique du code à partir des modèles conçus. Cette partie sera la suite logique de l'évaluation des modeleurs, nous allons évaluer les différentes méthodes que les modeleurs proposent concernant l'optimisation de leurs générateurs de code.
Le but est d'étudier les différentes solutions mises en place par les modeleurs pour customiser leurs générateurs de codes et le générer comme cela se fait actuellement au sein de Somfy.
Avant toute chose dans un simple exemple, nous allons montrer la différence entre la génération de code optimisé et non optimisé. Pour cela, nous allons générer le code d'une machine à états avec UML inside (générateur utilisé au sein de somfy) et générateur intégré au modeleur Rhapsody.

La machine à états à partir de laquelle la génération de code sera faite est représenté figure 13

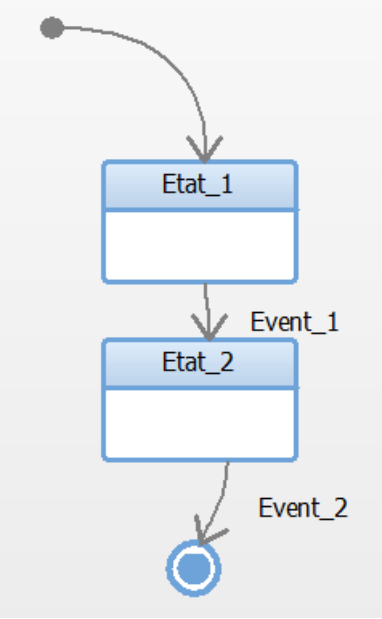


Figure 12 : Machine à états générée

Pour les codes générés, se référencer à l'annexe 9

2. Evaluation :

L'exemple fourni dans l'annexe 9 concernant les deux générations montre à quel point la phase d'optimisation du code généré est importante. Celle-ci permet de générer un code compact qui offre des meilleures performances temps réels, des empreintes mémoires moins conséquentes et une compréhension du code plus facile.
Cette évaluation va donc reposer sur les méthodes que chaque modeleur propose pour optimiser son générateur de code. En termes de démarche, cette évaluation suit l'évaluation sur les modeleurs. Nous allons évaluer les générateurs intégrés aux quatre modeleurs évalués précédemment. L'optimisation de la génération des machines à états est le point le plus important dans cette évaluation, ceci peut se justifié par la modélisation prépondérante de machines à états pour les projets au sein de Somfy.
D'autres critères sont ajoutés à ce point pour une meilleure évaluation :

- Possibilité d'optimiser le générateur de code ?
- Méthodes permettant l'optimisation
- Facilité de la rétro conception

2.1 : Rhapsody version 8.1.4 :

L'outil de modélisation Rhapsody intègre un générateur de code, qui permet la génération de plusieurs langages de programmation C, C++ et Java. L'outil dispose d'une interface qui permet d'éditer le code, et de le synchroniser par la suite grâce à la fonction de rétro conception avec le modèle UML.
Concernant l'optimisation de code généré, Rhapsody à mis en place trois méthodes :

- Modification des *properties* : le générateur de code référence de nombreuses propriétés qui peuvent être modifiables et optimisables selon les besoins de l'utilisateur. La figure 13 présente toutes les propriétés qui peuvent être optimisable lors de la phase de génération de codes, à partir de ces propriétés, on peut choisir par exemple la génération ou non des deux fonctions « accessors&mutator et setters », la génération de certains events des machines à états ou non ….

Voir annexe 10 pour un exemple

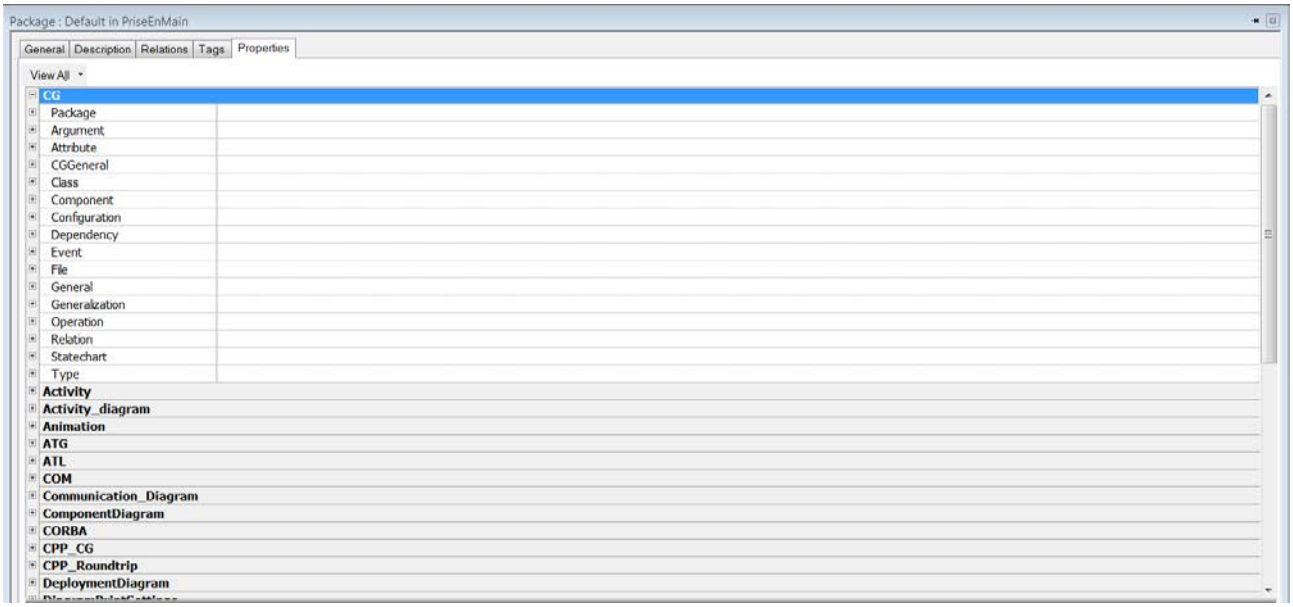


Figure 13 : Les propriétés de la génération de code

- *Stéréotype* : cette méthode permet à l'utilisateur de créer un stéréotype qui par la suite pourra être appliqué à tous les attributs, opérations, machine à états voulus. C'est un moyen qui facilite l'utilisation des *properties*. Le stéréotype regroupe toutes les propriétés de rhapsody, lorsque l'utilisation a personnalisé son stéréotype en fonction de ses besoins il peut être appliqué à tous les attributs, opération … . Ceci permet une optimisation de génération plus facile et représente un gain de temps.
- *Simplifier* : la méthode peut être expliquée par la figure 14 :

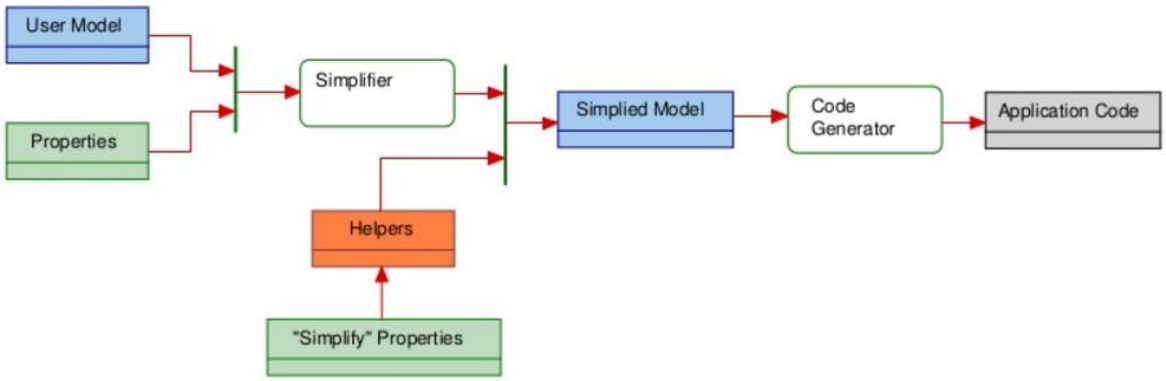


Figure 14: Démarche d'utilisation des Simplifier

Pour générer le code en C ou C++, Rhapsody suit le un processus particulier. Le générateur de code convertit temporairement le modèle UML conçu par l'utilisateur (user Model) en un modèle simplifié (Simplied

Model) et effectue la génération de code à partir de ce dernier. Le modèle simplifié est le modèle de l'utilisateur élargi auquel nous avons appliqué les propriétés de génération de code et les simplifiers que nous avons codés nous-mêmes. Le "« Simplify » Properties " est un modèle Rhapsody qui est fourni lors de l'installation de celui-ci, il donne accès au code des différentes propriètès, ce qui permet aux développeurs de coder leur propre façon de générer le code. Ces Simplifier sont codés en Java.
Les Helpers permettent de définir des aides pour contrôler la façon dont le « Simplied Model » est transformé et généré en langage de programmation voulu.

2.2 : Enterprise Architect :

Enteprise Architect est aussi un outil de modélisation qui permet la génération de codes en différents langages de programmation (C, C++, VHDL, Java). Son environnement de développement permet de générer, éditer le code et synchroniser le modèle uml avec le code source.
Pour optimiser la génération de code avec Enterprise Architect, l'outil va encore plus loin que rhapsody avec ses propriétés. Entreprise Architect donne accès au code scriptable sous forme de Template qui génère le code, ce code scriptable peut être modifié ou remplacer par un autre pour optimiser la génération des machines à états, attributs, opération…. Ce qui laisse à l'utilisateur le libre choix de générer le code comme il le souhaite. Ce qui constitue un avantage sur son concurrent Rhapsody.
Un autre avantage qu'Entreprise Architect propose est l'exportation des modèles UML sous format MDXML, format qui est compatible avec le générateur de code UML Inside déjà utilisé au sein de Somfy.
La figure 15 présente un des Template avec le code scriptable associé : la Template sur cette figure correspond au code qui permet la génération des machines à états que le modèle UML contient

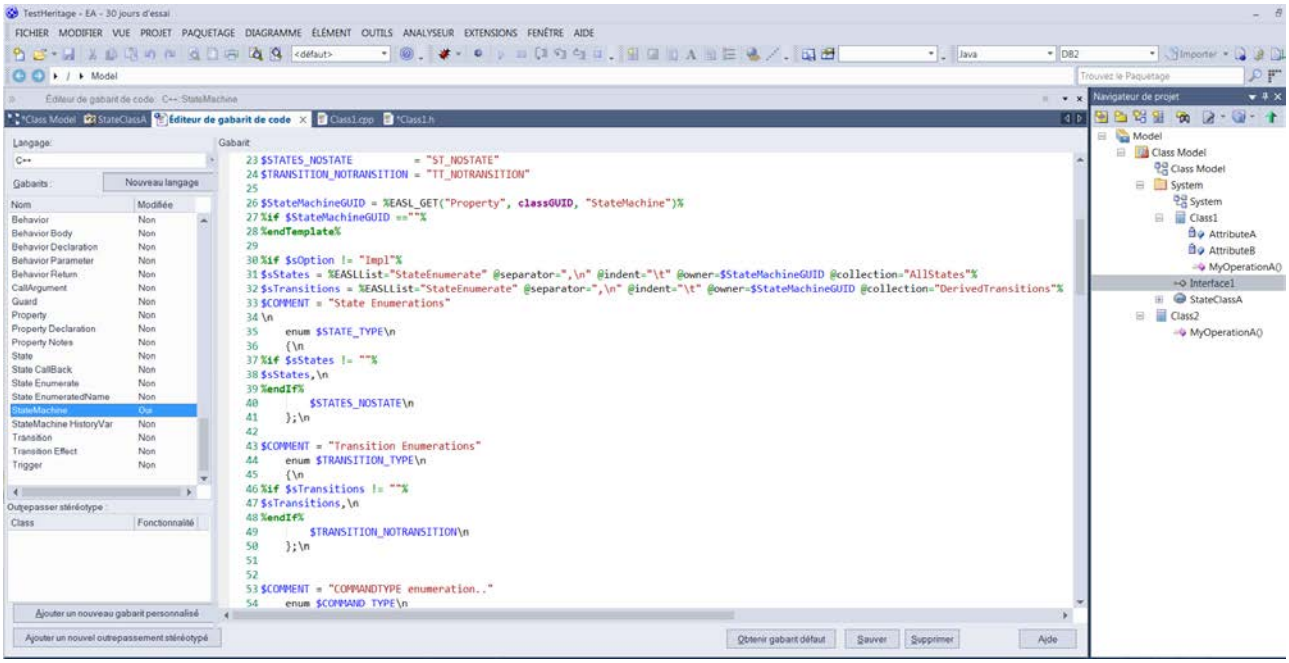


Figure 15 : Template des machines à états

2.2 : MagicDraw :

MagicDraw comme les deux autres outils vus précédemment permet de générer le code, éditer et effectuer la synchronisation des modèles avec le code à partir de son environnement de développement. Après plusieurs recherches et un contact pris avec l'équipe support de l'outil concernant les moyens mis en place pour optimiser le code généré. Il s'avère qu'aucun moyen n'est proposé pour optimiser sa phase de génération de code, ce qui présente un énorme désavantage par rapport aux autres outils.
Cependant malgré ce désavantage, MagicDraw permet lui aussi de générer des fichiers sous format MDXML pour pouvoir générer le code avec le générateur de code UML Inside. Mais certains éléments ne sont pas générés tel que le déroulement des machines à états dans les fichiers .cpp, ce qui doit être résolu par une mise à jour de générateur de code UML Inside pour pouvoir utiliser MagicDraw au sein de Somfy.

2.4 : Papyrus UML:

Le Plugin papyrus UML permet la génération de codes en C++, et sa rétro conception. La génération de codes est un peu particulière avec ce plugin, il ne génère aucune opération ni fonction automatiquement. Nous allons prendre pour exemple, la génération des constructeurs et destructeurs d'une classe. Pour que papyrus UML les génères il faut les créer sois même et les appliquer à la classe concernée, une fois cette étape réalisée il pourra les générer. La démarche est tout de même complexe et consommatrice en terme de temps, mais elle laisse à l'utilisateur le choix de générer ce dont il a besoin. Concernant l'optimisation des machines à états pour faire comme actuellement chez Somfy, la solution n'a pas été trouvée.

3 : Conclusion :

Cette partie est la suite logique de l'évaluation des modeleurs UML, la génération de codes est une phase très importante au sein de l'entreprise Somfy. Actuellement deux générateurs de code permettant de générer un code le plus compact possible et donc offre une consommation de mémoire la plus minime. De ce fait, il a fallu évaluer les différentes méthodes proposées par chaque modeleur pour optimiser son générateur de code intégré. Après cette évaluation, nous pouvons classer Enteprise Architect en première position. Car il donne accès au code scriptable des Template ainsi que la génération des fichiers en format MDXML compatible avec UML Inside. Ces deux solutions répondent clairement aux besoins pour générer le code comme cela est fait actuellement au sein de Somfy.

L'outil qui peut prétendre au changement d'outil de modélisation au sein de Somfy est Enterprise Architect. Lors des deux évaluations, à savoir celle des outils en fonction des critères définis et celle des générateurs de code intégré, il est celui qui propose ce dont les développeurs ont le plus besoins.

VI : Conclusion

L'objectif de ce stage est de répondre à trois questions pour le développement d'un projet important au sein du service métier logiciel :

- Quel langage de programmation entre le C et C++ utilisé pour le projet.
- Quel outil de modélisation choisir dans le futur pour les projets au sein de Somfy.
- Comment optimiser la génération de code à partir de ce nouvel outil.

Pour répondre à ces questions, nous avons tout d'abord réalisé un comparatif entre les deux langages de programmation a fin d'identifier le langage répondant au mieux aux contraintes de développement de la société Somfy. Ensuite nous avons réalisé une étude portant sur l'évaluation de quatre outils de modélisation ainsi que les solutions proposées pour l'optimisation du code généré.

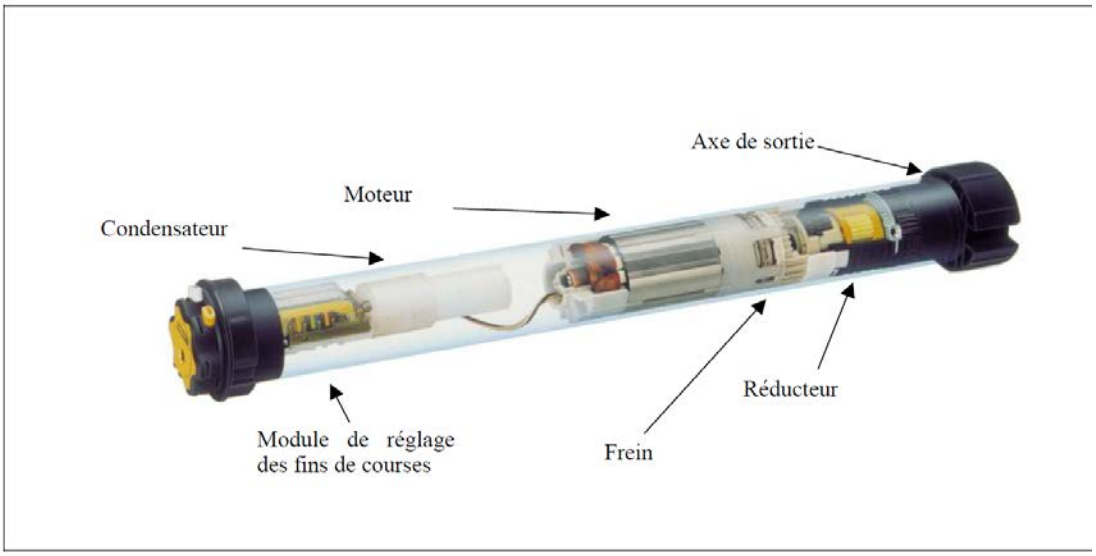
A ce stade de l'étude, la prochaine étape sera de développer un grand projet avec les réponses fournies durant mes six mois de stage. Le développement d'un projet est actuellement en cours au sein du service métier logiciel, une décision a déjà été prise en se basant sur mon stage qui est de coder ce projet en langage C++ et non en C_orienté_objet.

En ce qui concerne le plan professionnel et humain, je tiens à souligner que ce stage a été particulièrement profitable. En effet, c'est une expérience enrichissante du point de vue de la diversité des connaissances scientifiques et techniques qu'elle m'a apportée, notamment dans le domaine de l'ingénierie logiciel. Le fait d'être complètement intégré dans une équipe a été très bénéfique du point de vue de la connaissance de l'entreprise et de son fonctionnement.

J'ai également pu prendre conscience de l'importance de la communication et de la circulation des informations au sein d'une entreprise à travers les différentes réunions aux quelles j'ai pu assister. Le fait d'avoir intégré le site de Recherche et Développement de Somfy, m'a permis de côtoyer directement les développeurs et ainsi enrichir mon expérience sur mon futur métier.

ANNEXE

Annexe 1 : Exemple d'actionneur tubulaire Somfy



Annexe 2 : Exemple de commande électronique Somfy

Permet de commander les portails depuis les téléphones ou les tablettes des clients

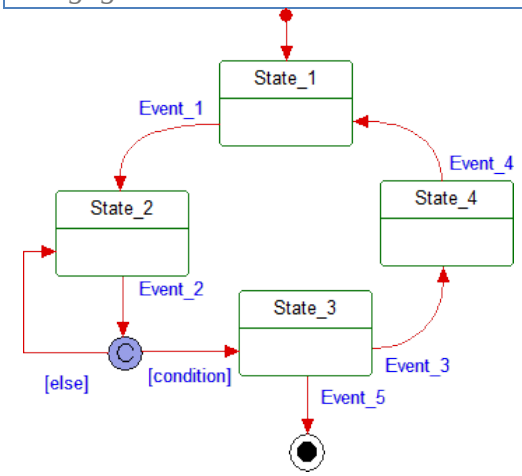
Annexe 3 – Exemple d'accessoire Somfy

Permet depuis son Smartphone ou sa tablette - de surveiller son domicile à distance à tout moment.

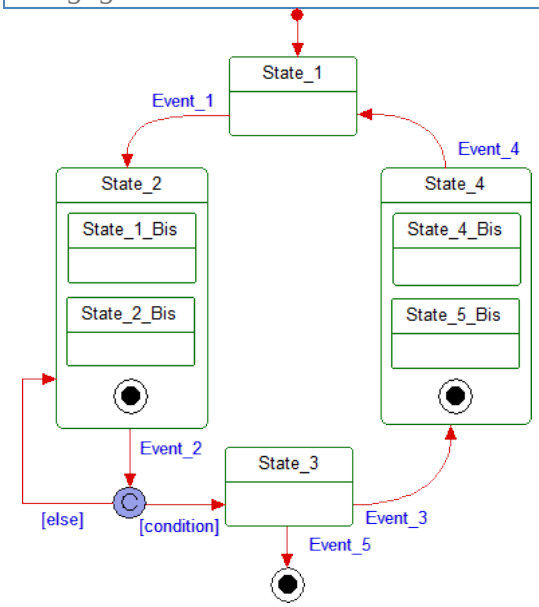
Annexe 4 : Machines à états :

4.1 : Machine à état de la classe A

Langage C



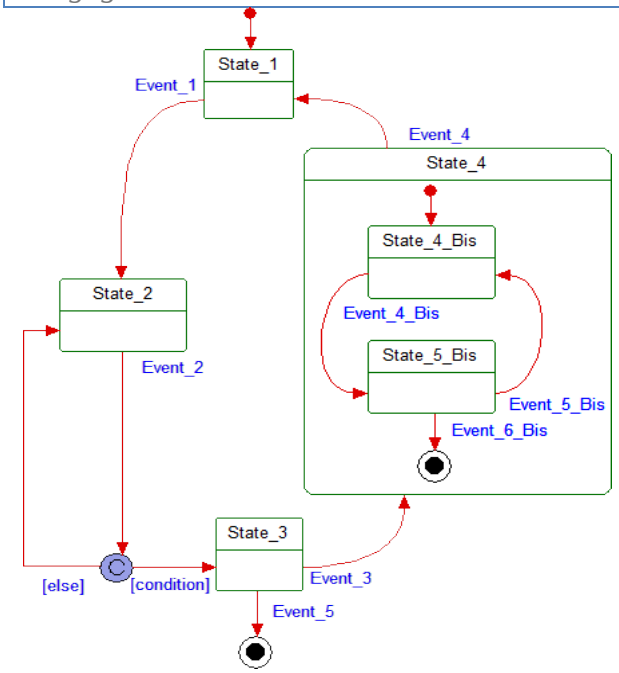
Langage C++



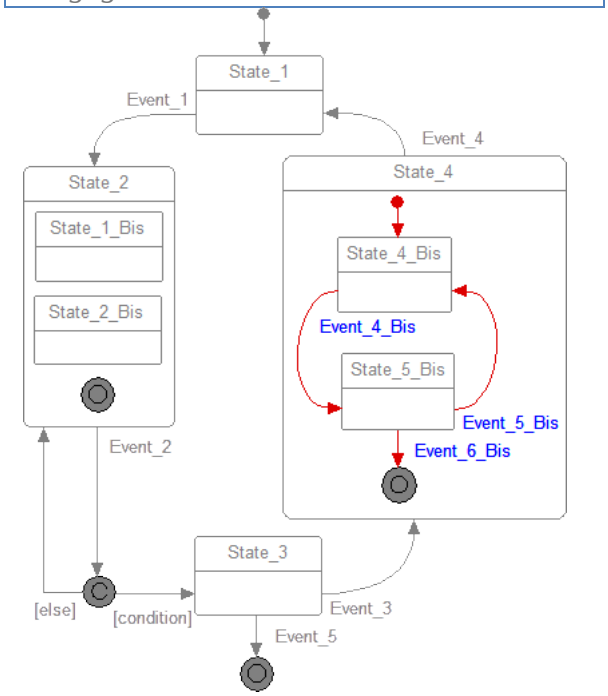
Pour la machine à état de la classe B, il s'agit d'un simple héritage de la machine à état de la classe A. c'est pourquoi nous n'allons pas la préciser ici

4.2 : Machine à état de la classe B1

Langage C

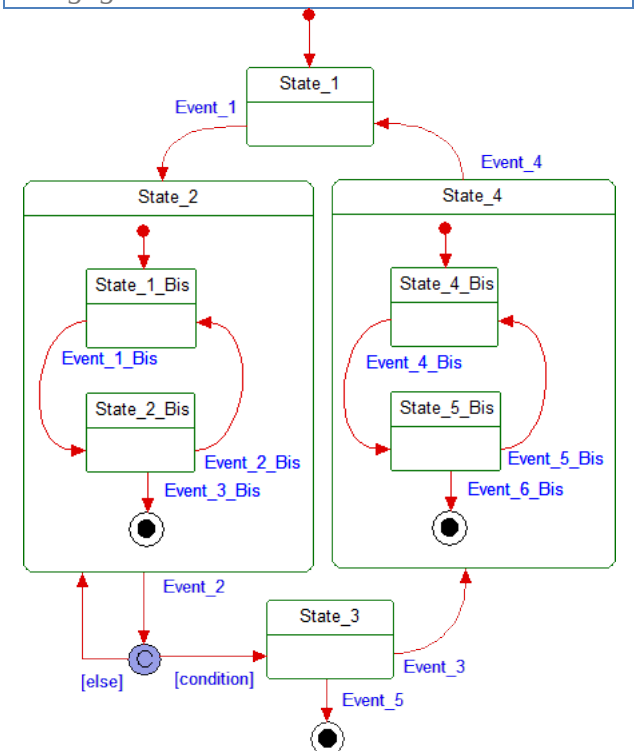


Langage C++

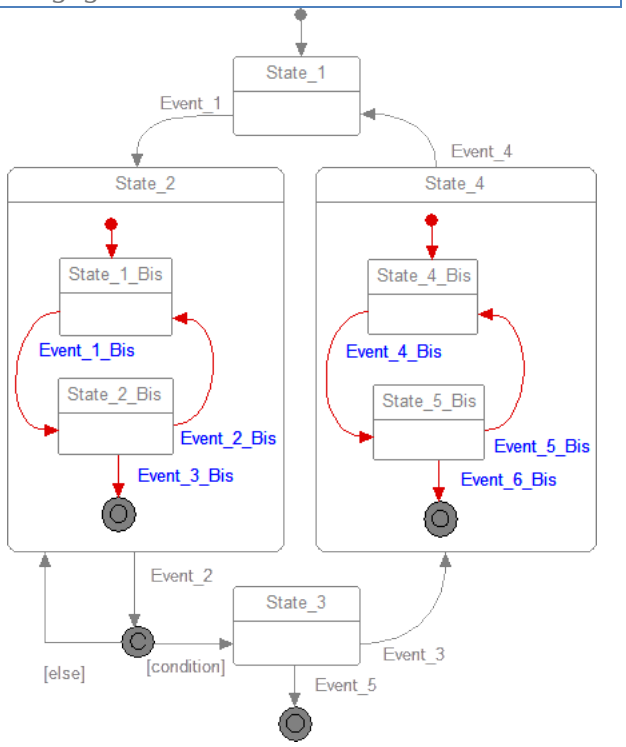


4.3 : Machine à états de la classe B2 :

Langage C



Langage C++



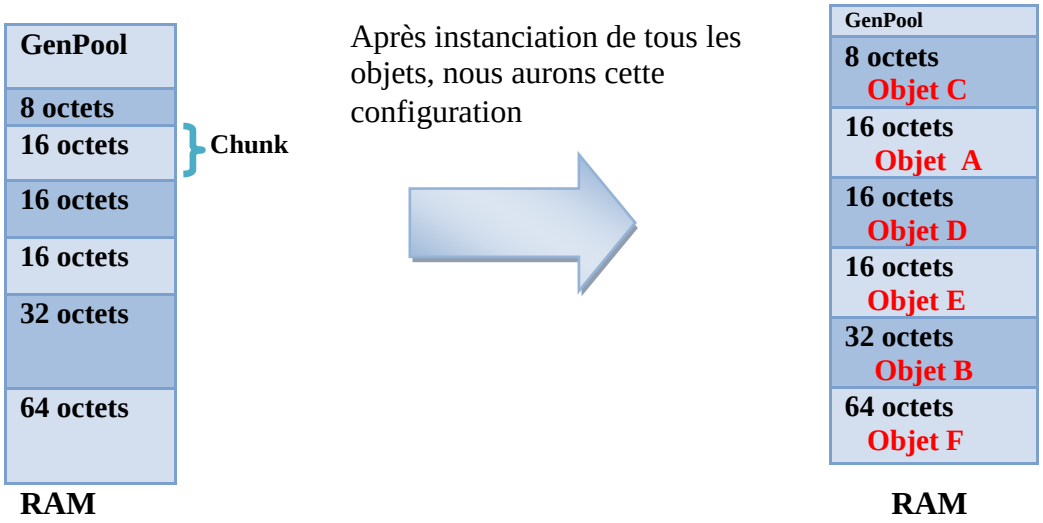
Annexe 5 – Utilisation de GenPool

Si nous souhaitons instancier 6 classes A,B,C,D,E et F de taille respective 16, 32,8,16,16 et 64 octets. Nous allons configurer notre GenPool de la sorte suivante : Nous allons déclarer 4 chunks :

Fichier *GenPool_config.h*

```
// Definition for GenPool semaphore Timeout
#define TIMEOUT_SEMAPHORE_GENPOOL   (0U)
#define GENPOOL_SEMAPHORE_ID        (GENPOOL_SEMAPHORE)
#define NB_OF_CHUNKS_8_BYTES                (1U)
#define NB_OF_CHUNKS_16_BYTES               (3U)
#define NB_OF_CHUNKS_32_BYTES               (1U)
#define NB_OF_CHUNKS_64_BYTES               (2U)
//Uncomment to throw an exception when we try to allocate outside the pools
#define BAD_ALLOC_IF_POOLS_FILLED (1)// If BAD_ALLOC_IF_POOLS_FILLED is not
defined, the GenPool will try to use the standard (non-overloaded) allocation methods
in the free RAM.

// Uncomment to add a counter of the maximum number of chunk requested in each pool
#define GATHER_STATS    (1)
// Number of different pools
#define NB_POOLS         (4)
// Pairs representing the size of a chunk and how many of them there is in each //
pool
// { {sizeChunk1,nbChunk1} , {sizeChunk2,nbChunk2} , ... }
#define CHUNKS          {\
                            {8,    NB_OF_CHUNKS_8_BYTES},    \
                            {16,   NB_OF_CHUNKS_16_BYTES},    \
                            {32,   NB_OF_CHUNKS_32_BYTES},   \
                            {64,   NB_OF_CHUNKS_64_BYTES}   \
}
// Total size of space need for the pool
// ( sizeChunk1*nbChunk1 + sizeChunk2*nbChunk2 + ... )/sizeof(TU8)
#define BUFFER_SIZE           (8    * NB_OF_CHUNKS_8_BYTES)      \
                            + (16   * NB_OF_CHUNKS_16_BYTES)     \
                            + (32   * NB_OF_CHUNKS_32_BYTES)     \
                            + (64   * NB_OF_CHUNKS_64_BYTES)
```

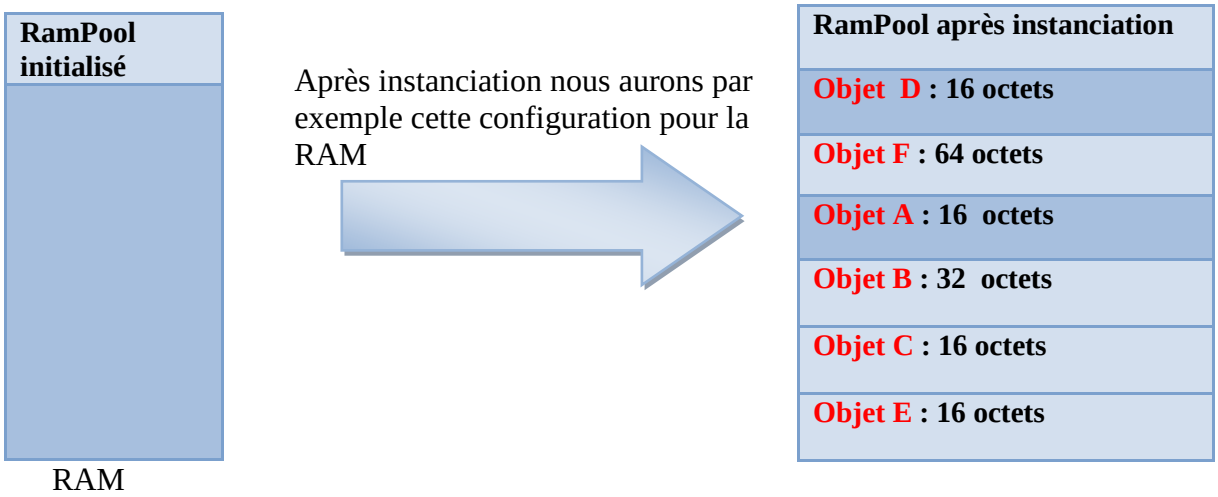
Après avoir configuré le GenPool, LA RAM est mutualisé sous forme de tableau de chunk déclaré précédemment (voir le code de GenPool_config.h) :



Cet exemple illustre la façon dont une allocation pseudo-dynamique d'un objet est effectué avec le GenPool. La manière qui consiste à déclarer soi-même les chunks, permet de mutualisé la mémoire et d'éviter la fragmentation de la mémoire en présence d'un objet de taille volumineuse à instancier.

Annexe 6 – Utilisation de RamPool

Concernant le RamPool, nous déclarons un tableau d'une certaine taille de mémoire dans lequel nous venons instancier nos objets. Ce tableau n'étant pas mutualisé, l'allocation des objets se fait de manière aléatoire entre les petits ou grands objets contrairement au GenPool ou c'est fait dans un ordre croissant. Ceci présente un inconvénient qui peut être la fragmentation de la mémoire en cas d'allocation d'un objet de grande taille.
Si nous prenons le même exemple que le GenPool, à savoir instancier 6 classes A, B, C, D, E et F de taille respective 16, 32, 8, 16, 16 et 64 octets: Dans un premier temps nous allons déclarer le tableau qui sera initialisé et par la suite nous allons instancier et allouer les objets de façon aléatoire sans mutualisé la mémoire :



Annexe 7 – Fichier *.map pour le comparatif C++/C

- **Fichier *.map* pour l'implémentation de l'héritage multiple avec un gestionnaire de mémoire GenPool en langage C++.**

STM32L152-EVAL_GenPool_Opt_Max.map.p

- **Fichier *.map* pour l'implémentation de l'héritage multiple avec un gestionnaire de mémoire RamPool en langage C++.**

STM32L152-EVAL_RamPool_opt_Max_Cpp.r

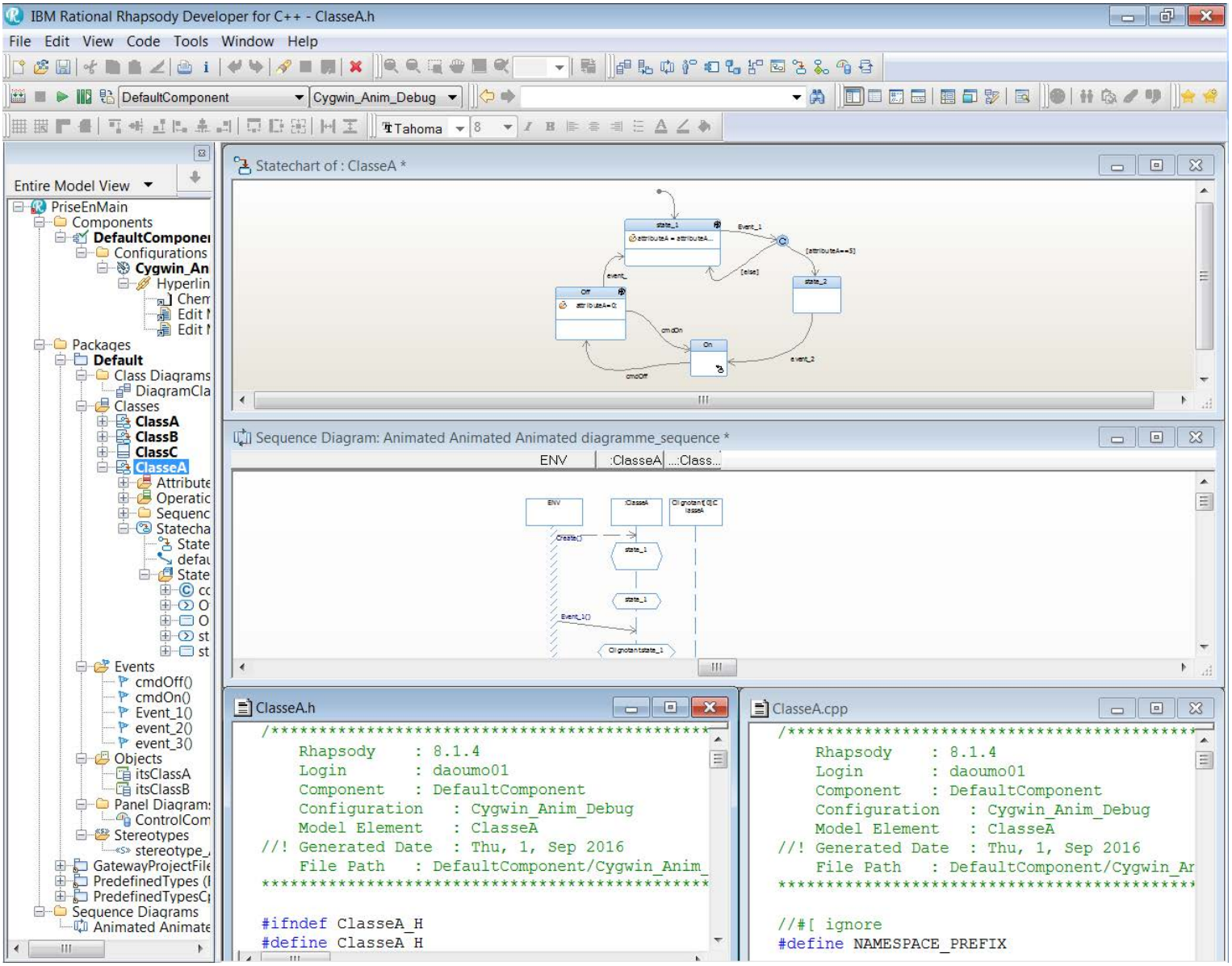
- **Fichier *.map* pour l'implémentation de l'héritage multiple avec un gestionnaire de mémoire RamPool en langage C_orienté objet.**

STM32L152-EVAL_Façon_SEM_T_opt_MAX

Annexe 8 : Environnement de travail des modeleurs évalué :

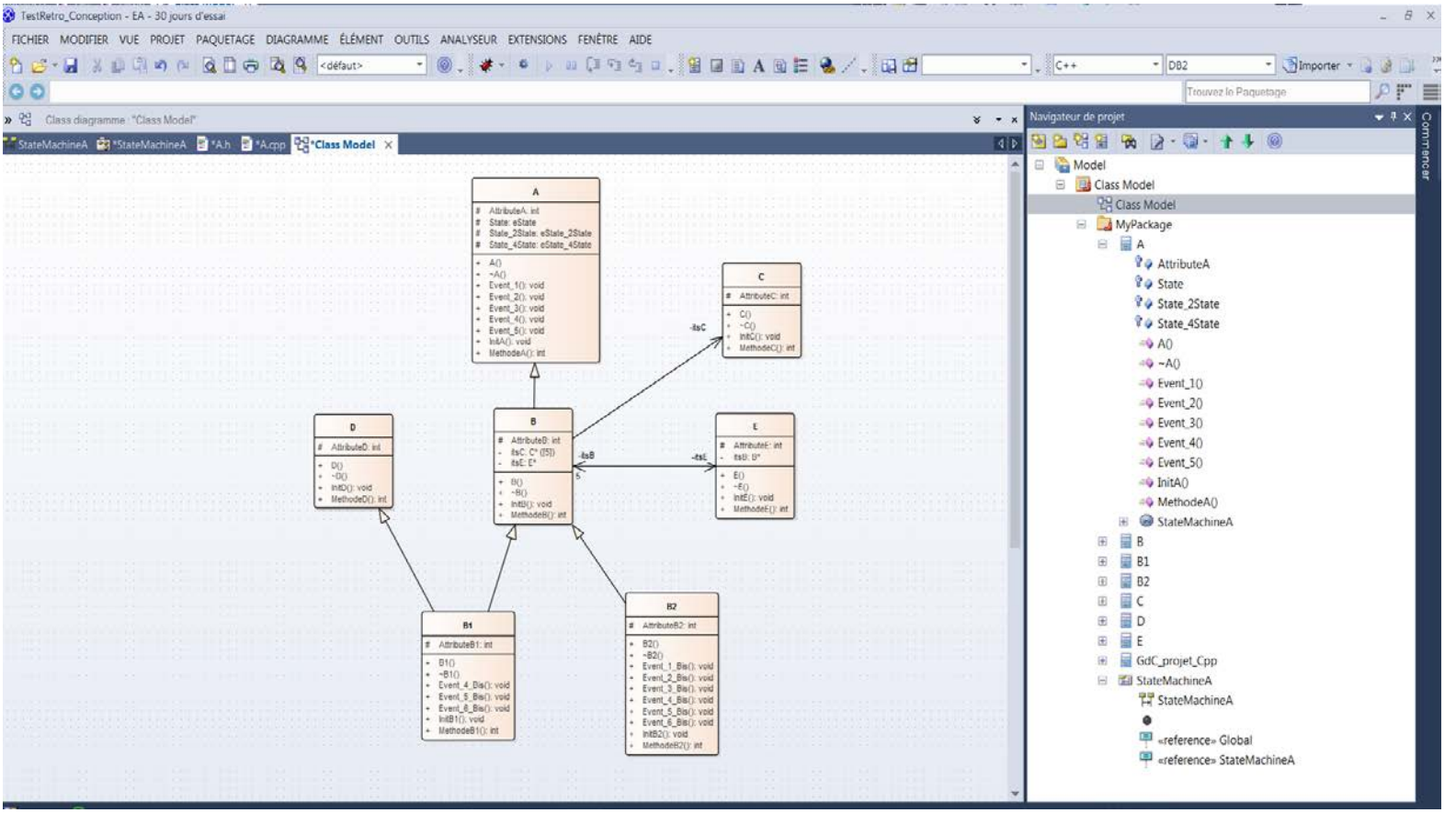
1. Rhapsody 8.1.4

Exemple de machine à états qui nous a permis d'évaluer le critère de génération de code en C et C++, puis la génération de diagramme de séquence à partir de l'animation de cette machine.



2. Enterprise Architect :

Exemple de modèle de classe utilisé pour l'évaluation de l'outil de modélisation Enterprise Architect.
Le modèle comporte sept classe avec des associations et de l'héritage. La classe A inclue la même machine à état de celle présenté sur l'outil de Rhapsody.



Annexe 9 : Différence entre un code optimisé de chez Somfy et un code généré sans optimisation :

Code optimisé et généré avec UML Inside utilisé chez Somfy

Fichier.h

```
// Protection
//-----------
#ifndef ExempleGeneration _H
#define ExempleGeneration_H
// Class declaration for bidirectionnal
associations
class ExempleGeneration;
// Static structure
//-----------------
class ExempleGeneration
{
    public:
    //Constructor
        ExempleGeneration();
   //Destructor
        ~ExempleGeneration();
    // Functions
        void Event_1(void);
        void Event_2(void);

    protected:
    private:
        // Types
        typedef enum
        {
            StateDefaut = 0,
            StateEtat_1,
            StateEtat_2,
            State_finale
        } eState;

        // Attributes
        eState State;

};

// End protection
//---------------
#endif // ExempleGeneration_H
```

Codé généré avec le générateur intégré à Rhapsody

Fichier.h

```
#ifndef ExempleGeneration_H
#define ExempleGeneration_H
#include <oxf/oxf.h>              //## auto_generated
#include "Default.h"              //## auto_generated
#include <oxf/omreactive.h>   //## auto_generated
#include <oxf/state.h>            //## auto_generated
#include <oxf/event.h>            //## auto_generated
//## class ExempleGeneration
class ExempleGeneration : public OMReactive {
   ////    Constructors and destructors    ////
public :
   //## auto_generated
   ExempleGeneration(IOxfActive* theActiveContext = 0);
   ~ExempleGeneration();     //## auto_generated
   ////    Additional operations    ////
   virtual bool startBehavior(); //## auto_generated

protected :
   void initStatechart();  //## auto_generated
   ////    Framework operations    ////

public :
   // rootState:
   inline bool rootState_IN() const; //## statechart_method
   inline bool rootState_isCompleted();//## statechart_method
   virtual void rootState_entDef();  //## statechart_method
   virtual IOxfReactive::TakeEventStatus rootState_processEvent();//##
statechart_method
   // State_finale:
   //## statechart_method
   inline bool State_finale _IN() const;
   // Etat_2:
   inline bool Etat_2_IN() const;  //## statechart_method
   // Etat_1:
   inline bool Etat_1_IN() const;  //## statechart_method
   ////    Framework    ////

protected :
   enum ExempleGeneration_Enum {
      OMNonState = 0,
      State_finale = 1,
      Etat_2 = 2,
      Etat_1 = 3
   };
   int rootState_subState;
   int rootState_active;
};
inline bool ExempleGeneration::rootState_IN() const {
   return true;
}
inline bool ExempleGeneration::rootState_isCompleted() {
   return ( IS_IN(terminationstate_3) );
}
inline bool ExempleGeneration::terminationstate_3_IN() const {
   return rootState_subState == terminationstate_3;
}
inline bool ExempleGeneration::Etat_2_IN() const {
   return rootState_subState == Etat_2;
}
inline bool ExempleGeneration::Etat_1_IN() const {
   return rootState_subState == Etat_1;
}
#endif
```

Fichier.Cpp avec le générateur de code UML Inside

```
// Internal dependency
//--------------------
#include "ExempleGe// Public implementations
//----------------------
void ExempleGeneration::Event_1(void)
{
    // State variable test
    switch (this->State)
    {
        case StateEtat_1 :
            // State updating
            this->State = StateEtat_2;
        break;

        case StateEtat_2 :
        case State_finale :
        default :
        break;
    }
}

void ExempleGeneration::Event_2(void)
{
    // State variable test
    switch (this->State)
    {
        case StateEtat_2 :
            // State updating
            this->State = State_finale;
        break;

        case State_finale :
        case StateEtat_1 :
        default :
        break;
    }
}
```

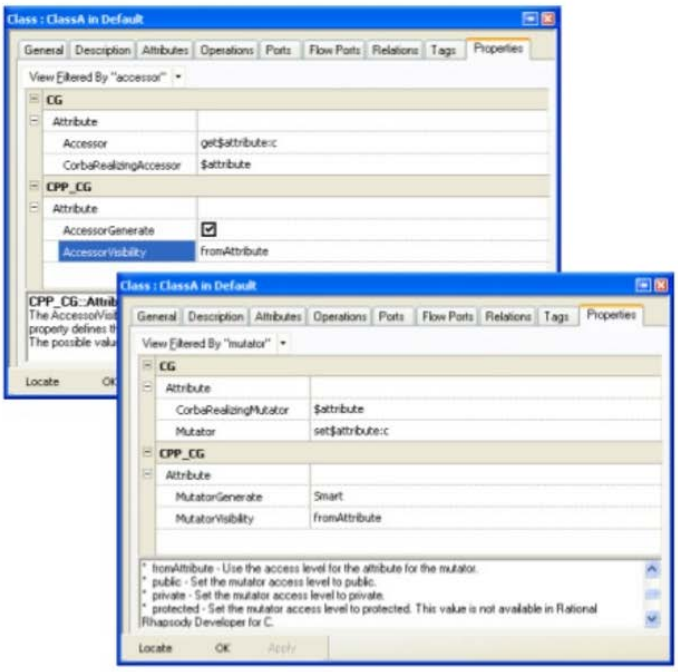
Fichier.cpp avec le générateur intégré à Rhapsody

```
//## auto_generated
#include <oxf/omthread.h>
//## auto_generated
#include "ExempleGeneration.h"
//## package Default
//## class ExempleGeneration
ExempleGeneration::ExempleGeneration(IOxfActive*
theActiveContext) {
   setActiveContext(theActiveContext, false);
   initStatechart();
}

ExempleGeneration::~ExempleGeneration() {
}

bool ExempleGeneration::startBehavior() {
   bool done = false;
   done = OMReactive::startBehavior();
   return done;
}

void ExempleGeneration::initStatechart() {
   rootState_subState = OMNonState;
   rootState_active = OMNonState;
}

void ExempleGeneration::rootState_entDef() {
   {
      rootState_subState = Etat_1;
      rootState_active = Etat_1;
   }
}

IOxfReactive::TakeEventStatus
ExempleGeneration::rootState_processEvent() {
   IOxfReactive::TakeEventStatus res = eventNotConsumed;
   switch (rootState_active) {
      // State Etat_1
      case Etat_1:
      {
         if(IS_EVENT_TYPE_OF(Event_1_Default_id))
            {
               rootState_subState = Etat_2;
               rootState_active = Etat_2;
               res = eventConsumed;
            }
      }
      break;
      // State Etat_2
      case Etat_2:
      {
         if(IS_EVENT_TYPE_OF(Event_2_Default_id))
            {
               rootState_subState = terminationstate_3;
               rootState_active = terminationstate_3;
               res = eventConsumed;
            }
      }
      break;

      default:
         break;
   }
   return res;
}
```

Annexe 10 : Exemple d'utilisation des *Properties* :

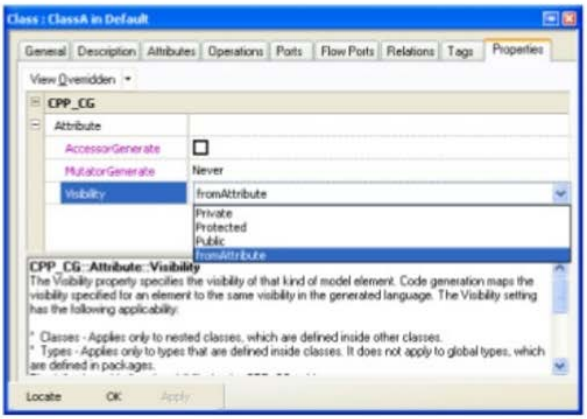
Considérons une classe avec un seul attribut (attribute1), par défaut Rhapsody génère les deux opérations accesor (getattribute1) et mutator (setattribute1) comme le montre la figure ci-dessus :


```
//## class ClassA
class ClassA {
    ////    Constructors and destructors    ////

public :

    //## auto_generated
    ClassA();

    //## auto_generated
    ~ClassA();

    ////    Additional operations    ////

    //## auto_generated
    int getAttribute1() const;

    //## auto_generated
    void setAttribute1(int p_attribute1);

    ////    Attributes    ////

protected :

    int attribute1;     //## attribute attribute1
};
```



Les développeurs au sein de Somfy n'utilisent pas ces deux opérations, nous avons cherché une propriété qui permet de les supprimer lors de la génération. Cette propriété se situe dans l'onglet attribute, deux cases à décocher « mutatorGenerate » et « accesorGenerate » permettent de répondre à notre besoin.
Ci-dessous le code généré après avoir décoché les deux propriétés :


```
//## class ClassA
class ClassA {
    ////    Constructors and destructors    ////

public :

    //## auto_generated
    ClassA();

    //## auto_generated
    ~ClassA();

    ////    Attributes    ////

    int attribute1;     //## attribute attribute1
};
```



RESUME :

J’ai effectué mon stage de fin d’études dans l’entreprise SOMFY à Cluses dans le département Haute Savoie au sein du service métier logiciel. L’intégration de ce service a comme mission de répondre à certaines questions sur la méthode d’ingénierie dirigée par les modèles. Pour ce faire, j’ai pu me confronter à différentes tâches réalisées par un ingénieur logiciel au sein de Somfy telles que la modélisation UML, la programmation, participation à des réunions…

Ce stage m’a permis de mettre en application les connaissances acquises lors de ma formation universitaire dans la qualité et m’a aussi donné la possibilité de travailler en équipe.

MOTS-CLES :

Modélisation, UML, modèles UML, Comparatif de langage, Evaluation modeleurs, diagrammes UML

ABSTRACT:

I realize my internship in the company Somfy in Cluses within the software engineering service. During this period, The integration of this service's mission is to answer some questions on the Model Driven Engineering . To do it, I was able to confront with different task realized by a software engineer within Somfy such as the modelling UML, the programming, the participation in meetings …

This internship allowed me to apply the knowledge acquired during my university education in the quality and also gave me the possibility of working as a team.

KEYWORDS :

Benchmark, UML, Unified Modeling Language, software engineering, UML diagrams

Bibliographie

[1] France, R. and Rumpe, B. *Model-driven Development of Xomplex Software* : A Research Roasdmap. IEEE Computer Society, Whashington, DC, USA, 2007, pages 6

[2] Laurent Piechocki. UML, *le langage de modélisation objet unifié*. *Laurent-piechnocki.developpez.com*/uml/tutoriel/Ip/cours/, 14 septembre 2009.

[3] OMG. *Unified Modeling Language* (OMG UML) *Superstructure*, Version 2.3. pages 7, 13 et 37.

[4] IBM Software Education. *UML and Rational Rhapsody WOrflows dor Embedded Code Development*, https://www.youtube.com/watch?v=yaLGw-ZSUKk, 19 décembre 2013

[5] Gabor Guta, *Model-to Text Transformation Modification by Examples*, Johannes Kepler Universitat, Linz, Avril 2012.

[6] Musset, J., Juliot, E., and Lacrampe, S. AcceleoTM 2.2 : *Guide utilisateur. Obeo, http ://www.acceleo.org/pages/accueil/fr*, 2008.

[7] Entreprise Architect, *version 12. : UML Tutoriel partie1,2 http://www.sparxsystems.fr/resources/tutorial/uml_tutorial2.html*, Mai 2016

[8] MagicDraw, Tutoriels sysml video magicDraw, *guide vidéo d'utilisation http://s1.ansoud.free.fr/sysml/tutoriel_sysml.html*

[9] Papyrus UML, Yassine Ouhammou, Mickael Baron : Apprendre les bases avec Eclipse papyrus, support internet sur le site developpez.com : *http://yassineouhammou.developpez.com/tutoriels/eclipse/uml-profil-papyrus/*