REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOHAMED BOUDIAF - M'SILA

FACULTE DE TECHNOLOGIE

DEPARTEMENT ELECTRONIQUE

N° :……..

FILIERE : ELECTRONIQUE

OPTION : SYSTEME EMBARQUE

Mémoire présenté pour l'obtention

Du diplôme de Master Académique

Par :

Delia bochra

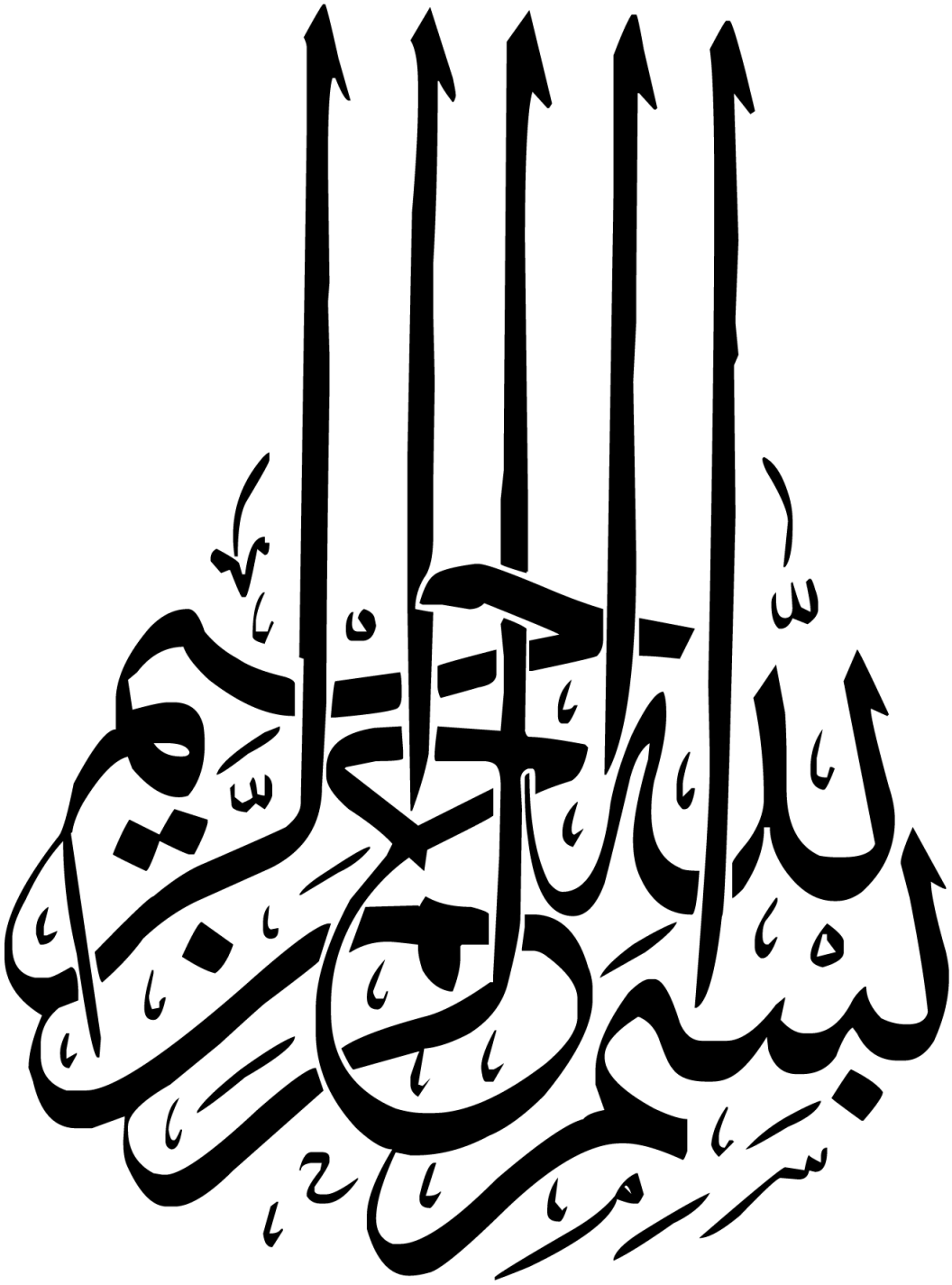
Intitulé

Conception d'un système embarqué à l'aide du langage UML

Soutenu devant le jury composé de:

Pr. KHADROUCHE Djamel	Université de M'sila	Président
Dr KAHLOUCHE Ahmed	Université de M'sila	Examinateur
Dr BENAHCENE MADANI	Université de M'sila	Encadreur

Année universitaire : 2020 /2021


بسم الله الرحمن الرحيم

Remerciements

En premier lieu nous tenons à manifester nos louanges à Dieu le tout puissant par aisance et excellence, de nous avoir donné la force et la patience de réaliser ce travail.

Je tiens particulièrement à remercier chaleureusement mon encadreur Dr. BENAHCENE Madani pour m'avoir guidé toute au long de ce travail.

J'adressemes remerciements mes remerciements aux membres de jury le Professeur KHADROUCHE Djameld'avoir fait l'honneur de présider le jury et le Docteur KAHLOUCHE Ahmed d'évaluer ce travail.

Je tiens à remercier également Mr le Chef de département de l'Electronique et tous les enseignants qui ont contribué à notre cycle de formation et aussi tout le personnel du Hall de Technologie pour leur serviabilité.

Je tiens à formuler ma gratitude et ma profonde reconnaissance à l'égard de mes parents pour leurs soutiens durant mes études. A tous les amis (es) qui m'ont aidéou contribué de près ou de loinà la réalisation de ce modeste travail.

Dédicace

Avant tous , je remecie dieu le tout puissant de

M'avoir donnè le courage et la patience pour rèaliser

Ce travail et

A la lumiére de ma vie ; ma mére .

A mon très chre père….

A mes sorurs et frères Abd kadar ; Mberak ; Nacera.

Mes copines Hanane ; fatima ; Dalila ;Sara et toutes les

Cpines de la chmbre z18

A mon mari Kamel Et toutes sa famille Chenaoua .

SOMMAIRE

LISTE DES FIGURES

LISTE DE TABLEAUX

LISTE DES ABREVIATIONS

Abréviations	**Notations**
UC	Use Case D
SD	State Chart Diagram
UML	Unified Modeling Language
SysML	System Modeling Language
STM	State Machine Diagram
IBD	Internal Bloc Diagram
BDD	Bloc Definition Diagram

Req	Requirement Diagram
Par	Parametric Diagram
Act	Activity Diagram
OMG	Object Management Group
MARTE	Modeling and Analysis of Real-Time and Embedded Systems

INTRODUCTION GENERALE

1. Introduction

Pour ressortir l'importance de l'ingénierie des systèmes on commence par relater une expérience scientifique vécue qui a tourné en catastrophe soulignant les risques liés à un projet mal défini.

Le vol inaugural d'Ariane 5 du 4 juin 1996 se solde par un échec. 40 secondes après le démarrage de la séquence de vol, le lanceur, qui se trouve à une altitude de 3700 mètres, dévie de sa trajectoire, se brise et explose. Des ingénieurs des équipes du projet Ariane 5 du CNES et de l'industrie commencent immédiatement à rechercher les causes de cet échec. Après enquête, les ingénieurs du CNES s'aperçoivent que, par mesure d'économie, le logiciel de navigation de la fusée Ariane 5 est celui qui avait été conçu pour Ariane 4 générant une incompatibilité entre le logiciel et le matériel. Tout tient à une seule variable : celle allouée à l'accélération horizontale. L'accélération maximum d'Ariane 4 a une valeur d'environ 64, la variable a été codée sur 8 bits. Mais, Ariane 5 est plus véloce : son accélération peut atteindre la valeur 300 (soit 1 0010 1100 en binaire et nécessitant 9 bits). La variable codée sur 8 bits a connu un dépassement de capacité. Ce dépassement a produit une valeur absurde dans la variable, ne pouvant correspondre à la réalité. Par effet domino, le logiciel face à des valeurs considérées comme anormales décide l'autodestruction de la fusée.

L'ingénierie des systèmes est une approche scientifique interdisciplinaire, dont le but est de formaliser et d'appréhender la conception et la validation de systèmes aussi vastes que complexes répondant à un ensemble précis d'exigences commerciales et techniques. Elle a pour objectif de maîtriser et de contrôler la conception de systèmes complexes. Par système, on entend un ensemble d'éléments humains ou matériels interdépendants les uns des autres et qui inter-opèrent à l'intérieur de frontières ouvertes ou non sur l'environnement. Les éléments matériels sont composés de sous-ensembles de technologies variées : mécanique, électrique, électronique, optique, matériels informatiques, logiciels, réseaux de communication, etc. L'aérospatiale et la défense utilisent l'ingénierie système depuis très longtemps. L'ingénierie système existe depuis les années 1940, mais elle n'est exploitée par la NASA que depuis les années 1990. C'est à ce moment-là que les fabricants ont commencé à transformer les produits ordinaires en systèmes intelligents, grâce au recours aux technologies de l'information. Le développement de produit entrait dans une nouvelle ère dans laquelle les logiciels occupent désormais le rôle principal.

Les efforts en ingénierie des systèmes embrassent l'ensemble du cycle de vie du système et leur mise en cohérence mobilise l'ensemble des corpus théoriques (sciences de l'ingénieur, sciences humaines, sciences cognitives, génie logiciel, etc.).

Il est clair que l'étude d'un système doit permettre de le comprendre sous un certain nombre d'angles différents pour pouvoir le construire. Dire qu'il faut le comprendre sous tous ses angles est bien sûr impossible, car tout système réel est régi par un nombre infiniment grand de variables que nous ne pouvons pas dénombrer. Le concepteur sélectionne alors, selon les limites de la science dont il dispose (ses compétences), selon le type du système étudié et ses finalités, les différents angles ou les vues sous lesquels il juge pertinent d'étudier le système. On peut ici citer Ross Ashby qui a défini le mot « système » en ces termes: « A System is a set of variables sufficientlyisolated to staydiscussablewhilewediscussit» [22]; Il met l'accent dans cette définition sur l'importance de l'abstraction dans la conception. Ceci implique qu'un modèle du système est spécifié (figure 1.1) car un modèle est une abstraction d'un système.

Une « simple » machine à laver le linge est l'association de composants divers : moteur électrique, poulie/courroie, électronique de gestion/programmation, tambour en tôle percé, électrovannes, capteur de niveau d'eau/température… Ces composants agissent ensemble de manière structurée pour laver un volume de linge, qui est la finalité. Cette finalité n'est visible que si la machine à laver est placée dans son environnement : arrivée d'eau, arrivée électrique, linge… et utilisateur. La machine à laver est un système qui n'a sa signification que lorsque ces éléments extérieurs sont réunis.

Au sens des sciences de l'ingénieur, un système complexe est un système intégrant des composants issus de domaines technologiques différents : électronique, mécanique, hydraulique, RF, optique, etc. Le comportement global du système émerge donc des interactions simples entre ses constituants, MAIS il est beaucoup plus riche que la somme des comportements individuels.

La machine à laver est un système complexe au sens des SI alors qu'un téléphone portable ne l'est pas. Une automobile, système mécanique au départ, est devenue complexe avec la présence de l'électronique de façon impressionnante.

Les méthodes de l'Ingénierie Système (IS) reposent sur des approches de modélisation et de simulation pour valider les exigences, et pour vérifier ou évaluer le système. La modélisation a donc couramment été utilisée pour l'IS, que ce soit pour des représentations concrètes avec des plans ou modèles réduits, ou plus abstraites avec des systèmes d'équations.

En général l'IS tend à modéliser les aspects suivants du système: décomposition fonctionnelle, flux de données, et décomposition structurelle. Exemples de techniques de modélisation employées :

• Le diagramme de flux de données (DFD ou Data Flow Diagram) pour définir les données traversant un système et leurs traitements éventuels

• Le diagramme de flux fonctionnel de bloc (FFBD ou Functional Flow Block Diagram) proche du diagramme UML d'activité ou du « flowchart »

La modélisation avec le langage UML est une pratique bien établie dans l'industrie logicielle. Bien que le langage UML permette par son caractère à usage général d'adresser de nombreux besoins pour l'IS, il est nécessaire d'adapter ce langage de modélisation par la définition de « profils UML ».

Le besoin de définir un langage basé sur UML pour l'IS a été initié en 2001 par l'organisation internationale

de l'ingénierie système INCOSE (International Council on Systems Engineering).
SysML (SystemsModelingLanguage) est basé sur UML et remplace la modélisation de classes et d'objets par la modélisation de blocs pour un vocabulaire plus adapté à l'Ingénierie Système. Un bloc englobe tout concept logiciel, matériel, données, processus, et même la gestion des personnes.
Comme représenté sur le diagramme suivant, SysML réutilise une partie d'UML, et apporte également ses propres définitions (extensions SysML) : 4 structurels, 4 dynamiques, et le diagramme d'exigences [20].
Structurels:
• Le « Block DefinitionDiagram » (BDD) remplace le diagramme de classes
• L' « Internal Block Diagram » (IBD) remplace le diagramme de structure composite
• Le diagramme de paquetage reste inchangé
• Le diagramme paramétrique est une extension SysML pour l'analyse de paramètres critiques du système
Dynamiques :
• Le diagramme d'activités est légèrement modifié pour SysML
• Les diagrammes de séquence, d'états, et de cas d'utilisations restent inchangés
Le diagramme d'exigences est une extension SysML.

Ainsi SysML permet de fournir un référentiel central qui supporte les analyses du système requises par l'IS, à savoir la décomposition fonctionnelle, les flux de données, et la décomposition structurelle.

2. Travaux dumémoire

Les travaux inscrits dans ce mémoire consiste à étudier l'approche orientée objet pour modéliser les systèmes embarqués. Nous avons étudié le langage UML (UnifiedModelingLanguage) et détaillé les 13 diagrammes qui permettent de modéliser les systèmes. Nous avons utilisé un autre langage Sysml (System ModelingLanguage) dérivé d'UML spécifiques aux systèmes industriels pour modéliser un cas d'étude qui est un lave-linge de capacité 5kg. Nous avons pour cela utilisé la suite logicielle MagicDraw pour créer les diagrammes d'exigences (req), de cas d'utilisation (uc), de séquence (sd), de définition de bloc (bdd), de bloc interne (ibd) et enfin d'état (stm). Ces diagrammes servent à évaluer, vérifier, valider les tests sur le produit en question durant son cycle de vie, depuis sa conception, sa réalisation, son exploitation et sa maintenance.

3. Structure du mémoire

Ce mémoire est organisé en trois chapitres comme suit:

Dansle**Chapitre1**,nousallonsexposerdans unpremiertemps une introduction aux systèmes embarqués en donnant quelques-unes de leurs définitions, leurs domaines d'application ensuite nous développerons leurs caractéristiques spécifiques qui permettent de définir les exigences à respecter lors de

leur conception et réalisation.

Dans le **Chapitre 2**, nous allons présenter le langage UML (UnifiedModelingLanguage) [3]. Nous définissons l'approche orientée objet dans la modélisation des systèmes embarqués et son avantage par rapport à l'approche fonctionnelle. Nous détaillons les diagrammes du langage UML sur différents exemples et montrons les structures des classes, des objets, des notions d'encapsulation, d'héritage et de polymorphisme.

Dans le **Chapitre 3**, Pour évaluer pratiquement les performances d'un système embarqué, nous avons traité un cas d'étude pour notre application qui est la carte de commande d'un lave-linge ménager. Pour cela nous avons utilisé la suite logicielle MagicDraw version 18.5 de démonstration et créé les diagrammes Sysml nécessaires pour la conception du système embarqué que nous avons choisi. Ces diagrammes serviront à contrôler l'exécution de la réalisation de la carte de commande suivant le diagramme des exigences qui a été établi. Le diagramme de séquence sert à vérifier le respect des contraintes temporelles. Le digramme d'état vérifie le respect de l'ordre des taches exécutées en fonctionnement.

Nous terminons par une conclusion générale et donnons quelques perspectives.

CHAPITRE I

Résumé :

Dans ce Chapitre, nous allons exposer en premier lieu une introduction aux systèmes embarqués en donnant quelques-unes de leurs définitions, leurs domaines d'application ensuite nous développerons leurs caractéristiques spécifiques qui définissent les exigences de leur conception. On termine par une conclusion

INTRODUCTION AUX SYSTEMES EMBARQUES

Sommaire du Chapitre:

1) Introduction :

Les systèmes embarqués sont des systèmes électroniques intégrant du logiciel et du matériel, inclus dans des objets ou des systèmes qui ne sont pas perçus comme des ordinateurs par leurs utilisateurs afin de fournir des fonctionnalités précises [13]. Actuellement, les systèmes embarqués sont présents dans des applications de plus en plus nombreuses, par exemple les cartes à puce, les systèmes mobiles communicants tels que les téléphones mobiles, l'automobile, l'avionique, les capteurs intelligents, la santé et l'électronique grand public.

La complexité croissante de la technologie de l'embarqué fait que le processus de développement des systèmes embarqués doit suivre une démarche rigoureuse basée sur une sémantique formelle pour consolider la description structurelle des composants de ces systèmes et évaluer au plus tôt leur comportement.

Un système embarqué est autonome et ne possède pas des entrées/sorties standards tels qu'un clavier ou un écran d'ordinateur. Contrairement à un PC, l'interface IHM (Interface Homme Machine) d'un système embarqué peut être aussi simple qu'une diode électroluminescente LED (*Light Emitter Diode)* qui clignote ou aussi complexe qu'un système de vision de nuit en temps réel; les afficheurs à cristaux liquides LCD (*Liquid Crystal Display)* de structure généralement simple sont couramment utilisés.

Afin d'optimiser les performances et la fiabilité de ces systèmes, des circuits numériques programmables FPGA (*Field Programmable GateArray),* des circuits dédiés à des applications spécifiques ASIC (*Application SpecificIntegrated Circuits)* ou des modules analogiques sont en plus utilisés.

Le logiciel a une fonctionnalité fixe à exécuter qui est spécifique à une application. L'utilisateur n'a pas la possibilité de modifier les programmes.

La **figure1-1** montre le schéma typique d'un système embarqué, on retrouve en entrée des capteurs généralement analogiques couplés à des convertisseurs Analogiques/Numériques, et en sortie des actionneurs généralement analogiques couplés à des convertisseurs Numériques/Analogiques. Au milieu, on retrouve le calculateur sous forme d'un processeur embarqué raccordé à des périphériques d'Entrées/Sorties. Il est à noter qu'il est complété généralement par une mémoire RAM ou/et ROM et d'un circuit FPGA jouant le rôle de coprocesseur afin de proposer une unité de calcul supplémentaire au processeur. Ou parfois il joue le rôle d'une unité d'entrée/sortie personnalisée pour le support d'une connectivité additionnelle pour le système embarqué.

Sur le schéma on peut remarquer la boucle d'un système d'asservissement entre les entrées et les sorties, c'est ce qui caractérise généralement les interactions des systèmes embarqués avec le monde extérieur. Il est à noter que l'expression qui découle de ce schéma peut inclure la majorité des classes des

systèmes embarqués. Pour illustrer ces propos, on peut donner l'exemple d'un système embarqué minimaliste avec comme capteurs des interrupteurs, et comme actionneurs des LED.

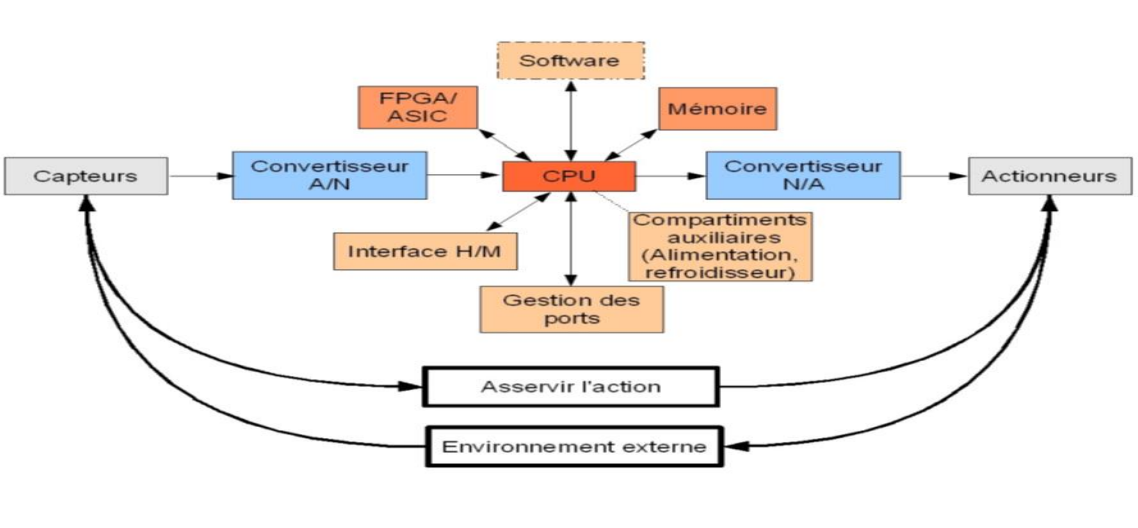


Figure I-1 *:* **Système embarqué typique.**

2) Domaines d'application des systèmes embarqués

Les systèmes embarqués sont désormais utilisés dans des applications diverses tels que : le transport (avionique, espace, automobile, ferroviaire), dans les appareils électriques et électroniques (appareils photo, jouets, postes de télévision, électroménager, systèmes audio, téléphones portables), dans la distribution d'énergie, dans l'automatisation, etc….

Figure I-2 *:* **Exemples de système embarqué.**

3) Caractéristiques spécifiques [13]

(a) Principales caractéristiques d'un système embarqué

Suite à une comparaison on peut énumérer quelques différences entre le système embarqué qui un est système dédié, et le PC qui est un système à usage général :

(i) Les systèmes embarqués sont des systèmes dédiés

Dans la littérature des systèmes embarqués un microprocesseur embarqué est souvent appelé microprocesseur dédié. Car il est habituellement programmé pour effectuer une seule, ou à la limite un nombre réduit de tâches spécifiques. La modification de la tâche est généralement associée au changement d'ensemble du système et de la réorganisation de ces composants. Par exemple un processeur qui gère un moniteur cardiaque n'est pas prévu pour faire du décodage/encodage vidéo.

(ii) Le Temps Réel (TR) dans les systèmes embarqués

Les opérations de calcul sont alors faites en réponse à un événement extérieur (interruption matérielle). La validité et la pertinence d'un résultat dépendent du moment où il est délivré. Autrement dit les temps de réponses de ces systèmes sont aussi importants que l'exactitude des résultats. Une échéance manquée induit une erreur de fonctionnement qui peut entraîner soit une panne du système (plantage), soit une dégradation non dramatique de ses performances.

(iii) Le coût des systèmes embarqués

Le coût que devrait prendre en général un système embarqué est la plupart du temps lié au coût de l'architecture globale du système (microprocesseur, mémoires et bus). Étant donné que le processeur est en général le composant le plus cher d'un système embarqué, certains concepteurs approuvent que ce dernier soit le facteur principal dans l'évaluation du coût d'un système embarqué. Quoi que cette approbation perde peu à peu sa légitimité en raison de l'apparition des puces SoC (System On Chip). Un SoC typique contient : un processeur, un DSP, de la mémoire (RAM, ROM, flash..), des convertisseurs

Lorsque les systèmes embarqués sont utilisés dans les produits de grande consommation, ils sont fabriqués en grande série. Les exigences de coût se traduisent alors en contraintes sur les différentes composantes du système : utilisation de faibles capacités mémoires et de petits processeurs (4 bits ou 8 bits), mais en grand nombre. Ainsi, les systèmes embarqués sont particulièrement sensibles au coût de production. Il existe des applications dans lesquelles les contraintes de coût de production et de maintenance ont une importance de même niveau que les performances envisagées.

Analogiques/Numériques Numériques/Analogiques et plusieurs unités d'E/S. En d'autres termes le système tout entier sur une même puce, ce qui réduit grandement l'encombrement, l'alimentation et le coût du système (du point de vue matériel, les systèmes embarqués sont détaillés dans le chapitre 4 : SBC pour systèmes embarqués). Le coût d'un système est primordial dans le domaine des systèmes embarqués, car

c'est très fréquent que ces petits appareils électroniques soient vendus sur une grande échelle de production. Ce qui fait qu'une petite variation du prix du produit se répercute sur l'échelle de leurs grand nombre par un grand bénéfice ou une grosse perte. Il est souvent dit que les systèmes embarqués sont sensibles au coût

(iv) Gamme de processeurs pour systèmes embarqués

De nos jours on apprend que plus de 900 différents microprocesseurs/ microcontrôleurs sont disponibles auprès de plus de 60 fournisseurs de semi-conducteurs [4].

Ces fournisseurs sont continuellement confrontés à des contraintes économiques les obligeant à se concurrencer les uns les autres pour obtenir le meilleur processeur visant des marchés divers et bien spécifiques, comme par exemple les processeurs pour les prochaines générations de PDA, ou encore des processeurs pour les puces graphiques destinées aux différentes consoles de jeux.

(v) Consommation réduite de l'énergie des systèmes embarqués

Dans les systèmes embarqués autonomes, la consommation d'énergie est un point critique pour le coût. En effet, une consommation excessive augmente le prix de revient du système embarqué, car il faut alors des batteries de forte capacité.

Pour le PC généraliste. Les CPU x86 comme le Pentium ou l'AMD Athlon ont besoin d'un dissipateur thermique pour protéger le processeur d'une surchauffe.

Les PCs n'ont pas des contraintes particulièrement sur l'alimentation puisqu'ils sont branchés à des sources de tension électrique standard de 110-220 Volt. Par contre les systèmes embarqués sont généralement alimentés par des batteries avec une puissance et une durée limitée. Comme c'est le cas sur les téléphones portables et les PDA, lorsqu'on est obligés de les recharger continuellement. Ce problème se discerne clairement pour les sondes d'explorations autonomes comme par exemple les sondes spatiales. Mise dans un environnement extrême, elles n'ont que les panneaux solaires comme source d'énergie.

(vi) Encombrement des systèmes embarqués

Les systèmes embarqués requièrent souvent un faible encombrement (faible poids) PDA (Personal Digital Assistant), Internet et téléphone mobiles, …). Leur technologie fait alors appel à une électronique et à des applications portables où l'on doit minimiser aussi bien l'encombrement que la consommation électrique. Par conséquent, la réalisation du packaging afin de faire cohabiter sur une faible surface de l'électronique analogique, de l'électronique numérique, des composantes RF (Radiofréquence) sans interférences est une tâche difficile. En effet, les performances des systèmes sur carte deviennent obsolètes dans le contexte des besoins actuels.

Dans les stratégies de conception actuelles, un système embarqué est généralement intégré sur un support silicium unique constituant ainsi un système complet intégré sur une puce SoC (System on Chip). Les systèmes sur puce contiennent généralement une grande variété de dispositifs programmables tels que

des microcontrôleurs, des processeurs de traitement de signaux DSP (Digital-Signal Processor) et des ASIC qui sont développés pour des applications complexes nécessitant une production en grande série.

Les mémoires (ROM et RAM) y sont intégrés pour le stockage des données et des programmes. Ces composants digitaux cohabitent généralement sur le même support de silicium avec des composants analogiques et mixtes divers tels que des composantes radiofréquence (RF) comme moyen de communication, des composantes optiques pour le transfert de données à haut débit, des MEMS (Micro Electro Mechanical System) pour l'interfaçage avec le monde externe, des convertisseurs analogiques/numérique et numérique/analogique requis pour le dialogue interne. L'objectif est d'obtenir une coopération harmonieuse entre composants embarqués afin de garantir des services globaux.

Des contraintes d'implémentation physique sont liées à la consommation de ressources et au contexte de déploiement tel que le poids, la taille physique, la résistance aux vibrations, ou aux irradiations, etc….

(vii) Criticité, fiabilité

Du fait de leur portabilité et de la mobilité des produits dans lesquels ils sont incorporés, les systèmes embarqués évoluent généralement dans des conditions environnementales non déterministes et souvent non maîtrisées. Ils sont exposés à des variations et autres contraintes environnementales susceptibles d'induire des défaillances : vibrations, chocs, variation de température, variations d'alimentation, interférences RF, corrosion, humidité, radiations, … D'où la nécessité de prendre en compte des évolutions des caractéristiques des composants en fonction des conditions environnementales.

(viii) Autonomie des systèmes embarqués

Les systèmes embarqués fonctionnent sans intervention humaines et doivent prendre en considération tous les événements possibles pour que le système global dans lequel puisse accomplir sa tâche dans les meilleures conditions.

(ix) Un système embarqué est un système réactif.

Un système réactif est un système en continuel interaction avec son environnement, il s'exécute avec un rythme déterminé par son environnement.

(b) Décomposition structurelle des systèmes embarqués

Quelle que soit la nature et la complexité du système, on décompose un système embarqués en :

- le système contrôlé
- le système de contrôle

Le système contrôlé = environnement (procédé) équipé d'une instrumentation qui réalise l'interface avec le système de contrôle

Le système de contrôle = éléments matériels (microprocesseurs…) et logiciels dont la mission est d'agir sur le procédé via les actionneurs en fonction de l'état de ce procédé indiqué par les capteurs de manière à maintenir ou conduire le procédé dans un état donné.

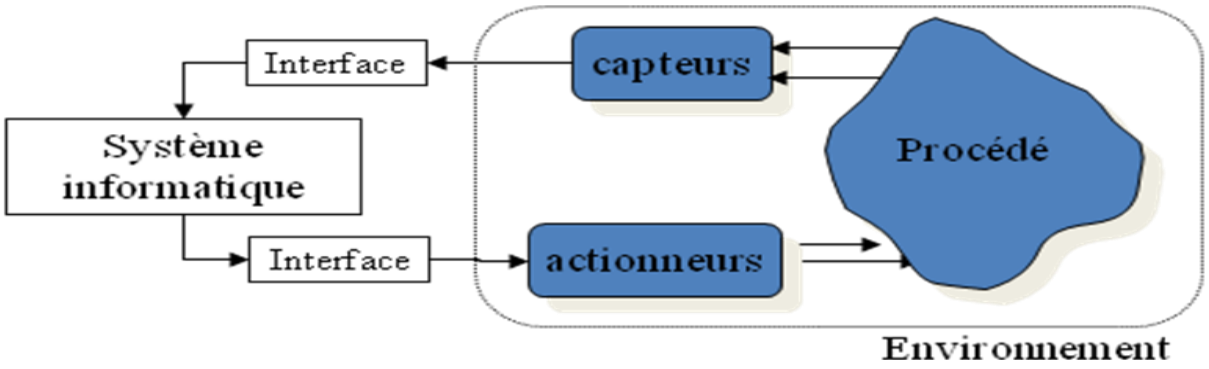


Figure I-3 *:* **Système informatique embarqué dans un procédé.**

Tableau I.1 Comparaison aux systèmes informatiques standards

Informatique :	**Embarqué :**

• Processeur standard
  ○ Multiples unités fonctionnelles (flottant)
  ○ Vitesse élevée (> GHz)
  ○ Consommation électrique élevée
  ○ Chaleur
  ○ Taille
• MMU (mémoire virtuelle)
• OS
• Cache
• Grand nombre de périphériques
	• Processeur dédié (contrôleur)
	  ○ Architecture adaptée
	  ○ Vitesse faible (~200 MHz)
	  ○ 8-32bits : mémoire limitée
	  ○ Basse consommation
	  ○ Petite taille, grand volume => faible coût
	• Processeur DSP (traitements)
	  ○ Très puissants
	• Quelques Mo de mémoire
	• RTOS

(c) Classification des systèmes embarqués :

i. Système Transformationnel

Activité de calcul, qui lit ses données et ses entrées lors de son démarrage, qui fournit ses sorties, puis meurt.

ii. Système Interactif :

Système en interaction quasi permanente avec son environnement, y compris après l'initialisation du système; la réaction du système est déterminée par les événements reçus et par l'état courant (fonction des événements et des réactions passés); le rythme de l'interaction est déterminé par le système et non par l'environnement.

iii. Système Réactif ou Temps Réel :

Système en interaction **permanente** avec son environnement, y compris après l'initialisation du système; la réaction du système est déterminée par les événements reçus et par l'état courant (fonction des événements et des réactions passées); **mais le rythme de l'interaction est déterminé par l'environnement et non par le système.**

4) Systèmes embarqués temps réel

4.1) Définition [13]

Peut être qualifiée de "temps-réel" (ou "temps contraint", ou encore "réactif") toute application mettant en œuvre un système informatique dont le fonctionnement est assujetti à l'évolution dynamique de l'environnement qui lui est connecté et dont il doit contrôler le comportement.

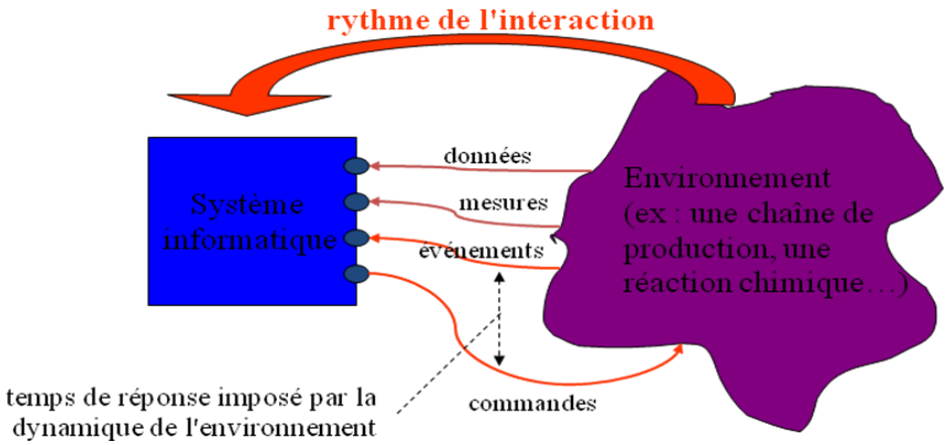


Figure I-4 *:* **Contraintes temps réel dans un système embarqué**

Ce sont des systèmes liés au contrôle de procédés l'exécution de programmes dans ces systèmes doit terminer avant une date appelée échéance au-delà de laquelle les résultats ne sont plus valides.

On distingue deux types de systèmes embarqués temps-réel :

Exemple critique : le contrôleur de frein d'une voiture

• **Système temps-réel dur/stricte** (hard real-time)

Le non-respect des échéances peut avoir des conséquences graves sur le fonctionnement du système ou sur son environnement (autopilotage, freinage, assistance médicalisée…). Les échéances ne doivent jamais être dépassées.

• **Système temps-réel mou** (soft real-time)

Le non-respect des échéances ralentit le système sans conséquences graves (billetterie automatique…). Le système doit répondre au mieux, le plus rapidement possible.

La plupart des systèmes embarqués sont dit mutlirate ou multi-période

- Les données sont capturées à un certain rythme
- Les traitements sur ces données ne sont pas forcément à la même granularité
- Différents traitements peuvent intervenir de manière indépendante
- Les actionneurs fonctionnent à une fréquence différente

4.2) Architecture générale et modes de fonctionnement

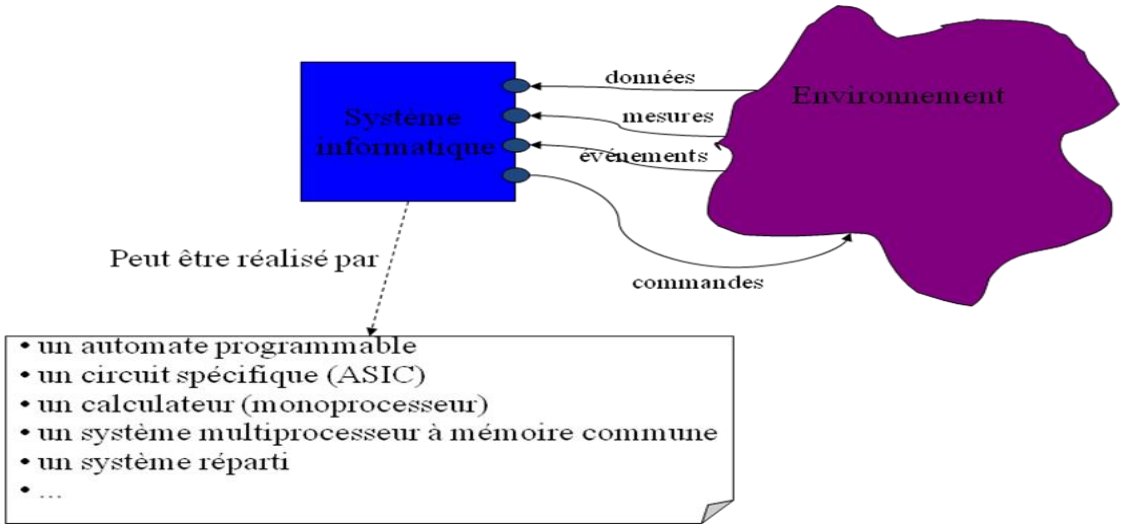


Figure I-5 *:* Diversité d'architecture des systèmes embarqués

4.2.1) Fonctionnement général : boucle infini

Tant que TOUJOURS faire

- Acquisition des entrées (données capteurs, mesures…)
- Calcul des ordres à envoyer au procédé
- Émission des ordres

Fin tant que

Mais, deux modes de fonctionnement :

- fonctionnement **cyclique** (time driven ou système "synchrone")
- fonctionnement **événementiel** (eventdriven)
- fonctionnement mixte : à base de traitements périodiques et et apériodiques (déclenchés sur événements)

4.2.2) Fonctionnement Cyclique

- scrutation d'une mémoire d'entrée périodiquement (polling)

- échantillonnage des entrées sur l'horloge du système
- activation du système à chaque top d'horloge

A chaque top d'horloge faire

- Lecture de la mémoire des entrées
- Calcul des ordres à envoyer au procédé
- Émission des ordres

Fin

Mais ce type de système est peu "réactif" si l'environnement produit des informations à des fréquences différentes ce qui oblige à prévoir toutes les réactions du système dans la même boucle donc il y a un problème de performance et en est obligé à imbriquer des boucles de fréquences multiples ce qui implique des difficultés de réalisation, de lisibilité du code et d'évolution.

4.2.3) Fonctionnement Evénementiel

Le fonctionnement est basé sur le principe d'activation du système à chaque événement (notion d'interruption)

A chaque interruption faire

- Lecture de l'information arrivée
- activation du traitement correspondant
- Émission des ordres issus de ce traitement

Fin

Mais dans ce cas le problème réside dans le cas où une interruption survient alors que le système est en train de traiter une interruption précédente**, ce qui implique des contraintes de programmation :**

- notion de priorité des interruptions
- notion de "tâche" associée à une ou plusieurs interruptions
- mécanisme de préemption et de reprise de tâche
- gestion de l'exécution concurrente des tâches (ordonnancement)

=> Un système temps réel est souvent un système multitâche incluant un gestionnaire de tâches (Ordonnanceur)

4.2.4) Exemple [13]

Le système de contrôle des gouvernes d'un avion est un système embarqué dont les exigences sont :

- le système doit traduire en ordres de déflection des gouvernes les ordres de pilotage venant du pilote

ou de pilote automatique (facteur de charge)

- Le système doit maintenir l'avion dans son domaine de vol quelles que soient les commandes du pilote ou du pilote automatique
- Compte tenu de la dynamique de l'avion, les ailerons et la gouverne de direction doivent être asservis à une période minimale de 10ms
- Compte tenu de la dynamique de l'avion, la gouverne de profondeur doit être asservie à une période minimale de 30ms
- La perte de contrôle à la fois des ailerons et des spoilers est catastrophique (taux de panne de 10^{-9})
- La perte de contrôle de la profondeur est catastrophique (taux de panne de 10^{-9})
- La perte de contrôle des spoilers, des ailerons, de la direction et de la profondeur ne doit pas être causée par une panne unique

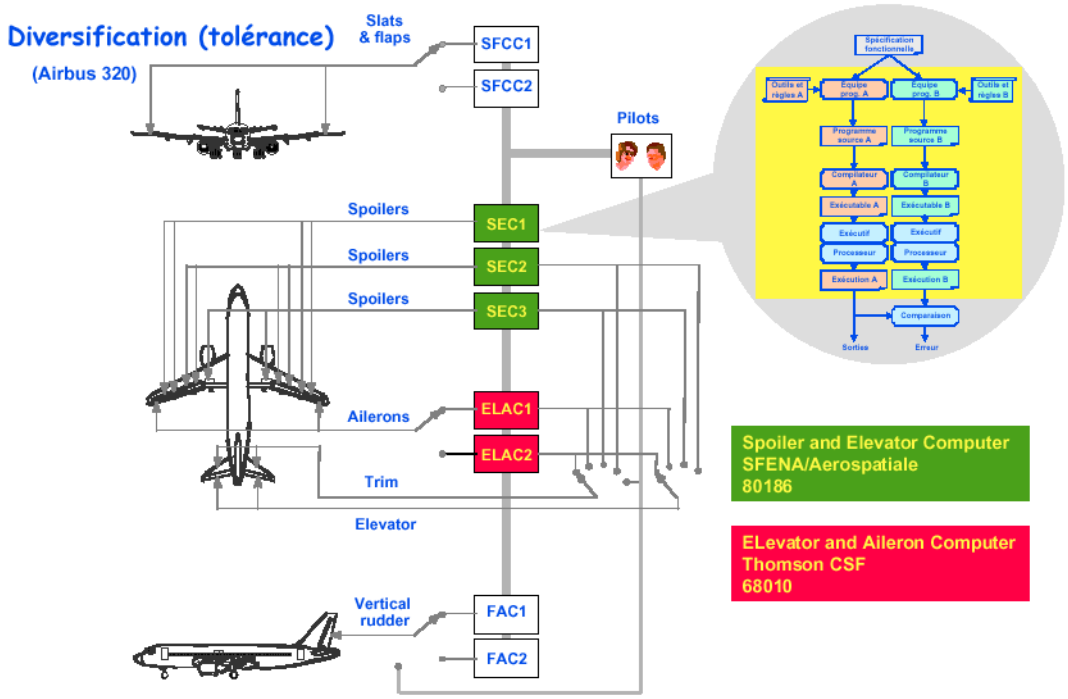


Figure I.6 Le système de contrôle des gouvernes d'un avion

4.3) Les systèmes embarqués et les contraintes temps réel

Généralement, un système embarqué doit respecter des contraintes temporelles, où l'on y trouve enfoui un système d'exploitation ou un noyau Temps Réel (RTOS : Real Time Operating System). Cela veut dire que dans le cas d'une information arrivant de façon aléatoire (généralement sous forme d'une interruption), les temps d'acquisition et de traitement doivent rester inférieurs à la période de rafraîchissement de cette information. Pour cela, il faut que le noyau ou le système Temps Réel soit

déterministe et préemptif pour toujours donner la main durant la prochaine interruption à la tâche de plus forte priorité. (Le temps réel sera plus approfondi dans le chapitre suivant : Les systèmes embarqués et le temps réel). Il ne faut pas confondre Temps Réel et rapidité de calcul. Par exemple le système antiblocage (ABS) du freinage d'une voiture avec un microcontrôleur 16 bits de 20 Mhz [5] de puissance a été conçu pour répondre à des délais bien précis, qu'en aucun cas on ne doit dépasser, lui implanter un processeur plus puissant serait du gaspillage et ne serait d'aucune utilité.

4.4) Système d'exploitation temps réel (RTOS)

Un système d'exploitation temps réel (RTOS) est un système d'exploitation multitâche destiné aux applications temps réel. Ces applications sont généralement des applications pour systèmes embarqués. En réalité, la relation temps réel systèmes embarqués est très forte. Les systèmes embarqués sont souvent soumis à des contraintes temporelles temps rée. Un RTOS n'est pas nécessairement plus rapide qu'un système d'exploitation généraliste, mais un OS temps réel est conçu, pour garantir l'exécution des taches en respectant les délais impartis pour chacune d'elles dans le système. Si ces garanties sont respectées à l'absolu sur tous les délais des tâches, dans ce cas le système est dit temps réel strict (hard real-time) ou si une tolérance vis-à-vis du non-respect strict de tous les délais est permise alors le système est dit temps réel souple (soft real-time).

Un RTOS utilise généralement un algorithme d'ordonnancement (scheduler) spécifique aux systèmes temps réel, dans le but de fournir aux développeurs la capacité de produire des applications avec un comportement déterministe et prévisible dans le système final.

Contrairement à un OS généraliste, un RTOS est évalué beaucoup plus sur sa fiabilité, son déterminisme et sa réactivité, que par sa rapidité ou sa richesse fonctionnelle.

5) Les craches logiciel dans les systèmes embarqués

La question qui se pose souvent aux concepteurs et aux développeurs de systèmes embarqués est de comment savoir si le code est sans bug ? Comment faire pour tester les logiciels complexes qui doivent fonctionner correctement dans toutes les conditions? Le point le plus important à tirer de ces interrogations, c'est que l'échec d'un logiciel est plus grave dans un système embarqué que dans un PC de bureau.

C'est inadmissible pour les concepteurs et les développeurs d'essayer de produire un code avec 0 % de bug. Dans la majorité des cas la phase de test dans le cycle de développement d'un système embarqué est la phase la plus longue et la plus coûteuse, et elle est généralement relative à la nature critique du produit. Il est évident que cela n'est pas suffisant pour prévenir les craches logiciel pour systèmes embarqués, c'est pour cette raison que ces derniers contiennent en général un mécanisme de réinitialisation ou de secours. On peut citer par exemple les chiens de garde (Watch dog timer), qui permettent de réinitialiser le système si le logiciel perd le contrôle.

6) La connectivité des systèmes embarqués

Les systèmes embarqués sont aujourd'hui fortement communicants. Cela est devenu possible grâce à la monté en puissance du calcul offert par les processeurs récents pour systèmes embarqués (32 bits en particulier) et grâce à l'utilisation de la connectivité IP. La connectivité IP permet essentiellement de commander à distance un système embarqué par Internet.

7) La conception d'un système embarqué

Du point de vue technique, la conception d'un système embarqué exige d'être pluridisciplinaire : électronique, informatique, réseaux, sécurité. Mais le concepteur se doit aussi d'être un bon gestionnaire car concevoir un système embarqué revient finalement à un exercice d'optimisation : minimiser les coûts de production pour des fonctionnalités optimales.

8) Conclusion

Ce chapitre a pour but de décrire les différentes caractéristiques des systèmes embarqués. On constate que le domaine des systèmes embarqués est un domaine vaste et pluridisciplinaire, qui a réussi à faire converger plusieurs disciplines a première vue totalement disjointes pour former une technologie des plus utilisée de nos jours. Les principaux domaines qui ont une relation directe ou indirecte avec les systèmes embarqués sont : les systèmes d'exploitation et la programmation système, le génie logiciel, la conception et/ou exploitation d'architecture matériel, l'électronique, les bus de communication, les réseaux de télécommunication, la sécurité informatique et le temps réel. Le chapitre suivant traite la modélisation orienté objet des systèmes embarqués avec le langage de modélisation unifié UML.

CHAPITRE II

Modélisation orienté objet des systèmes embarqués avec UML

Résumé :

Dans ce chapitre on aborde le langage me modélisation UML après avoir introduits les concepts de modélisation fonctionnelle et modélisation orienté objet. Nous avons donné quelques définitions des notions de programmation orientée objet à savoir l'encapsulation des classes et des objets, l'héritage et le polymorphisme des classes. Nous ensuite détaillés les diagrammes à aspect statique et dynamique de modélisation des systèmes embarqués. On termine par une conclusion

Sommaire du Chapitre:

II. 4 Modéliser avec UML

II. 5 Conclusion

1. Introduction à UML

Un modèle est une abstraction de la réalité. Modéliser consiste à identifier les caractéristiques intéressantes ou pertinentes d'un système dans le but de pouvoir l'étudier et prévoir son évolution dans le futur. Par exemple avant de construire une voiture, on conçoit des plans, des tests, des essais de moteurs…c'est la même chose avant de construire une maison, un plan est conçu, les tests par rapport à la résistance des matériaux sont réalisés etc…. Les météorologues utilisent des modèles pour prévoir la météo. On utilise les modèles dans tous les domaines scientifiques et de la réingénierie.

L'approche objet dans la modélisation est apparue en 1967. Simula est premier langage de programmation à implémenter le concept de type abstrait à l'aide de classes [3]. En 1976 déjà, Smalltalk implémente les concepts fondateurs de l'approche objet : encapsulation, agrégation, héritage [3]. Les premiers compilateurs C++ datent du début des années 80 et de nombreux langages orientés objets "académiques" ont étayés les concepts objets (Eiffel, Objective C, Loops...). Il parait donc déjà longtemps que l'approche objet est devenue une réalité. Les concepts de base de l'approche objet sont stables et largement éprouvés. De nos jours, programmer en orienté objet, c'est bénéficier d'une panoplie d'outils et de langages performants. L'approche objet est une solution technologique incontournable. Ce n'est plus une mode, mais un réflexe quasi-automatique dès lors qu'on cherche à concevoir des logiciels complexes qui doivent résister à des changements constants.

UML (UnifiedModelingLanguage) est né de la fusion des trois méthodes qui ont le plus influencé la modélisation objet au milieu des années 90. UML est issu d'un consensus des travaux de trois experts à savoir OMT, Booch et OOSE [3]. De très nombreux acteurs industriels de renom ont adopté UML et participent à son développement.

Comme les systèmes temps-réel embarqués sont devenus plus complexe et le logiciel avancé de programmation structurée s'est avéré limité ce qui a nécessité d'introduire de nouveaux outils de modélisation aux méthodologies orientées objet. UML a été développé en réponse à cette nécessité comme un langage de modélisation orienté objet standardisé. UML peut être adapté à concevoir une variété de systèmes en temps réel, à partir de petits systèmes de microcontrôleurs 8 bits jusqu'à de grands systèmes complexes avec réseau de multiprocesseurs.

Avec UML on peut effectuer les tâches suivantes :

Analyser les besoins du système : Identifier ce que le système embarqué qu'on doit concevoir, doit accomplir et sous quelles contraintes il fonctionnera. Les contraintes dans un système temps réel embarqué comprennent généralement le temps, la capacité mémoire et le débit du processeur. Notre énoncé du problème et les besoins des utilisateurs sont les entrées pour cette étape. Le résultat de cette étape sera un ensemble d'exigences du système qui décrivent ce que le système fait et peut également diviser les exigences en deux parties entre le matériel et le logiciel.

Développer les cas d'utilisation d'UML. Avec les besoins de notre système comme un point de départ, on développe des cas d'utilisation qui couvrent ces besoins. Les cas d'utilisation illustrent les communications entre un système temps réel embarqué et les acteurs externes. Les cas d'utilisation pour les systèmes temps réel également définissent les délais et les exigences de synchronisation.

Définir la structure de l'objet avec le diagramme de classe d'UML. Une fois les cas d'utilisation ont été définis, la structure de l'objet peut être définie. La structure de l'objet d'un système embarqué en temps réel comprend la définition des classes d'objets et des données de chaque classe d'objets qu'il doit contenir. Les diagrammes de classes et des diagrammes d'objets peuvent être utilisés pour modéliser la structure de l'objet. Les diagrammes de classes illustrent les attributs, les opérations et les relations d'une classe à d'autres classes.

Définir le comportement de l'objet. Les objets dans notre conception de la structure de l'objet auront des comportements qui correspondent à la fonctionnalité nécessaire pour les cas d'utilisation. Le comportement de l'objet peut être modélisé par les diagrammes d'états, de séquence et de collaboration d'UML.

Concevoir une architecture. Une architecture pour un système temps réel embarqué comporte des aspects à la fois physiques et logiciels. Il existe des dessins d'architectures de haut niveau qui englobent les systèmes matériels et logiciels dans leur ensemble. L'architecture physique peut être modélisée à l'aide de diagrammes de déploiement d'UML. L'architecture logicielle peut être modélisée à l'aide des modèles de conception d'ULM. La structure de l'objet et le comportement définis dans les étapes précédentes sont utilisés comme des entrées pour la conception de l'architecture logicielle.

Effectuer la conception mécanique. Conception mécanique inclut l'utilisation de classes et d'objets graphiques pour représenter le comportement de collaboration entre de petits groupes de classes ou d'objets.

Effectuer la conception détaillée. Le comportement détaillé et la structure des classes d'objets individuels sont définis lors de la phase de conception détaillée. La conception détaillée est utilisée pour écrire le code pour le logiciel embarqué temps réel.

2. Présentation générale d'UML

Lorsqu'il faut approcher un problème informatique, l'approche intuitive est de le découper en aspects fonctionnels. En effet, cette approche semble être la plus naturelle puisque lorsqu'on programme, il est tout aussi naturel de représenter les fonctions de l'application que l'on veut modéliser.

Prenons l'exemple d'un logiciel de gestion de bibliothèque (figure 1). Ainsi, le logiciel est composé d'une hiérarchie de fonctions, qui ensemble, fournissent les services désirés, ainsi que de données qui représentent les éléments manipulés (livres, etc…).

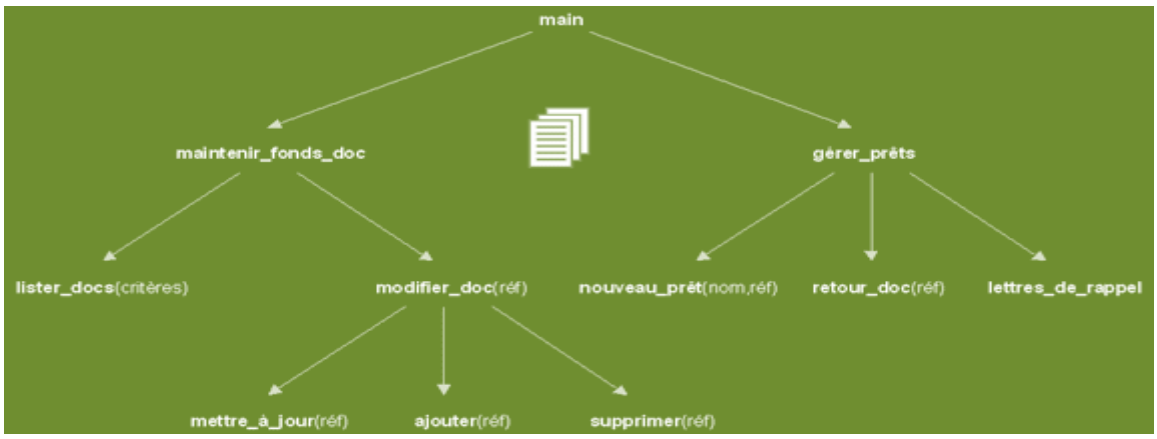


Figure II.1 Gestion bibliothèque : Approche fonctionnelle

➢ Problèmes avec l'approche fonctionnelle

En découpant le logiciel en fonctions, il devient *difficile de faire une simple mise à jour*. En effet, un changement effectué sur une des fonctions peut impliquer des changements majeurs dans une multitude d'autres fonctions. On pourrait contourner ce problème en utilisant des fonctions plus génériques, mais ensuite l'écriture du logiciel devient beaucoup plus difficile.

➢ Vers une approche objet

Pourquoi utiliser une approche orientée objet? Pourquoi ne pas rester avec l'approche fonctionnelle? Un *objet*, en génie logiciel, constitue une entité autonome, qui regroupe un ensemble de propriétés cohérentes et de traitements associés. Nous définissons un objet par les caractéristiques suivantes:

- une entité aux frontières précises qui possède une identité (un nom)
- un ensemble d'attributs caractérise l'état de l'objet

- un ensemble d'opérations (méthodes) en définissent le comportement
- un objet est une instance de classe
- une classe est un type de données abstrait, caractérisé par des propriétés (attributs et méthodes) communes à des objets et permettant de créer des objets possédant ces propriétés

Exemple:

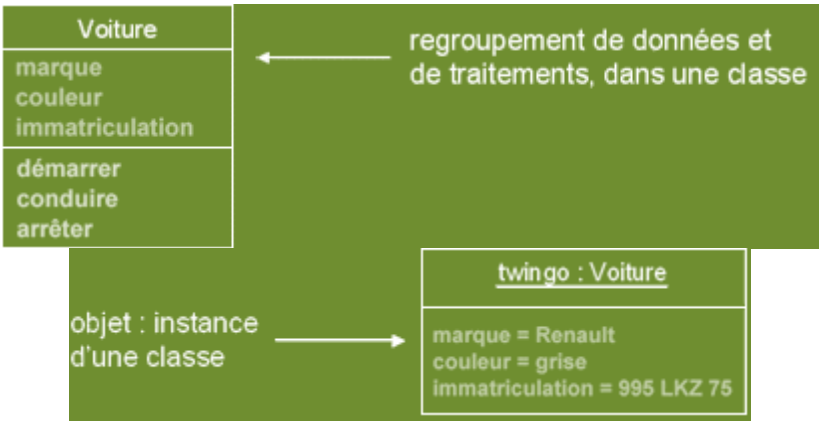


Figure II.2 Approche orientée objet

Une approche objet permet d'obtenir entre autres:

1. *L'encapsulation*
    - consiste à masquer les détails d'implémentation d'un objet, en définissant une interface
    - l'interface est la vue externe d'un objet, elle définit les services accessibles (offerts) aux utilisateurs de l'objet
    - l'encapsulation facilite l'évolution d'une application car elle stabilise l'utilisation des objets : on peut modifier l'implémentation des attributs d'un objet sans modifier son interface
    - l'encapsulation garantit l'intégrité des données, car elle permet d'interdire l'accès direct aux attributs des objets (utilisation d'accesseurs)
2. *Héritage et polymorphisme*
    - l'héritage est un mécanisme de transmission des propriétés d'une classe (ses attributs et méthodes) vers une sous-classe.
    - une classe peut être spécialisée en d'autres classes, afin d'y ajouter des caractéristiques spécifiques ou d'en adapter certaines.
    - plusieurs classes peuvent être généralisées en une classe qui les factorise, afin de regrouper les caractéristiques communes d'un ensemble de classes.
    - la spécialisation et la généralisation permettent de construire des hiérarchies de classes. L'héritage peut être simple ou multiple.
    - l'héritage évite la duplication et encourage la réutilisation.

- le polymorphisme représente la faculté d'une méthode à pouvoir s'appliquer à des objets de classes différentes.
- le polymorphisme augmente la généricité du code.
-

Exemple de l'héritage:

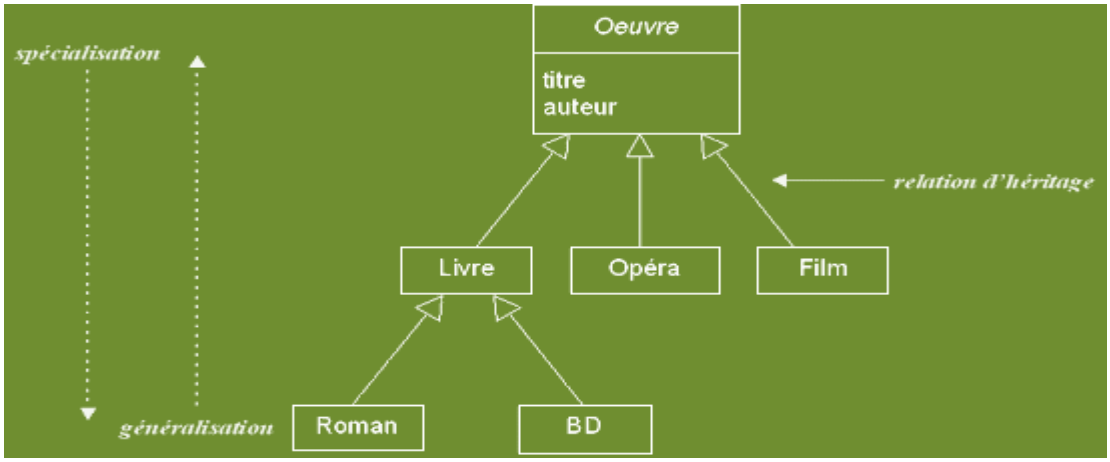


Figure II.3 Héritage

Exemple de polymorphisme:

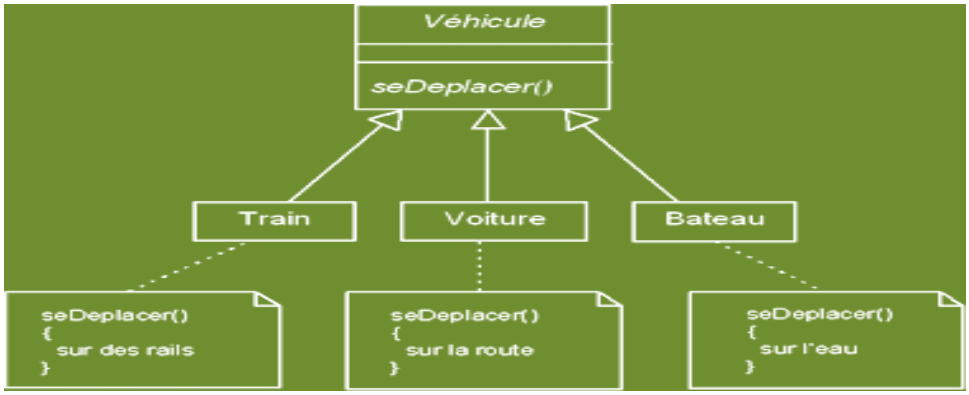


Figure II.4 Polymorphisme

3. *Agrégation*
   - relation entre deux classes, spécifiant que les objets d'une classe sont des composants de l'autre classe. Une relation d'agrégation permet donc de définir des objets composés d'autres objets.
   - l'agrégation permet d'assembler des objets de base, afin de construire des objets plus complexes

Exemple d'agrégation:

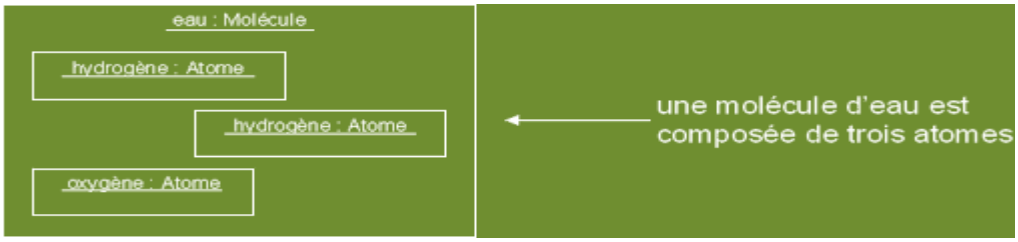


Figure II.5 Agrégation

3. Les objets et l'utilisation d'UML

L'approche objet c'est:

1. un ensemble de concepts stables, éprouvés et normalisés
2. une solution destinée à faciliter l'évolution d'applications complexes

Les obstacles de l'approche objet:

1. l'approche objet est moins intuitive que l'approche fonctionnelle
    - il est plus facile de penser en termes de fonctions que de penser en termes d'objets et d'interactions entre objets
2. l'application des concepts objets nécessite une grande rigueur
    - il est plutôt difficile de décrire la structure objet d'un système de manière pertinente sans risque d'ambiguïtés

Il nous faut donc un "langage" pour exprimer de façon claire et précise les concepts objets du système que l'on veut modéliser. Ce "langage" est UML (UnifiedModelingLanguage).

1. représenter graphiquement des concepts abstraits (diagrammes)
2. limiter les ambiguïtés (parler un langage commun)
3. faciliter l'analyse (simplifier la comparaison et l'évaluation de solutions)
4. définir les vues qui permettent de couvrir tous les aspects d'un système avec des concepts objets
5. penser objet dès le départ

- **Avantages et inconvénients d'UML**

    - ✓ Points forts:

        - UML est un langage formel et normalisé
            - clair et précis
            - encourage l'utilisation d'outils

- UML est un support de communication performant
    - l'analyse devient plus facile
    - facilite la compréhension de représentations abstraites complexes

✓ Point faible:

- La mise en pratique d'UML nécessite un apprentissage et passe par une période d'adaptation

4. Modéliser avec UML

Modéliser consiste à identifier les caractéristiques intéressantes d'une entité, en vue d'une utilisation précise. Une bonne modélisation d'un système devrait permettre de *faciliter la compréhension* du système en question. En effet, un bon modèle devrait réduire la complexité du système et non de le complexifier. Également, un bon modèle devrait permettre de *simuler le système* en question. Un bon modèle devrait reproduire les comportements importants du système étudié.

4.1 Étapes de modélisation avec UML

Il existe plusieurs approches pour modéliser un système avec UML. Nous allons vous proposer une méthode générale pour construire le modèle d'un système. Voici les grandes lignes:

1. Identifier les classes (objets)
    - rechercher les classes candidates à l'aide du diagramme d'objets
    - filtrer les classes redondantes, trop spécifiques ou vagues, qui ne représentent qu'une opération ou un attribut
2. Identifier les associations entre classes / interactions entre objets (instances)
    - rechercher les connexions sémantiques et les relations d'utilisation
    - documenter les relations (nom, cardinalités, contraintes, rôles des classes, etc.)
    - en spécification, filtrer les relations instables ou d'implémentation
    - définisser la dynamique des relations entre objets (les interactions entre instances de classes et les activités associées)
3. Identifier les attributs et les opérations des classes
    - rechercher les attributs dans les modèles dynamiques (rechercher les données qui caractérisent les états des objets)
    - rechercher les opérations parmi les activités et actions des objets
4. Optimiser les modèles
    - utiliser la généralisation et la spécialisation (classification)
    - documenter et détailler vos modèles, encapsulés

4.2 Comment modéliser avec UML?

La partie la plus importante dans l'élaboration d'un modèle avec UML est de définir et visualiser le modèle à l'aide de diagrammes.

- un diagramme UML est une représentation graphique, qui s'intéresse à un aspect précis du modèle ; c'est une perspective du modèle, pas "le modèle"
- chaque type de diagramme UML possède une structure (les types des éléments de modélisation qui le composent sont prédéfinis)
- un type de diagramme UML véhicule une sémantique précise (un type de diagramme offre toujours la même vue d'un système)
- combinés, les différents types de diagrammes UML offrent une vue complète des aspects statiques et dynamiques d'un système.

4.3 Modéliser les aspects statiques

✓ Diagramme de cas d'utilisation (use cases)

- Objectifs:
  - Ce type de diagramme doit représenter le *modèle conceptuel*. En d'autres termes, un coup d'œil à ce diagramme devrait permettre une meilleure compréhension du système.
  - Il sert également à présenter les interfaces entre tous les acteurs du système.
  - Le diagramme de cas d'utilisation vient préciser les utilisateurs (acteurs), les besoins de l'utilisateur et les objectifs du système que l'on modélise. En réponse à l'action d'un acteur, le système fournit un service qui correspond à son besoin.
  - Ce type de diagramme cherche à clarifier, filtrer et organiser les besoins.
- Éléments de base:
  - *Acteurs*: entité externe qui agit sur le système (opérateur, autre système...)
  - *Use case* : ensemble d'actions réalisées par le système, en réponse à une action d'un acteur

Exemple de diagramme d'utilisation pour un système de distributeur d'argent

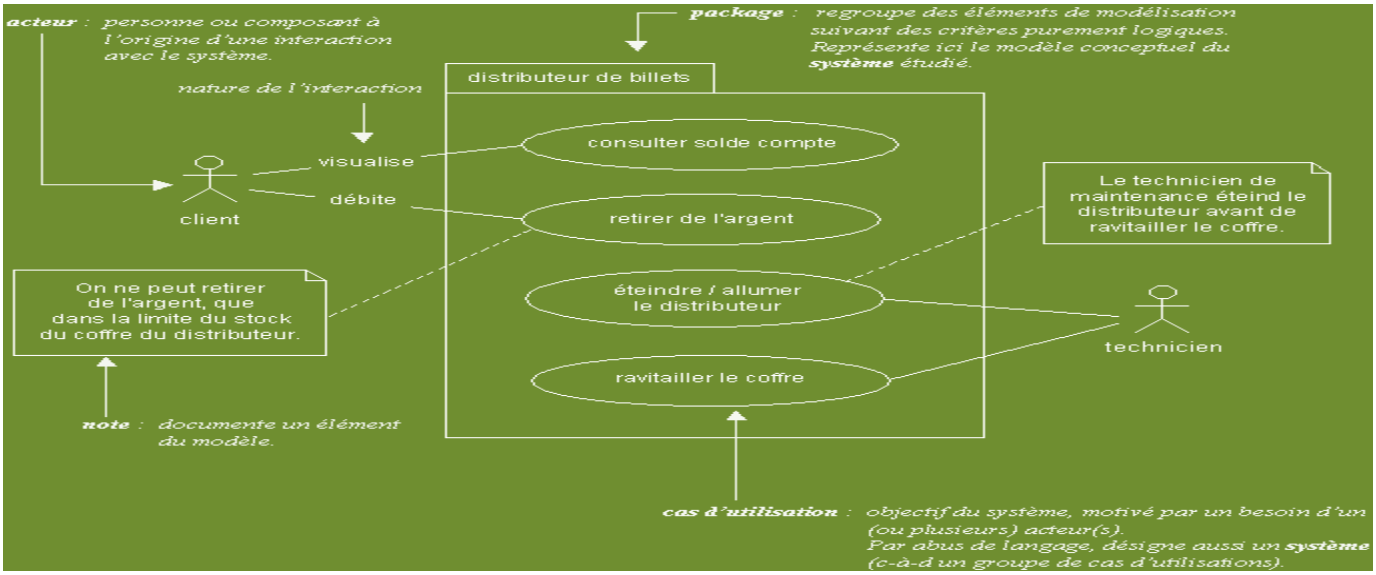


Figure II.6 Diagramme d'utilisation (Use Case) pour un système de distributeur d'argent

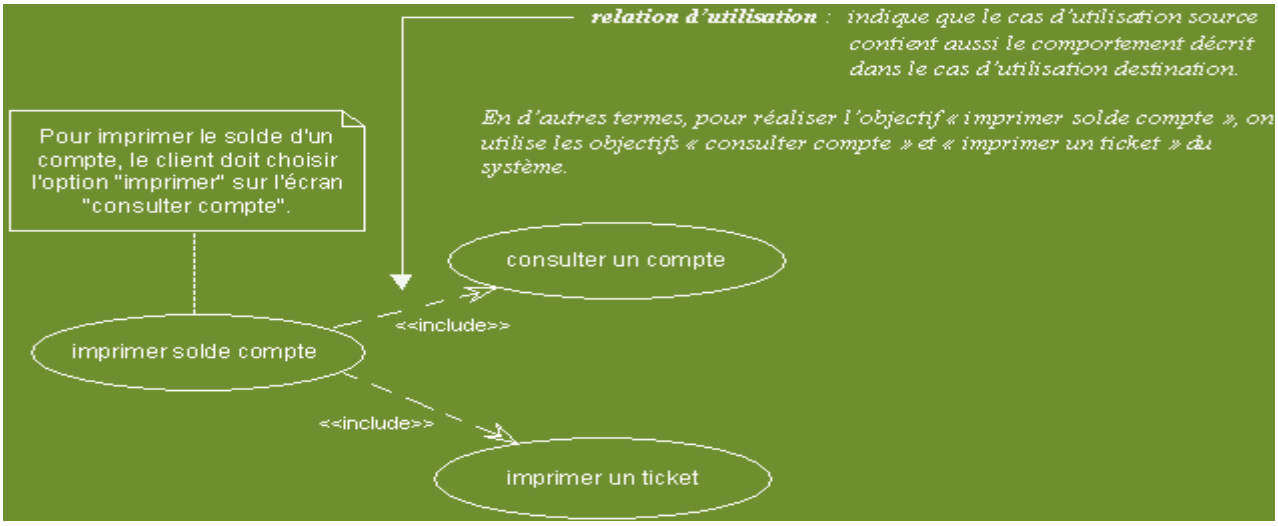


Figure II.7 Relation d'utilisation « include »

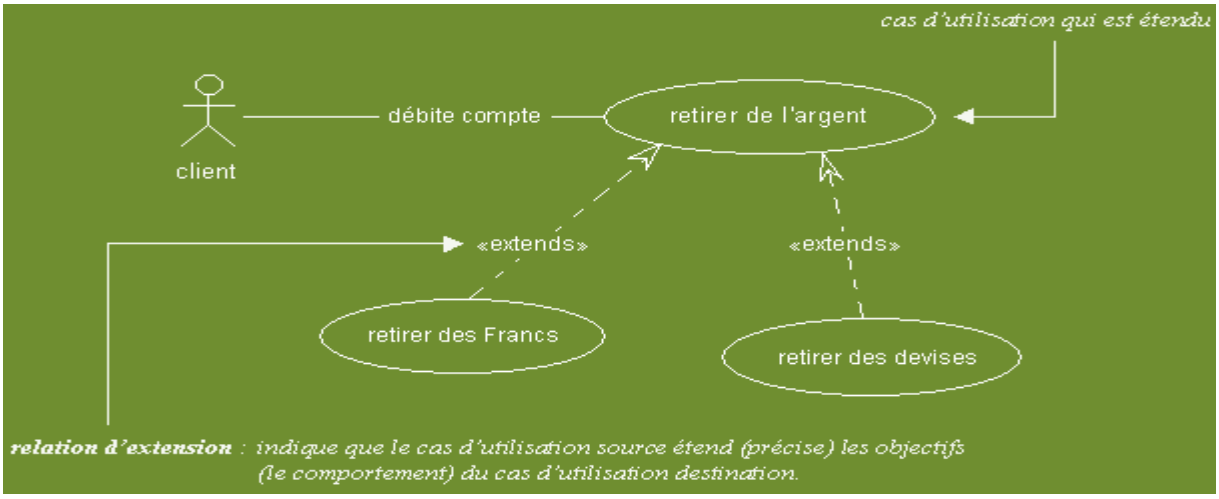


Figure II.8 Relation d'extension « extend »

✓ **Diagramme d'objets**

- Objectif:
  - Ce type de diagramme UML montre des objets (instances de classes dans un état particulier) et des liens (relations sémantiques) entre ces objets à un moment donné.
  - Ce diagramme doit représenter un ensemble d'instances des entités rencontrées dans un diagramme de classe. En d'autres termes, il exprime la partie statique d'une interaction qui comprend les objets qui collaborent mais sans aucun des messages qui circulent entre eux.
- Éléments de base:
  - *Objets*:
  - *Liens*: liaison sémantique entre objet

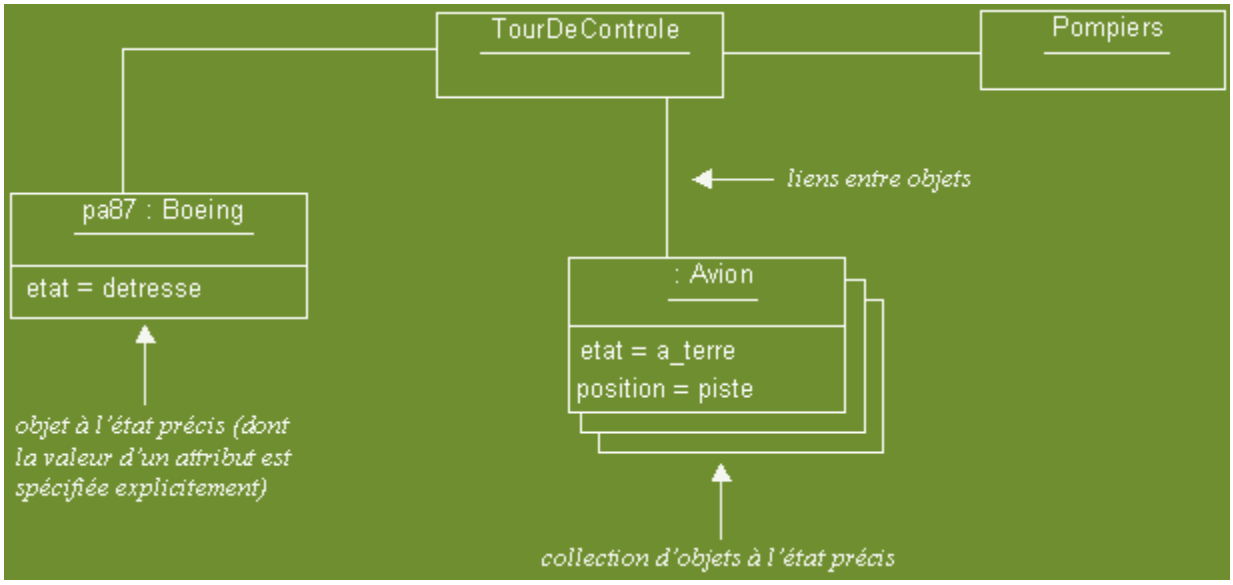


Figure II.9 Exemple de diagramme d'objets pour un système de communication entre la tour de contrôle et les avions

✓ **Diagramme de classes**

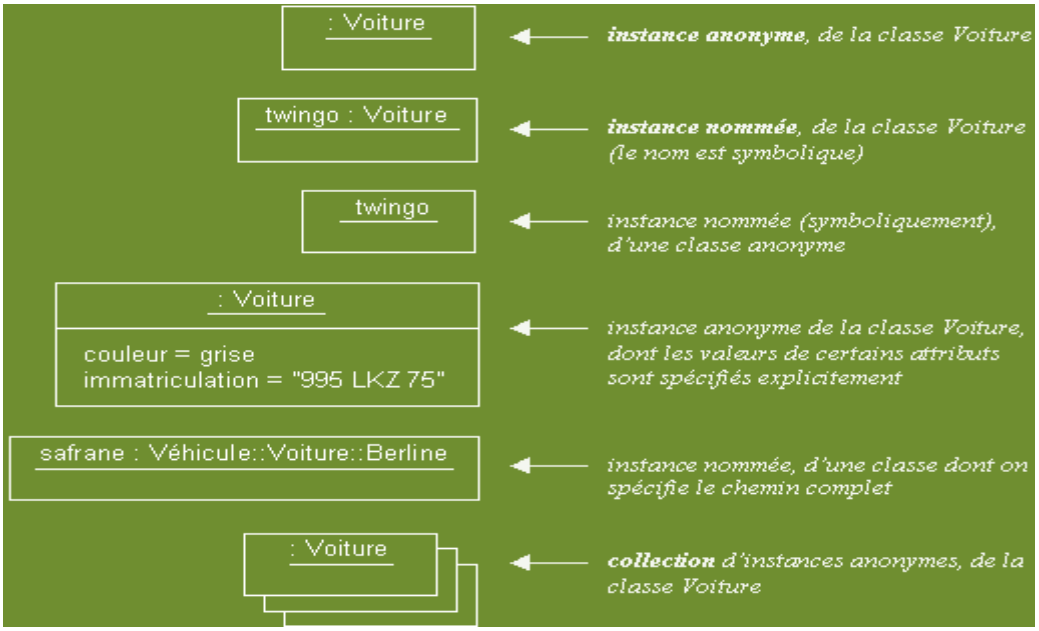


Figure II.10 Diagramme de classe

- Objectifs
  - Un diagramme de classes représente un ensemble de classes, d'interfaces et de collaborations ainsi que leurs relations.
  - Un coup d'œil à ce diagramme devrait indiquer les exigences fonctionnelles d'un système, i.e. les services que le système doit fournir aux utilisateurs finaux.
  - Il doit faire abstraction des aspects dynamiques et temporels.
  - Pour un modèle complexe, plusieurs diagrammes de classes complémentaires doivent être construits.
  - Pour représenter un contexte précis, un diagramme de classes peut être instancié en diagrammes d'objets.
- Éléments de base:
  - *Classes*: type abstrait caractérisé par des propriétés (attributs et méthodes) communes à un ensemble d'objets et permettant de créer des objets ayant ces propriétés.
  - *Relations de dépendance, de généralisation et d'association*: liens entre les classes. Les associations représentent un lien sémantique entre plusieurs classes. Cela veut dire qu'un objet d'une classe a un rapport particulier avec un autre objet d'une autre classe.

Documentation d'une association

- Cardinalité: précise le nombre d'instances qui participent à une relation
  - n : exactement "n" (n, entier naturel > 0) exemples : "1", "7"
  - n..m : de "n" à "m" (entiers naturels ou variables, m > n) exemples : "0..1", "3..n", "1..31"
  - * : plusieurs (équivalent à "0..n" et "0..*")

- n..* : "n" ou plus (n, entier naturel ou variable) exemples : "0..*", "5..*

- Agrégation: représente une relation de type "ensemble / élément"
  - elle peut exprimer qu'une classe (un "élément") fait partie d'une autre ("l'agrégat")
- Composition: la composition est une agrégation forte

On doit se servie de l'agrégation et de la composition pour ajouter de la sémantique à nos modèles lorsque cela est pertinent.

- Contrainte sur une association: expression qui précise le rôle ou la portée d'un élément de modélisation (elles permettent d'étendre ou préciser sa sémantique)
  - graphiquement, il s'agit d'un texte encadré d'accolades

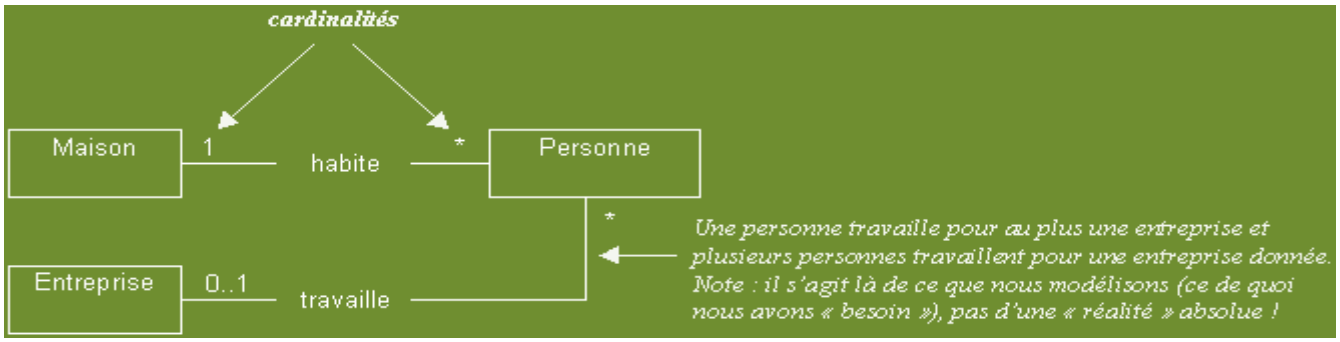


Figure II.11Association entre classes en indiquant les cardinalités

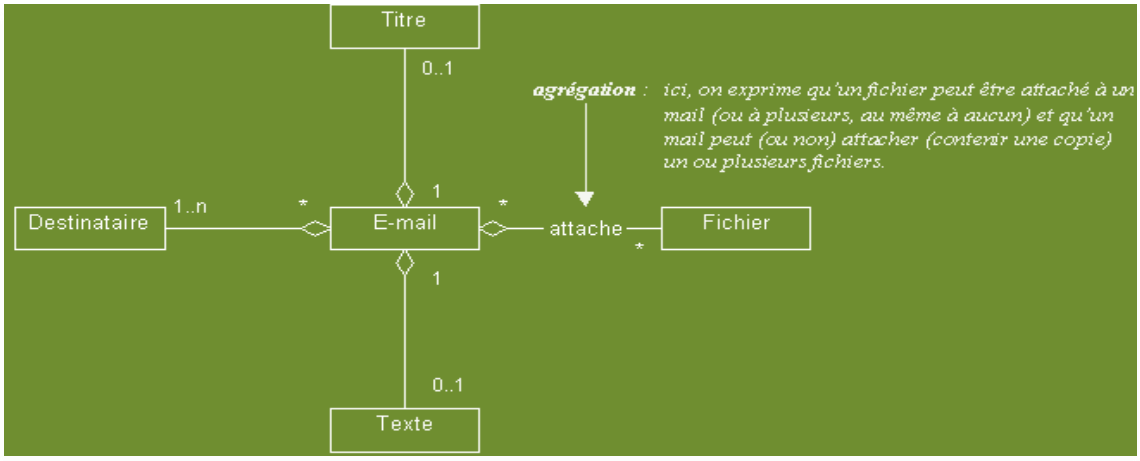


Figure II.12 Association et agrégation de classes

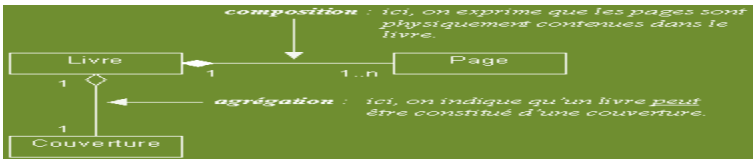


Figure II.13 Association et composition de classe

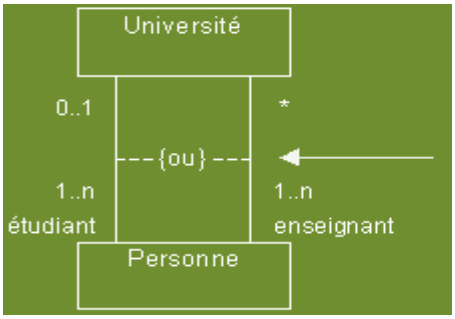


Figure II.14 Contraintes d'association de classes

✓ Directives pour un diagramme de classes bien structuré

- le diagramme doit être centré sur l'aspect statique du système
- disposer ses éléments en évitant que les lignes se croisent
- organiser ses éléments dans l'espace de façon que les éléments proches du point de vue sémantique soient également proches du point de vue physique
- essayer de ne pas montrer trop de types de relations. Fournir les détails qui correspondent à son niveau d'abstraction

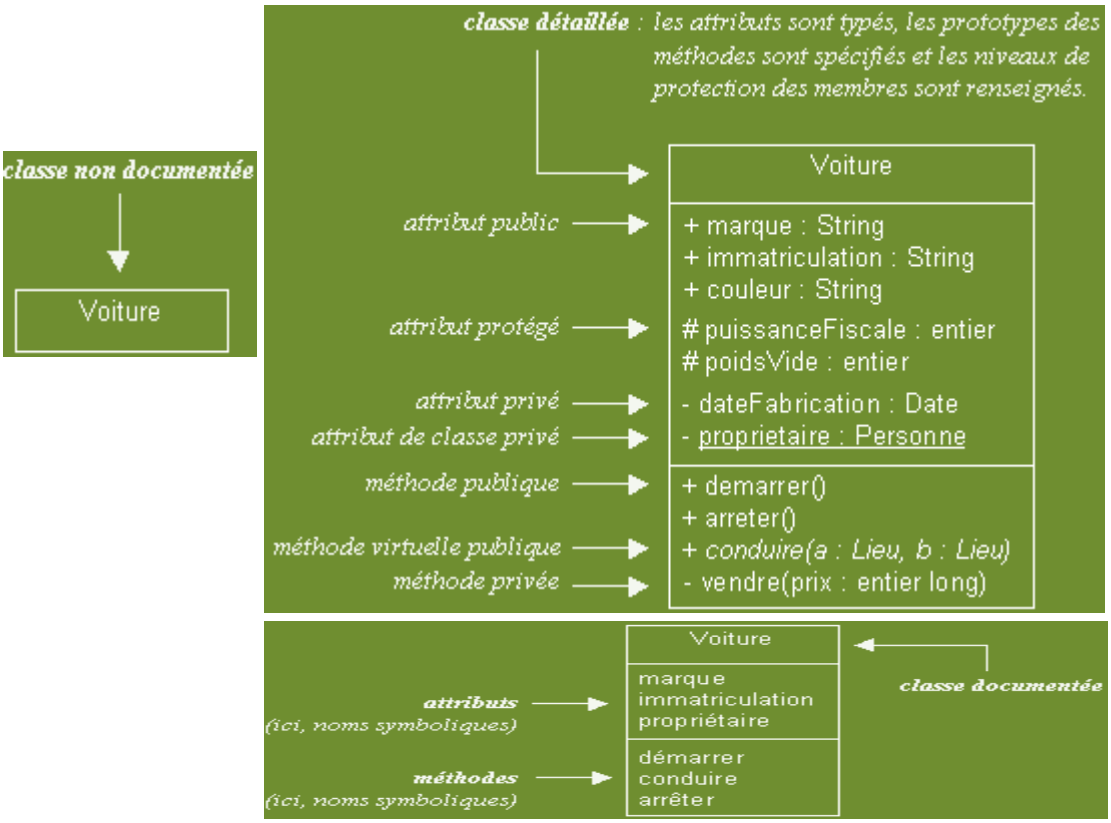


Figure II.15 Diagramme de classes bien structuré

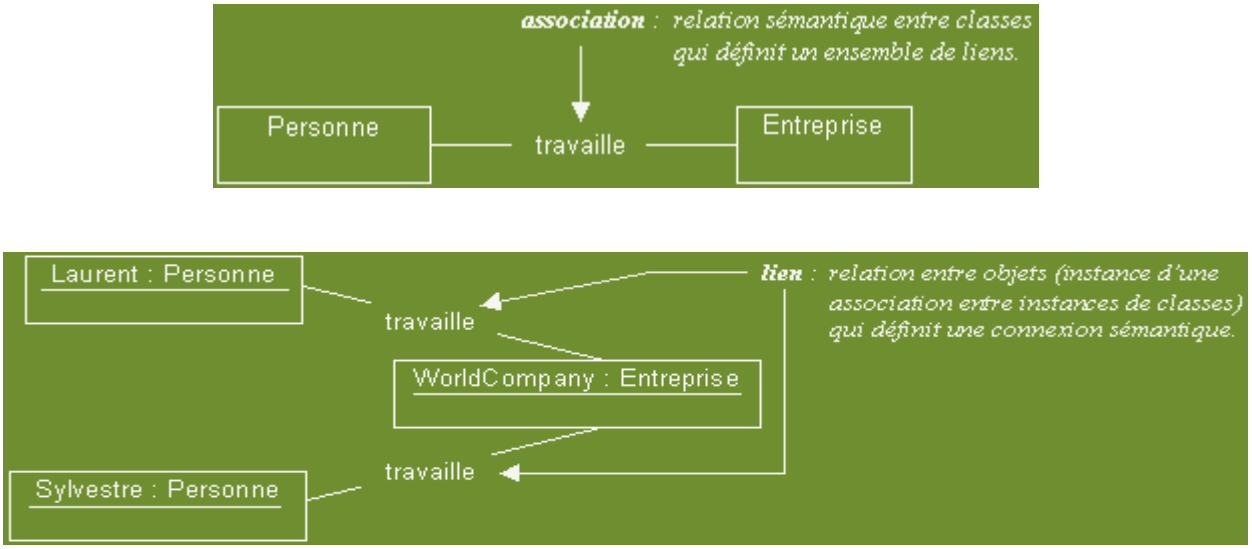


Figure II.16 Relations entre classes et objets instances de classe

4. **Diagramme de composants**

- Objectifs:
  - Il permet de décrire l'architecture physique et statique d'une application en termes de modules : fichiers sources, librairies, exécutables, etc. De plus, il présente l'affectation des objets aux différents composants de cette architecture.
  - Il montre la mise en œuvre physique des modèles de la vue logique avec l'environnement de développement.
  - Ce type de diagramme est important pour visualiser, spécifier et documenter des systèmes basés sur des composants, mais aussi pour construire le système.

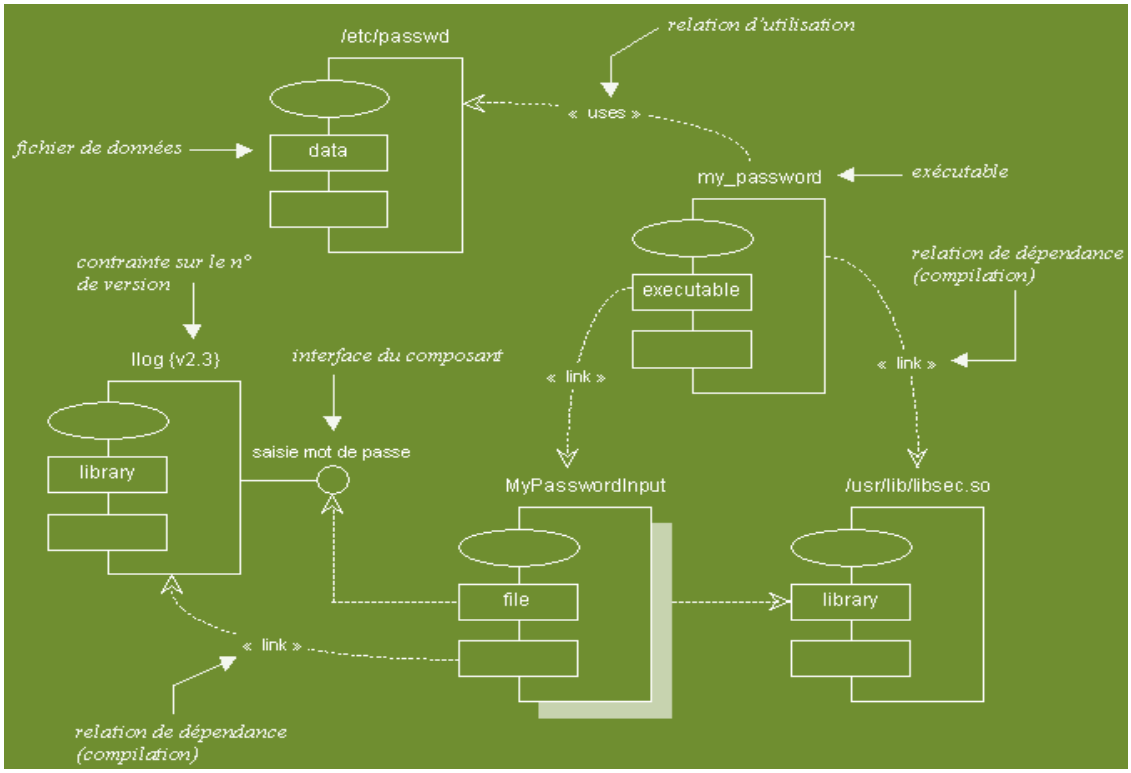


Figure II.17 Diagramme de composant

Diagramme de déploiement

- Objectifs :
  - Ce diagramme montre la disposition physique des matériels qui composent le système et la répartition des composants sur ces matériels à l'exécution.
  - Les ressources matérielles sont représentées sous forme de nœuds.
  - Les nœuds sont connectés entre eux, via un support de communication.
  - Il peut montrer des instances de nœuds (un matériel précis), ou des classes de nœuds.

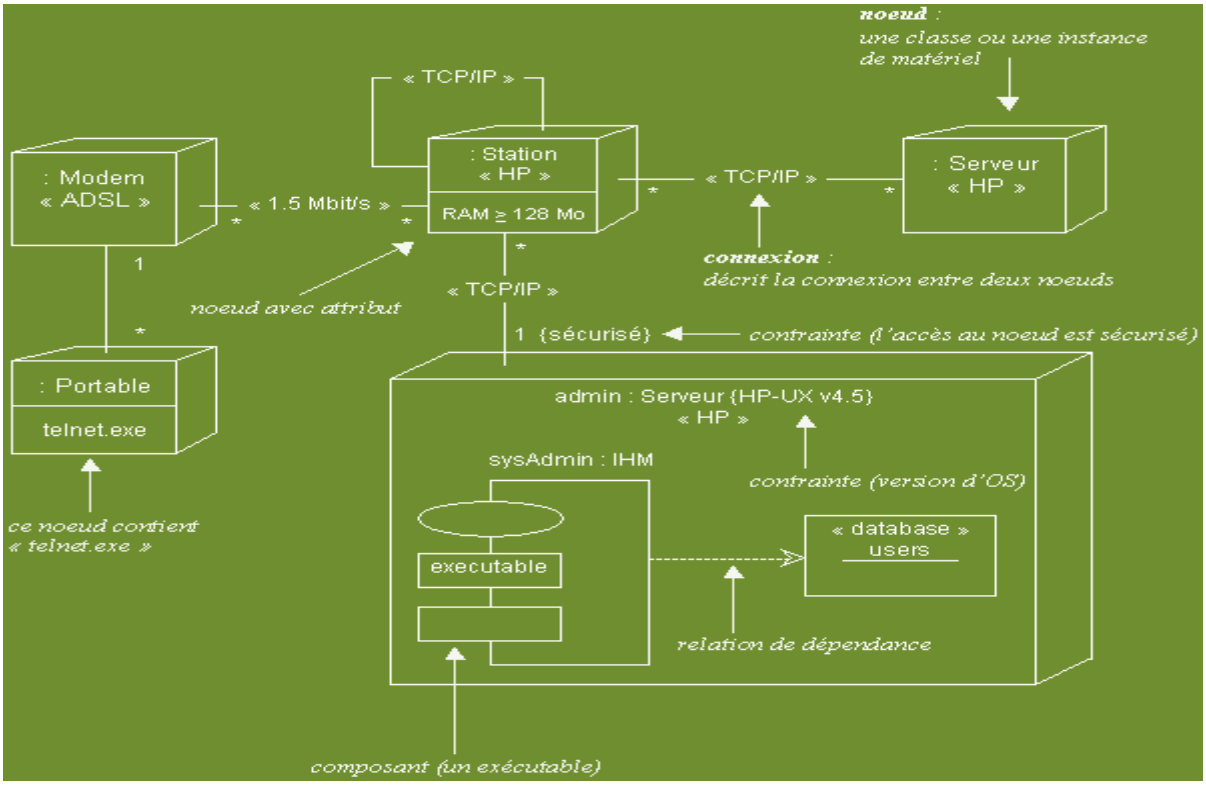


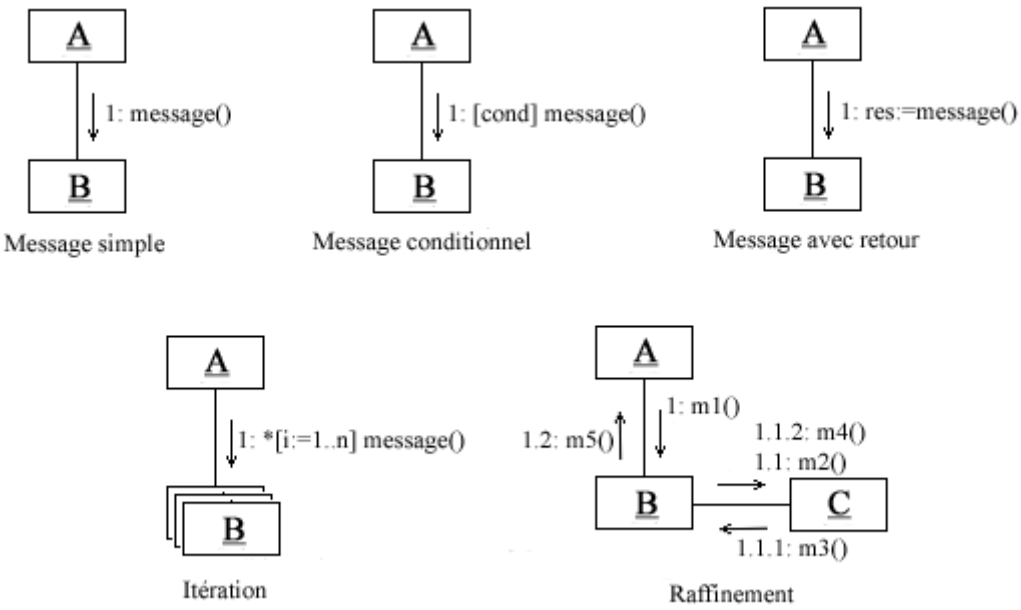
Figure II.18 Diagramme de déploiement

4.4 Modéliser les aspects dynamiques

Diagramme de collaboration

- Objectifs:
  - Ce type de diagramme montre des interactions entre objets (instances de classes et acteurs). Il s'intéresse à la structure de collaboration entre objets (séquencement, itération, concurrence, etc.)
  - Il permet de représenter le contexte d'une interaction, car on peut y préciser les états des objets qui interagissent.
  - Ce diagramme est équivalent au diagramme de séquences. Cependant, l'aspect temporel n'apparaît pas, mais l'aspect chronologique est présent.
- Éléments de base:
  - Un tel diagramme est formé d'un sous-ensemble du diagramme de classes et les messages échangés entre objets à travers les liens.
  - Pour représenter l'aspect chronologique (ordre d'envoi des messages), il est important de numéroter les messages.
  - Le sens des messages nous permet de déterminer l'envoyeur et le receveur.
  -

Syntaxe des messages



Synchronisation des messages

- UML permet de spécifier de manière très précise l'ordre et les conditions d'envoi des messages sur un diagramme dynamique
- Exemples
  - **3 : bonjour()**

    Ce message a pour numéro de séquence "3".

  - **[heure = midi] 1 : manger()**

    Ce message n'est envoyé que s'il est midi.

  - **1.3.6 * : ouvrir ()**

    Ce message est envoyé de manière séquentielle un certain nombre de fois.

  - **1.3, 2.1 / [t < 10s] 2.5 : âge := demander Age (nom, prénom)**

    Ce message (numéro 2.5) ne sera envoyé qu'après les messages 1.3 et 2.1, et que si "t < 10s".

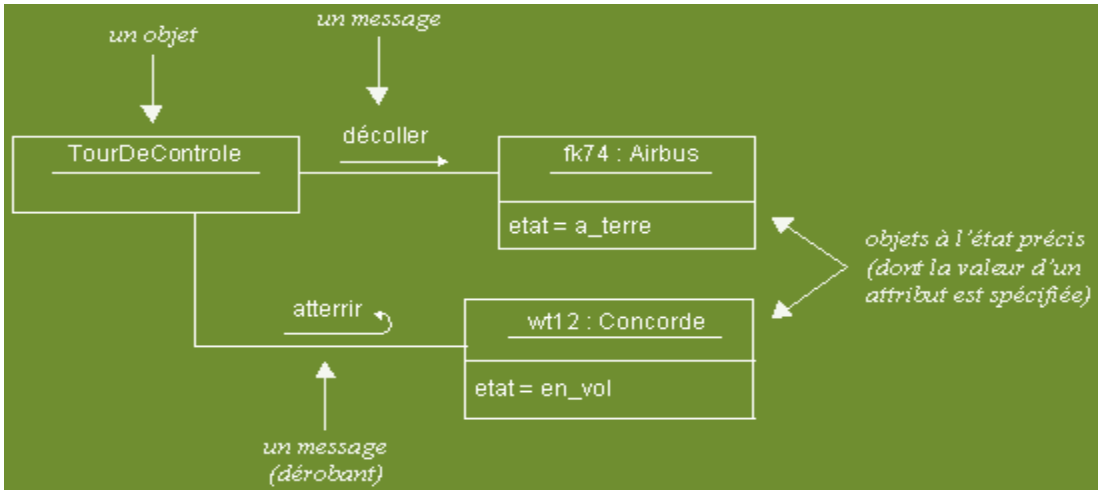


Figure II.19 Exemple de diagramme de collaboration indiquant le sens des messages

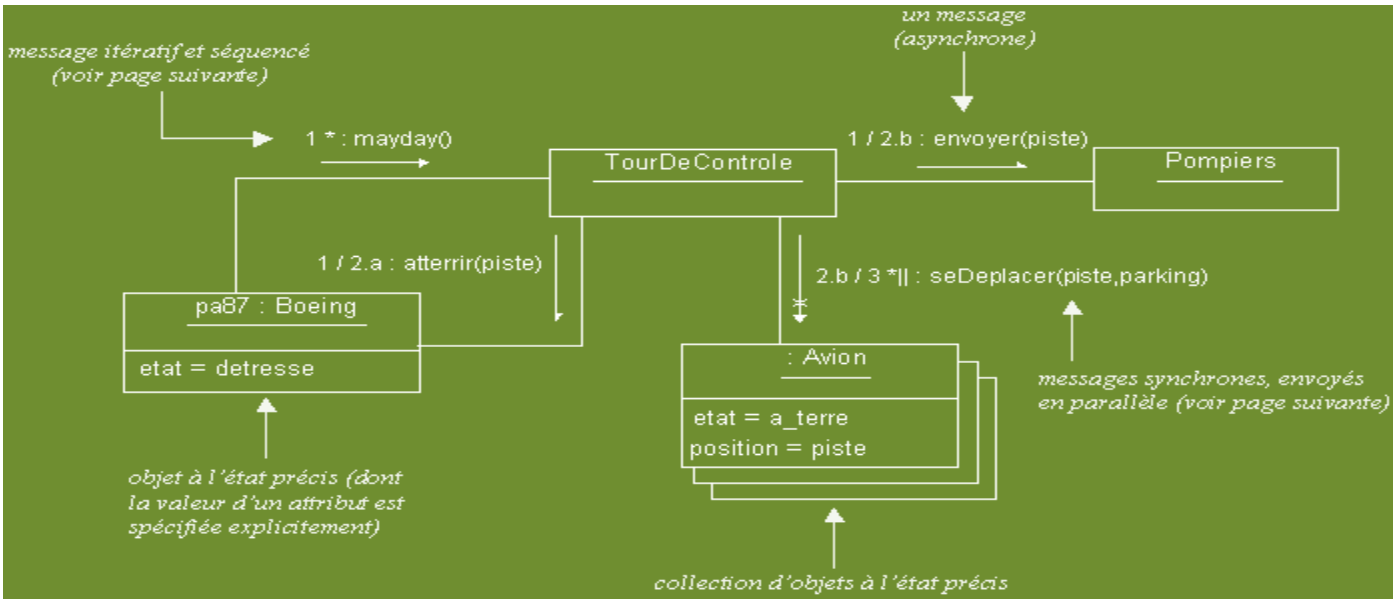


Figure II.20 Exemple de diagramme de collaboration indiquant l'ordre d'envoi des messages

Diagramme de séquences

- Objectifs:
  - Ce type de diagramme permet de représenter des collaborations entre objets selon un point de vue temporel, on y met l'accent sur la chronologie des envois de messages.
  - Il doit montrer l'échange de messages entre objets en fonction du temps.
  - Contrairement au diagramme de collaboration, on n'y décrit pas le contexte ou l'état des objets, la représentation se concentre sur l'expression des interactions.
- Éléments de base:
  - Le diagramme a deux dimensions:
    - dimension verticale: *temps*

- dimension horizontale: *objets*

- L'ordre d'envoi d'un message est déterminé par sa position sur l'axe vertical du diagramme ; le temps s'écoule "de haut en bas" de cet axe.
- La disposition des objets sur l'axe horizontal n'a pas de conséquence pour la sémantique du diagramme.

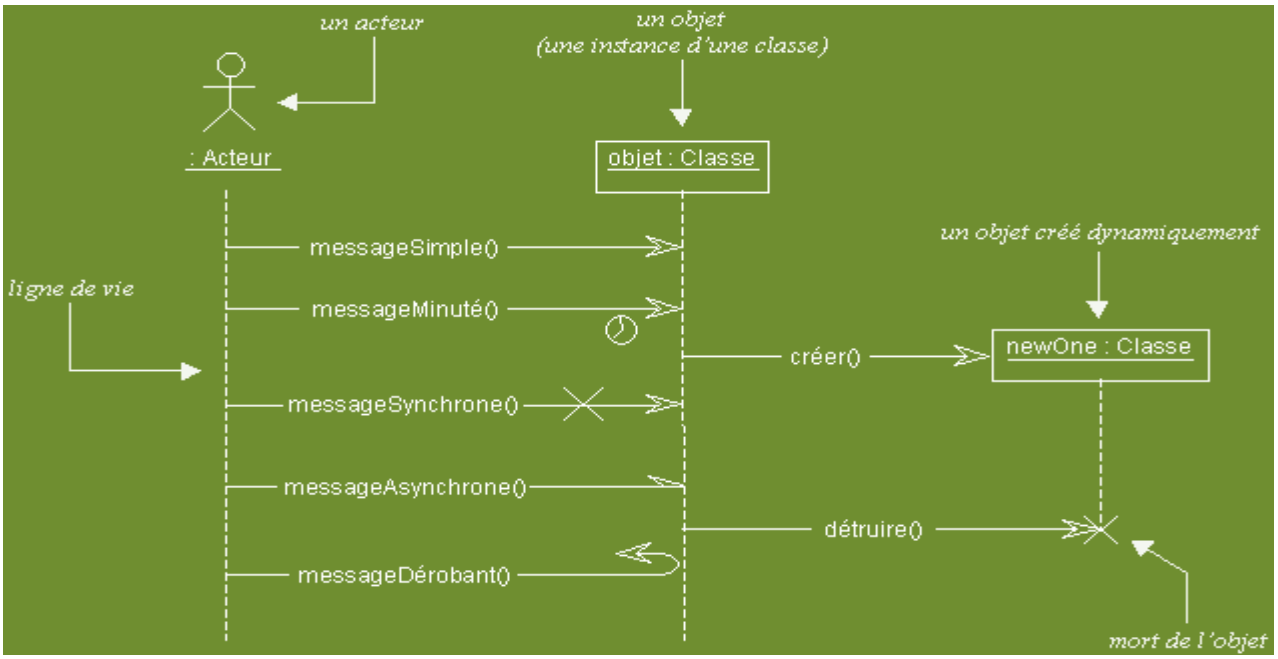


Figure II.21 Exemple des types de messages dans un diagramme de séquences

Types de messages

- **message simple**

  Message dont on ne spécifie aucune caractéristique d'envoi ou de réception particulière.

- **message minuté (timeout)**

  Bloque l'expéditeur pendant un temps donné (qui peut être spécifié dans une contrainte), en attendant la prise en compte du message par le récepteur.L'expéditeur est libéré si la prise en compte n'a pas eu lieu pendant le délai spécifié.

- **message synchrone**

  Bloque l'expéditeur jusqu'à prise en compte du message par le destinataire. Le flot de contrôle passe de l'émetteur au récepteur (l'émetteur devient passif et le récepteur actif) à la prise en compte du message.

- **message asynchrone**

  N'interrompt pas l'exécution de l'expéditeur. Le message envoyé peut être pris en compte par le récepteur à tout moment ou ignoré (jamais traité).

- **message dérobant**

  N'interrompt pas l'exécution de l'expéditeur et ne déclenche une opération chez le récepteur que s'il s'est préalablement mis en attente de ce message.

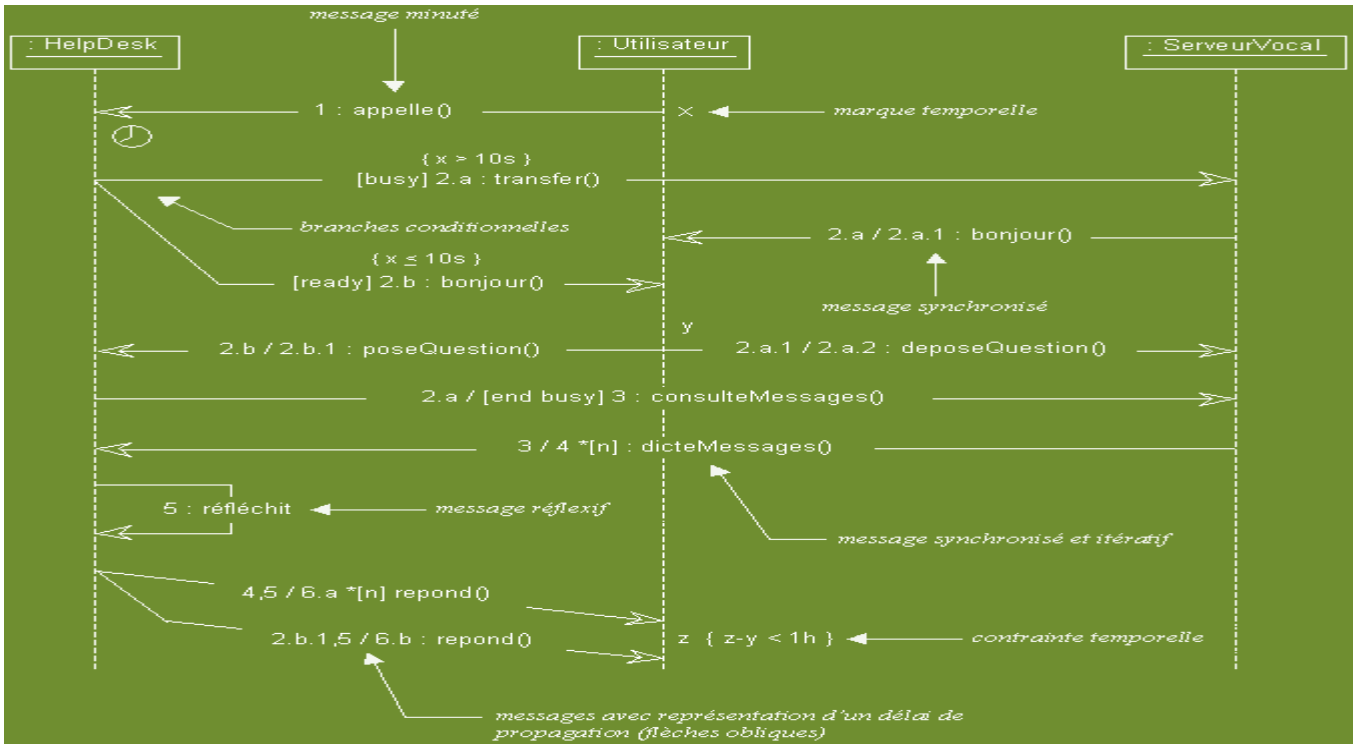


Figure II.22 Exemple complet de diagramme de séquences

Diagramme d'états-transitions

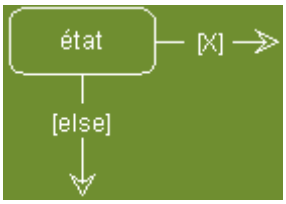
- Objectifs:
  - Ce diagramme sert à représenter des automates d'états finis, sous forme de graphes d'états, reliés par des arcs orientés qui décrivent les transitions.
  - Les diagrammes d'états-transitions permettent de décrire les changements d'états d'un objet ou d'un composant, en réponse aux interactions avec d'autres objets/composants ou avec des acteurs.
- Éléments de base:
  - *États*: valeurs des attributs représentées par des nœuds

- Un diagramme d'état peut contenir plusieurs états initiaux, plusieurs états finaux
- Il peut contenir un état historique (sauvegarde de l'état d'exécution)

- *Transitions*: événements représentés par des arcs orientés
  - Une transition représente le passage instantané d'un état vers un autre.
  - Une transition est déclenchée par un événement. En d'autres termes : c'est l'arrivée d'un événement qui conditionne la transition.
  - Les transitions peuvent aussi être automatiques, lorsqu'on ne spécifie pas l'événement qui la déclenche.
  - En plus de spécifier un événement précis, il est aussi possible de conditionner une transition, à l'aide de "gardes" : il s'agit d'expressions booléennes, exprimées en langage naturel (et encadrées de crochets).

Notation graphique d'un état, transition et événement

Exemple de transition conditionnelle



Exemple de diagramme d'états-transitions d'une machine à laver les voitures

En phase de lustrage ou de lavage, le client peut appuyer sur le bouton d'arrêt d'urgence. S'il appuie sur ce bouton, la machine se met en attente. Il a alors deux minutes pour reprendre le lavage ou le lustrage (la machine continue en phase de lavage ou de lustrage, suivant l'état dans lequel elle a été interrompue), sans quoi la machine s'arrête. En phase de séchage, le client peut aussi interrompre la machine. Mais dans ce cas, la machine s'arrêtera définitivement (avant de reprendre un autre cycle entier).

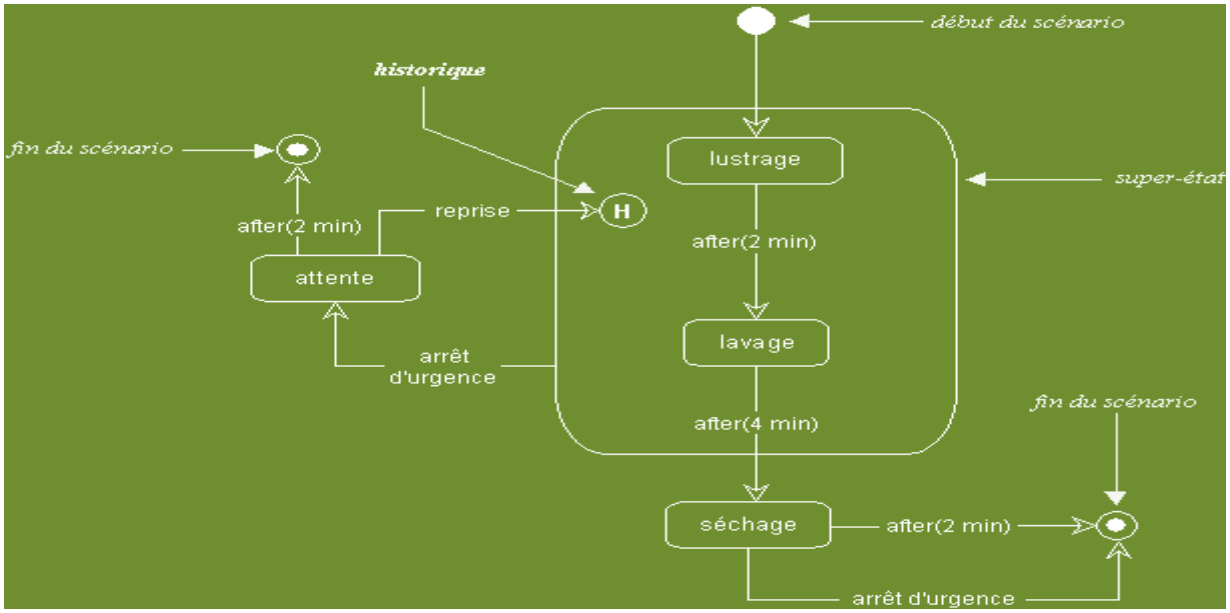
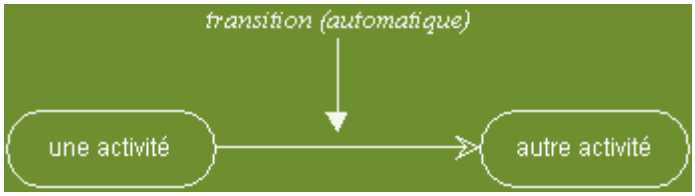


Figure II.23 Diagramme d'états-transitions d'une machine à laver les voitures

Diagramme d'activités

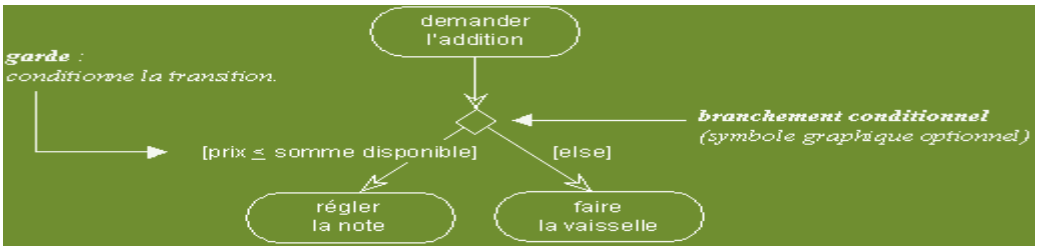
- Objectifs:
  - UML permet de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation, à l'aide de diagrammes d'activités (une variante des diagrammes d'états-transitions).
  - Ce type de diagramme doit décrire les actions et leurs résultats (implantation d'une opération).
  - Il s'agit d'une variante aux diagrammes d'états-transitions où dans ce cas les états sont des actions.
- Éléments de base:
  - *Activité*: exécution d'un mécanisme, un déroulement d'étapes séquentielles.
    - Le passage d'une activité vers une autre est matérialisé par une transition.
    - Les transitions sont déclenchées par la fin d'une activité et provoquent le début immédiat d'une autre (elles sont automatiques).
    - En théorie, tous les mécanismes dynamiques pourraient être décrits par un diagramme d'activités, mais seuls les mécanismes complexes ou intéressants méritent d'être représentés.
  - *Transitions*: représentées par des arcs
    - Elles sont automatiques. Il n'y a donc pas d'événement déclencheur.

Notation graphique d'activités et de transition



Exemple de diagramme d'activités conditionnelles

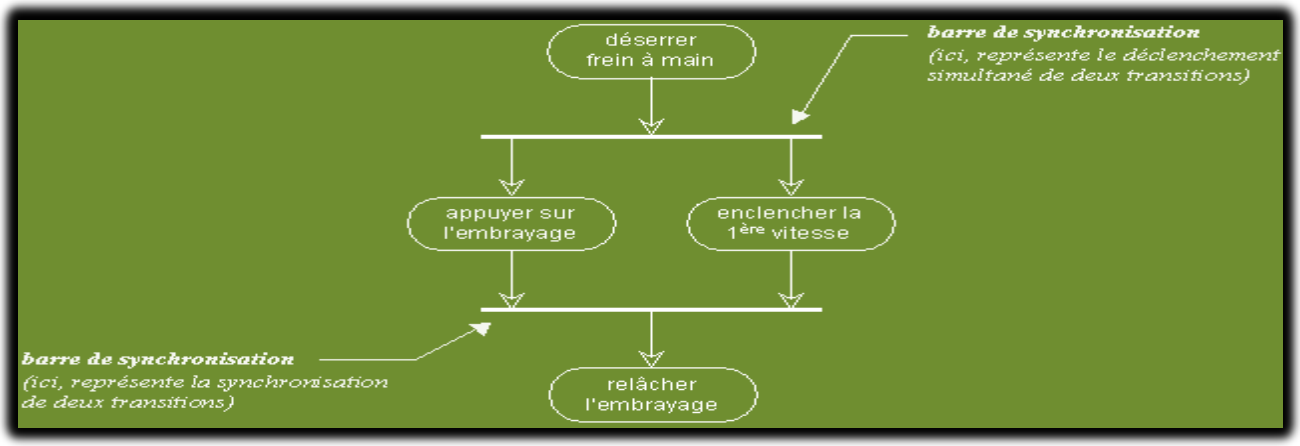
Pour représenter des transitions conditionnelles, utilisez des gardes (expressions booléennes exprimées en langage naturel).



Synchronisation des transitions

- Il est possible de synchroniser les transitions à l'aide des "**barres de synchronisation**"
- Une barre de synchronisation permet d'ouvrir et de fermer des branches parallèles au sein d'un flot d'exécution.
    - Les transitions qui partent d'une barre de synchronisation ont lieu en même temps
    - On ne franchit une barre de synchronisation qu'après réalisation de toutes les transitions qui s'y rattachent

Exemple d'utilisation de barre de synchronisation



Couloir d'activités

- Afin d'organiser un diagramme d'activités selon les différents responsables des actions représentées, il est possible de définir des "couloirs d'activités"
- Il est même possible d'identifier les objets principaux, qui sont manipulés d'activités en activités et de visualiser leur changement d'état

Exemple de diagramme d'activités en utilisant les couloirs d'activités

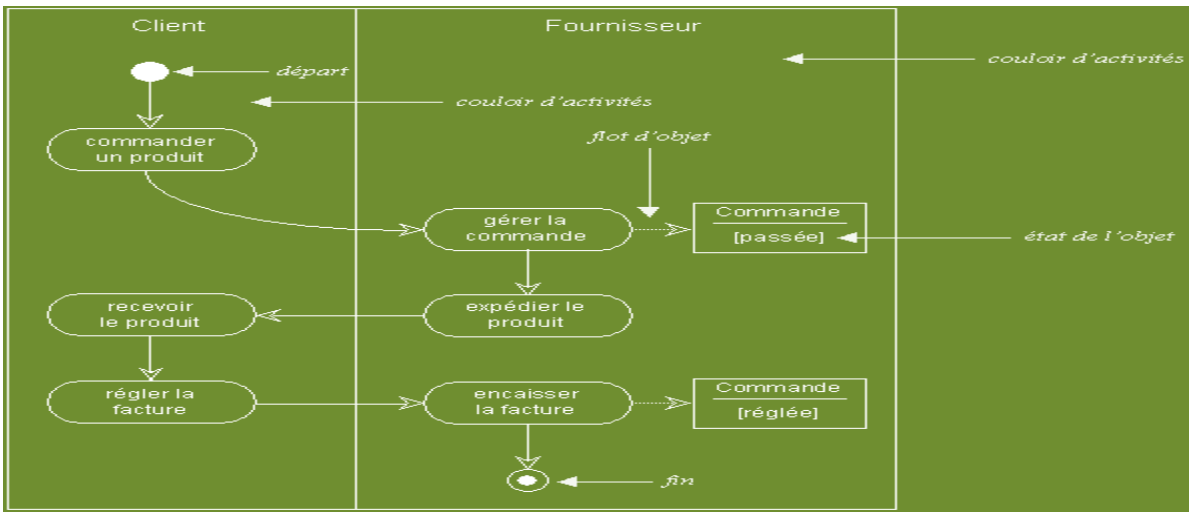


Figure II.24 Diagramme d'activités en utilisant les couloirs d'activités

5. **Conclusion** : Dans ce chapitre nous expliqué en détails la structures des diagrammes du langage UML (UnifiedModelingLanguage) sur différents exemples de cas hétérogènes et ce après avoir défini les notions relatives à la programmation orientée-objet. Cela montre que le langage UML est utilisé dans tous les domaines de l'activité humaine, soit économique, industriel, sociologique, médical etc… Nous avons traité les diagrammes qui modélisent les aspects statiques des systèmes à savoir les diagrammes use case (cas d'utilisation), d'objets, de classes, de composants et de déploiement. Ensuite nous avons traité les diagrammes qui modélisent les aspects dynamiques des systèmes à savoir les diagrammes de collaboration, de séquences, d'états-transitions et enfin d'activité. Ces diagrammes modélisent sous des angles différents n'importe qu'elle système et en particulier les systèmes embarqués. Dans le chapitre suivant nous allons traiter un cas d'étude pratique d'un lave-linge ménager en utilisant un langage dérivé de l'UML qui est le Sysml (System ModelingLanguage) qui comporte des diagrammes spécifiques aux systèmes embarqués.

CHAPITRE III

Application de la modélisation UML/Sysml à un cas d'étude de système embarqué : Le Lave-linge

Résumé :

Dans ce Chapitre, nous allons utiliser SysML un profil d'UML pour une application sur un cas d'étude choisi : un lave-linge ménager. Pour cela nous allons utiliser la plate-forme logicielle MagicDraw pour modéliser le système embarqué du lave-linge en créant les différant diagrammes modélisant ses aspects statiques et dynamiques en respectant les exigences issus d'un cahier des charges et en respectant les contraintes temps réel de son fonctionnement. On termine par une conclusion.

Sommaire du Chapitre:

1. L’origine de SysML et la notion de profil

SysML est issu du langage UML dont il partage un certain nombre d’éléments tout en apportant des compléments on le qualifie alors de profil d’UML. Devenu une référence pour la modélisation systèmes, il permet d’unifier à la fois les notations et les concepts orientés objets. Un profilsert à adapter un langage de base à un domaine particulier ou à une problématique spécifique. SysML et MARTE sont deux profils du langage UML, le premier dédié à l’ingénierie système et le second aux systèmes embarqués.

Le langage SysML permet l’analyse (Bruel, 2013), la spécification, la conception, la vérification et la validation des systèmes complexes. Ces systèmes se composent de composants mécaniques, électroniques (HW), logiciels (SW), d’utilisateurs et de procédures. SysML peut également être utilisé conjointement avec d’autres langages plus spécifiques à un domaine, comme UML pour les logiciels et VHDL pour les composants électroniques. SysML permet de représenter les éléments suivants d’un système :

- La structure du système, incluant la hiérarchie et la classification des composants.
- Le comportement du système, qui peut être fonctionnel, ou bien par échange de messages ou par des machines à états.
- Des contraintes applicables aux propriétés du modèle.
- Des relations d’allocations entre les comportements, les exigences et la structure du système.
- Des exigences et leurs relations avec les autres éléments du modèle.

Les concepts du langage de base UML sont décrits par des classes, étendu par de nouvelles classes appelées stéréotypes dans le profil Sysml. On peut voir un **stéréotype** comme une étiquette que l’on rajoute à certains éléments du langage de base. On distingue deux types de profils (Sélic, 2014), les profils destinés à étendre les concepts existants dans UML (profil de langage) et les profils apportant des informations supplémentaires permettant d’analyser un modèle ou de générer du code (profil d’annotation). Dans la Figure III.1 « exemples de stéréotype SysML» (SysML, 2015) une exigence fonctionnelle «functionalrequirement» est définie par la spécialisation d’une exigence SysML «requirement». Ce nouveau stéréotype dispose également d’une relation avec une fonction de type « Behavior » correspondant à une classe de base (une « metaclass») du langage UML. Le stéréotype « Configuration Item » est défini par extension des méta-classe UML «NamedElement» et «Directed Relationship». La figure 5 représente un modèle utilisant ces stéréotypes pour représenter une **exigence** (Requirement) liée à une distance de freinage d’un véhicule.

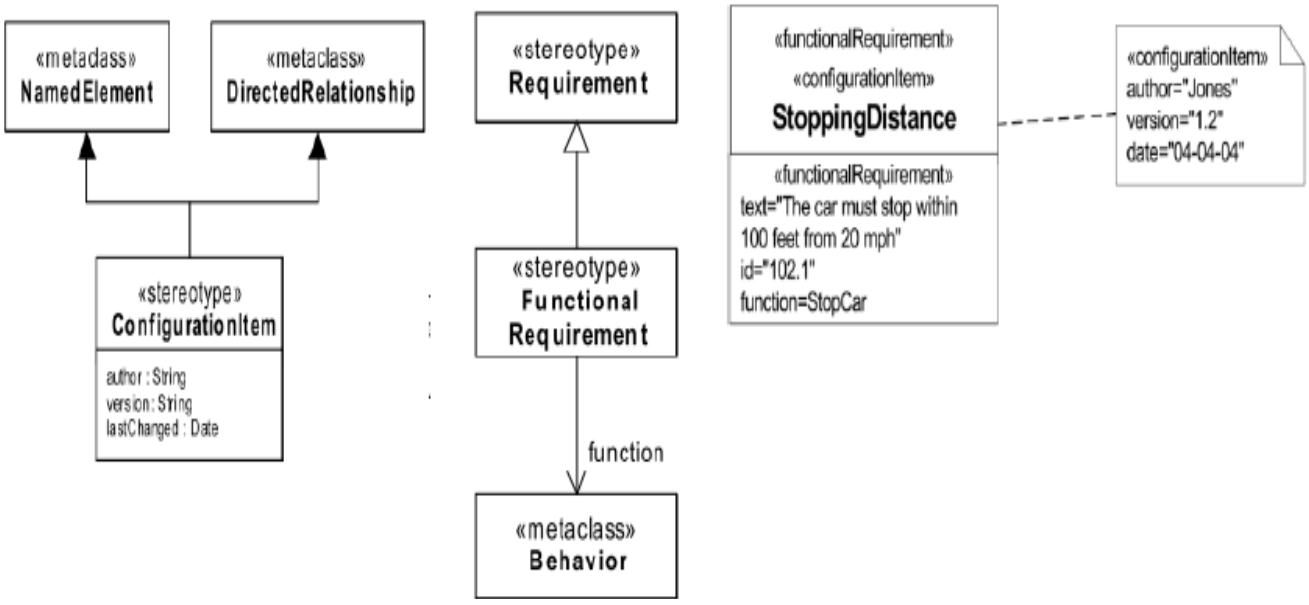


Figure III.1 Exemples de stéréotype SysML

La Figure III.2 « Diagrammes SysML » présente certaines composantes du modèle SysML. Ce modèle se compose d'un ensemble de diagrammes de type **Internal Bloc Diagram** « ibd », **ActivityDiagram** « act », **RequirementDiagram** « req » et **ParametricDiagram** « par ». Ces diagrammes rassemblent différents éléments du modèle : « ibd » les éléments de structure du système (éléments physiques, composants mécaniques ou électroniques), « act » les éléments logiques du système (composant logiciel), « req » les exigences du système et « par » les équations de la physique s'appliquant au système. Entre ces différents éléments, les flèches représentent des liens de type «allocation», entre une exigence et un composant par exemple ou bien entre un composant logique et un composant physique.

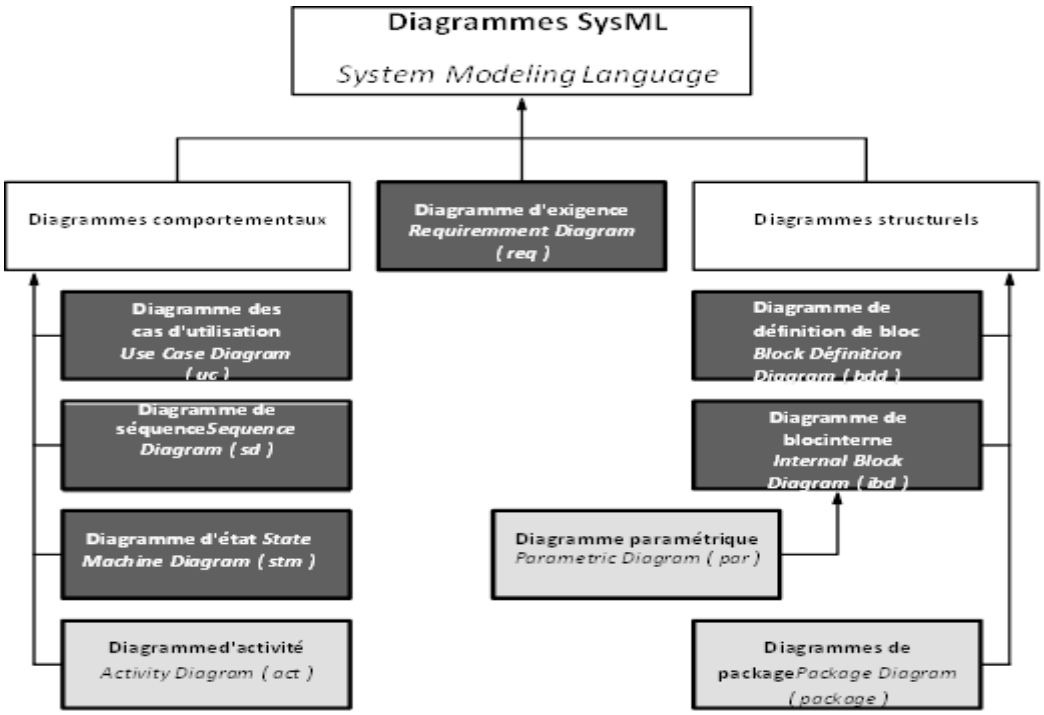


Figure III.2 Diagrammes SysML

1. Application

Dans ce chapitre nous allons modéliser à l'aide de la suite logicielle MagicDraw Version DEMO 18.5, un simple cas d'étude de système embarqué à savoir le lave-linge. Cette application sert à illustrer la méthodologie de conception des systèmes embarqués en détaillant les différents diagrammes Sysml du cas d'étude choisi.

L'interface du logiciel MagicDraw que nous avons utilisé se présente comme suit :

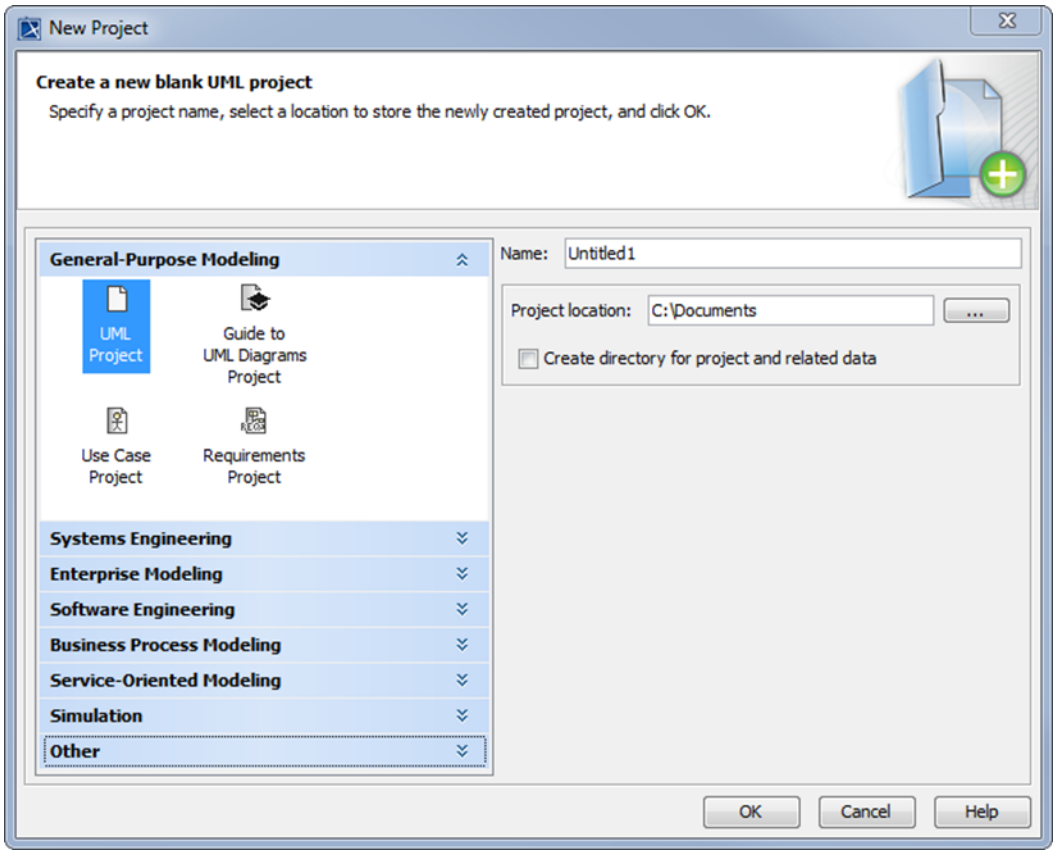


Figure III.3 L'interface du logiciel MagicDraw

Après avoir créé un nouveau projet dans la rubrique Systems Engineering Sysml de MagicDraw , on le nomme « lave-linge ».

a. Diagramme d'exigences d'un lave-linge ménager

D'abord on doit créer le premier **diagramme d'exigence** (req) d'un lave-linge ménager qui permet le lavage de 5 kg de linge.

Le diagramme d'exigence décrit le cahier des charges du système. Il définit les contraintes que doit remplir le système pour répondre aux besoins à satisfaire par lesystème.

On introduit dans le diagramme une exigence globale «Exigences fonctionnelles» et 6 exigences qui lui sont incluses par des liaisons de **Confinement** :

1. Lavage 2. Chauffage 3. Essorage 4. Programmation 5. Remplissage 6. Vidange.

La vidange comprend l'exigence vidange manuelle et comprend un composant, le bloc pompe avec des liaisons de **Complément** : « Refine » et « Satisfy ».

On introduit ensuite dans le diagramme une autre exigence globale «Exigences environnementales» qui inclut 3 exigences : 1. Encombrement 2. Branchement électrique 3. Branchement hydraulique

Le branchement hydraulique comprend l'exigence dimensionnement des raccords.

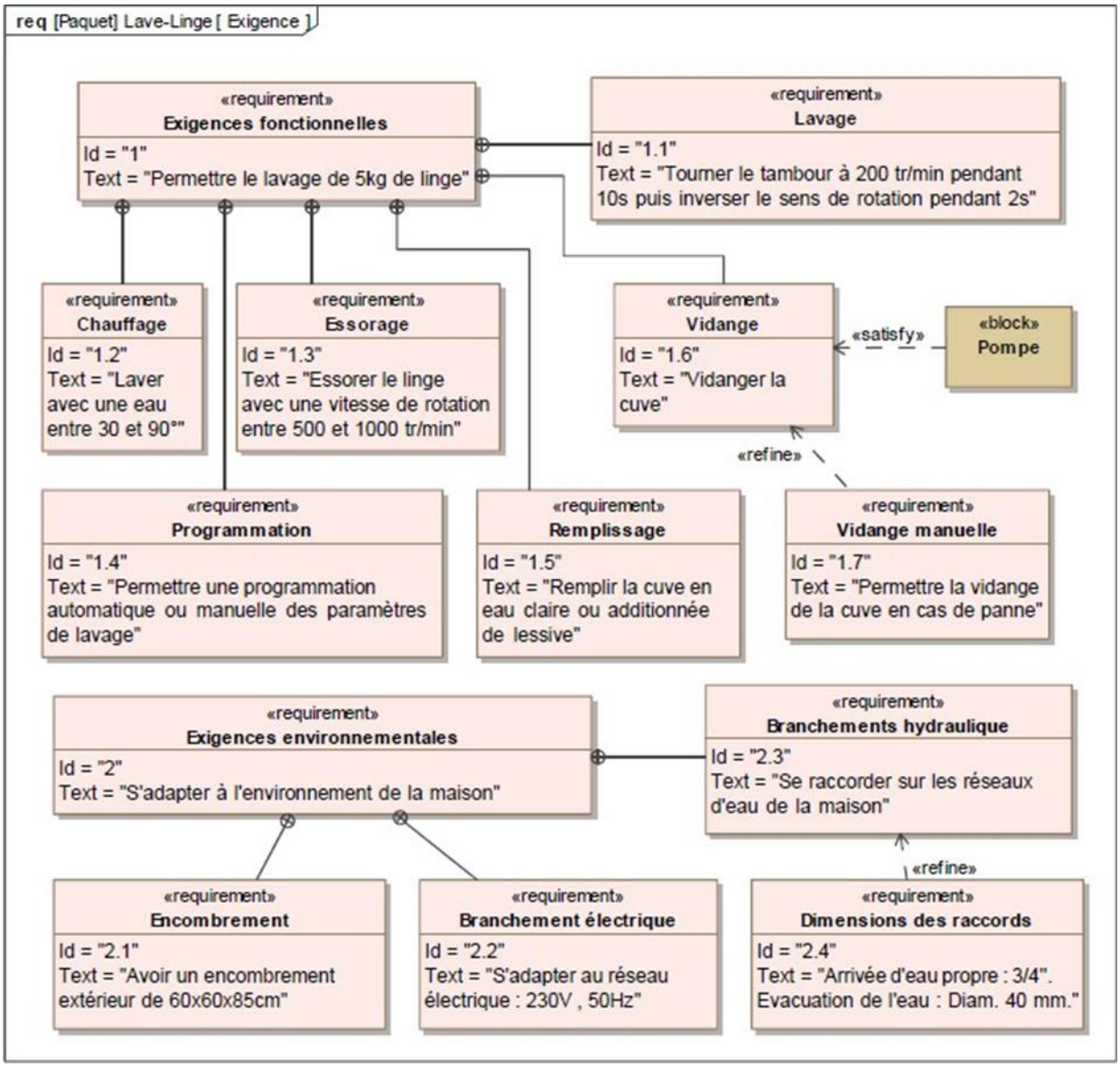


Figure III.4 Diagramme d'exigences d'un lave-linge ménager

b. Diagramme de cas d'utilisation du lave-linge ménager

On crée le deuxième diagramme qui est le **cas d'utilisation.** Le diagramme de cas d'utilisation (UC) permet de représenter les besoins attendus par le système. On se place pour cela du coté utilisateur. Il définit les différentes utilisations du système (dans un ovale) et leurs associations ainsi que les acteurs extérieurs, humains ou non, associés à ces utilisations**.**

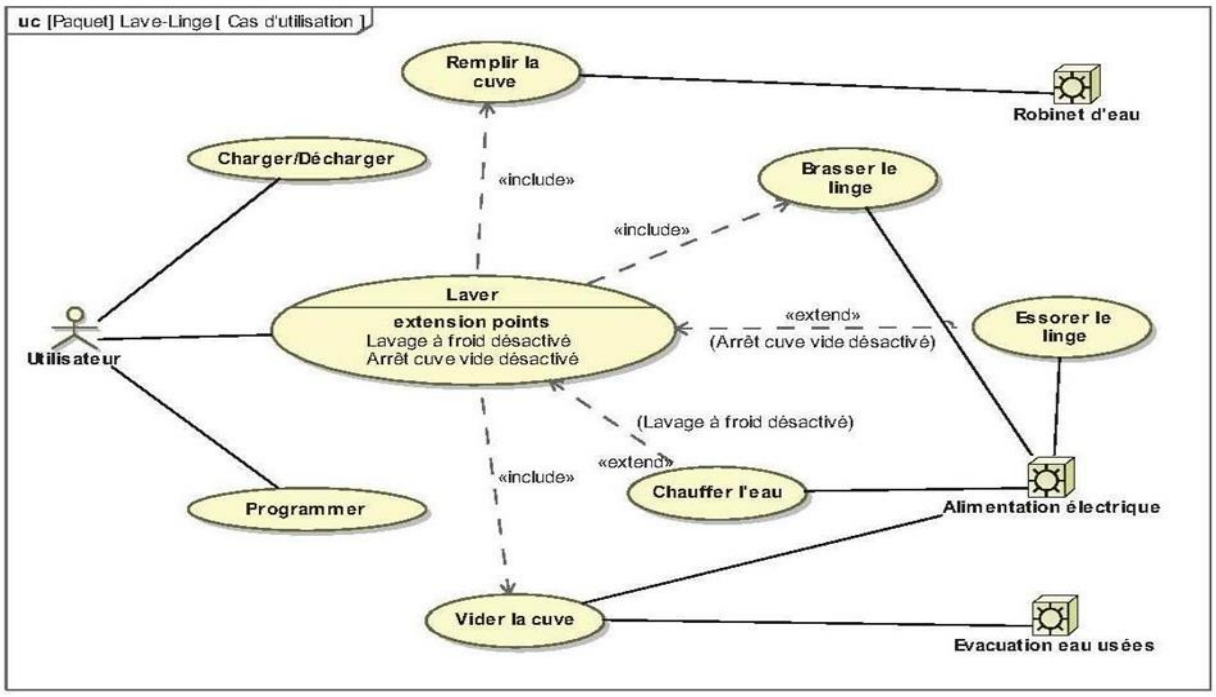


Figure III.5 Diagramme de cas d'utilisation d'un lave-linge ménager.

c. Le diagramme de séquence du lave-linge ménager

On crée le troisième diagramme qui est le **diagramme de séquence.**

Le diagramme séquence (sd) permet de représenter un déroulement chronologique du système. Il décrit chronologiquement un scénario d'un cas d'utilisation en montrant les échanges d'informations entre les acteurs et les instances de bloc.

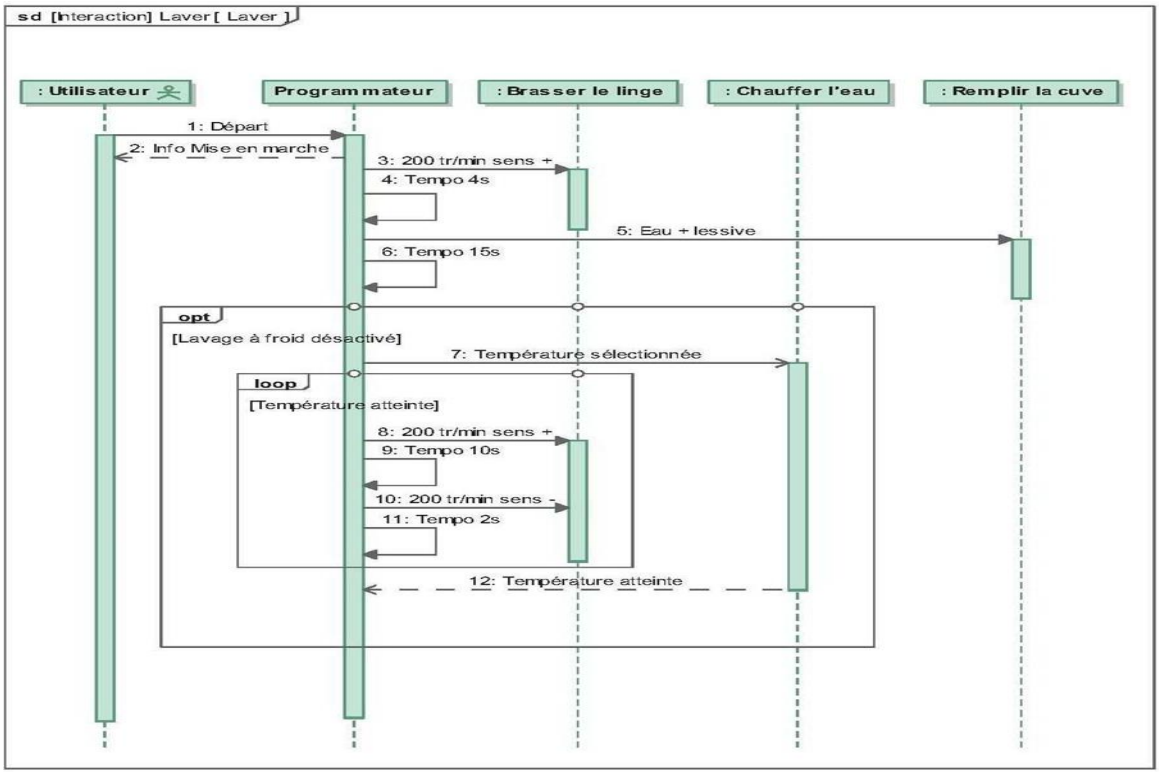


Figure III.6 Diagramme de séquence du lave-linge ménager

d. Le diagramme de définition de bloc

On crée le quatrième diagramme qui est le **diagramme de définition de bloc**. Le diagramme de définition de bloc permet de représenter la structure interne et externe au système. On représente deux diagrammes de définition de bloc (BDD). L'un de contexte présentant la structure externe au système, et l'autre la structure interne du système avec les sous-ensembles qui le composent**.**

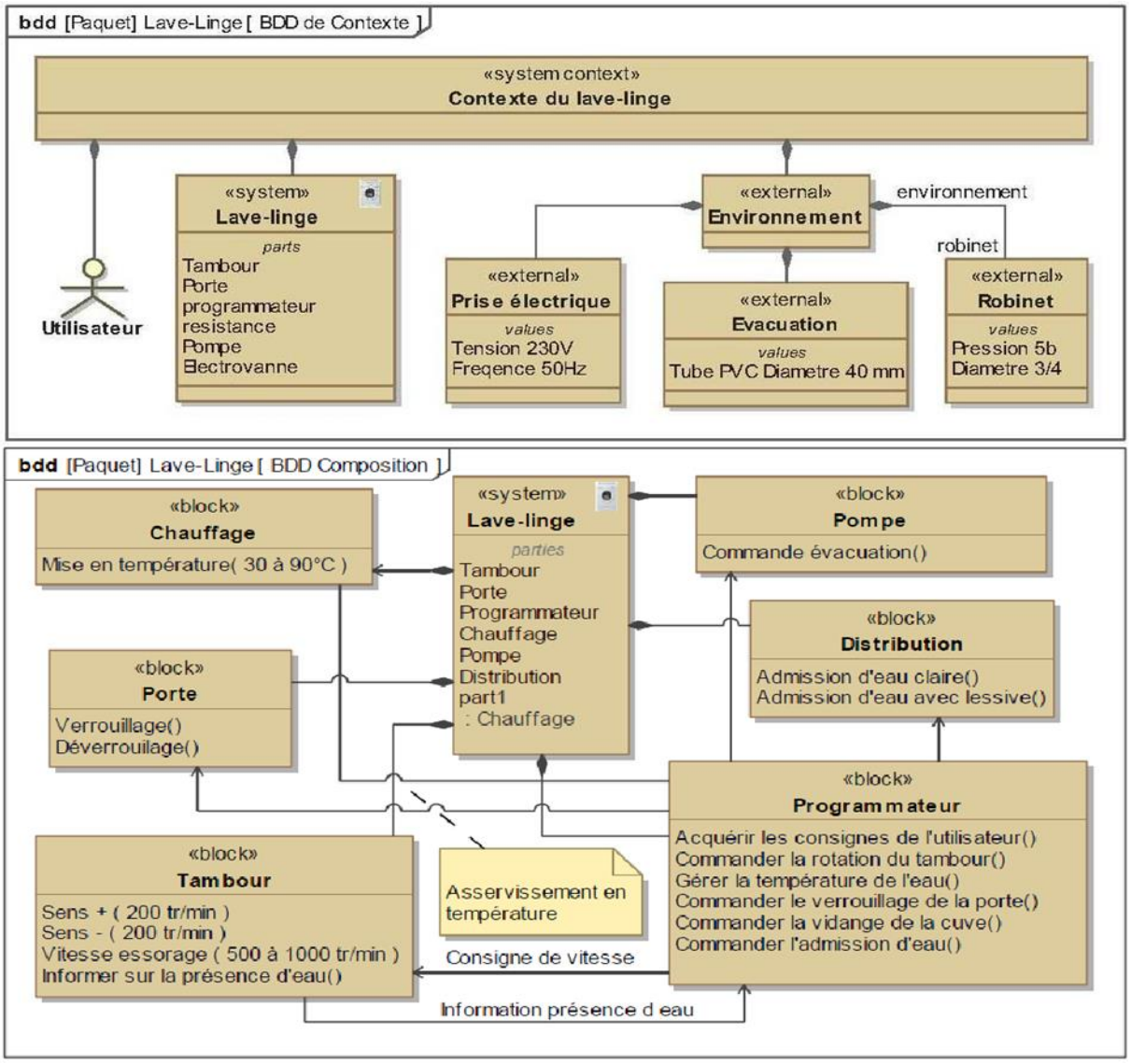


Figure III.7 Diagramme de définition de bloc du lave-linge ménager

<u>Relations entre les blocs</u>

Composition : Lien physique d'appartenance

Agrégation : Lien d'appartenance non physique (Ex : Batterie)

Lien entre un bloc (inclus dans plusieurs blocs) et un plus général

ou Liens d'association (Echange d'énergie de matièreou d'information) mono ou bi directionnel.

e. Le diagramme de bloc interne

On crée le cinquième diagramme qui est le **diagramme de bloc interne**. Le diagramme de bloc interne (IBD) permet de représenter la structure interne voir externe au système. Il reprend un ou plusieurs diagrammes de définition de bloc pour détailler les échanges d'informations, d'énergie ou de matière entre ces différents blocs internes.

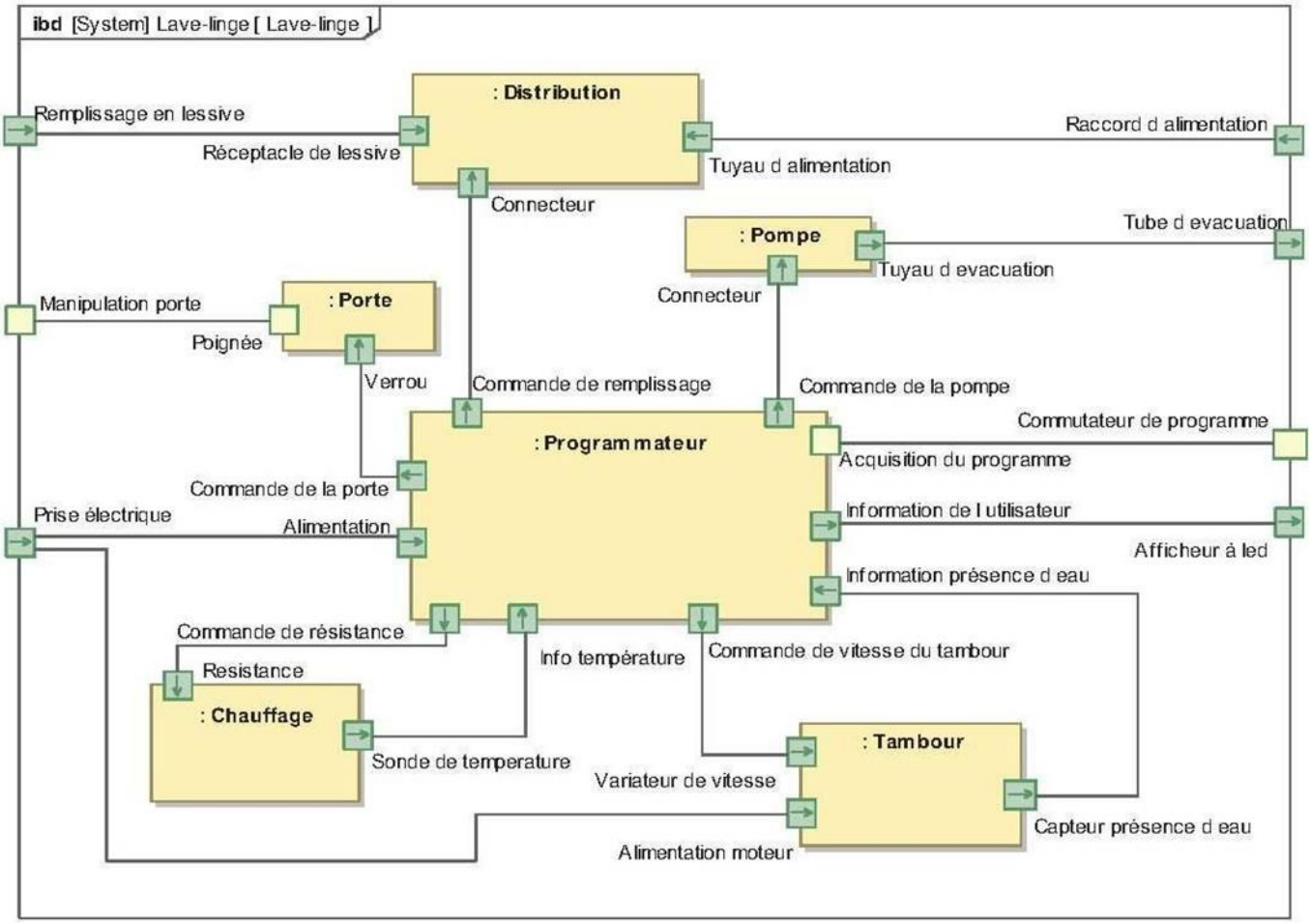


Figure III.8 Diagramme de bloc interne

Ports de blocs

Les ports sont des points d'échange d'un bloc avec l'extérieur.

On trouve plusieurs types de ports :

ou Les ports de flux : Pour des Echanges d'information d'énergie ou de matière Mono ou bidirectionnel.

Les ports standards : Interfaces decommande.

f. Le diagramme d'état

On crée le sixième et dernier diagramme qui est le **diagramme d'état**. Le diagramme d'état permet de décrire le cycle de vie d'un bloc. Il décrit les différents états dans lequel peut être le bloc. Il décrit

également les évènements conduisant aux changements d'états ainsi que les actions à mener lors de ces changement ou lorsque le système est dans un état donné. Ce diagramme est particulièrement intéressant pour décrire le fonctionnement et programmer les systèmes à logiques séquentiels.

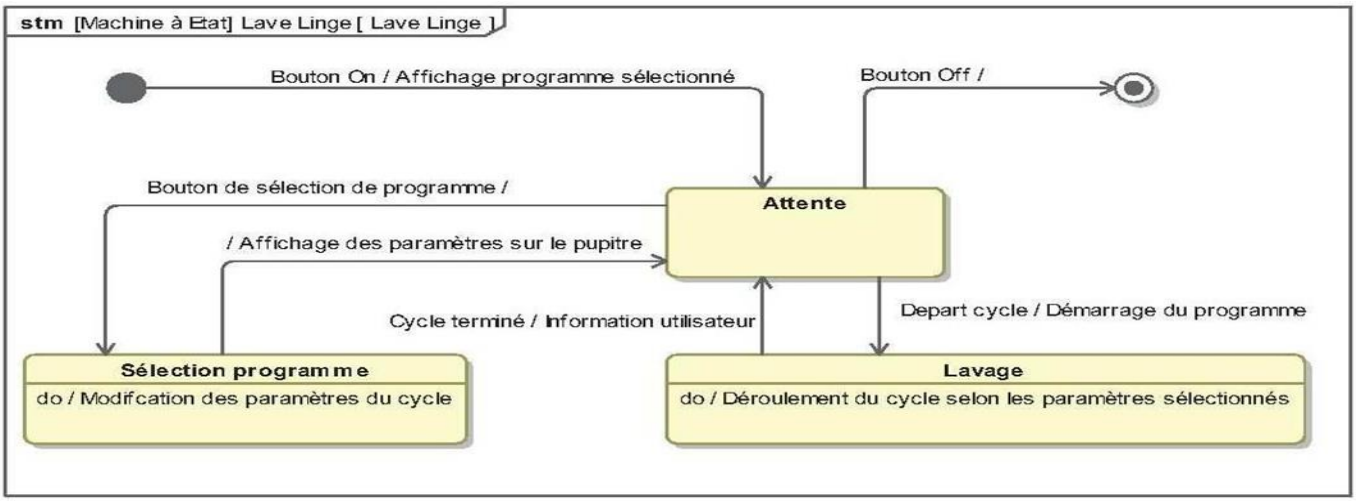


Figure III.9 Diagramme d'état d'un lave-linge ménager

Syntaxe du diagramme

Chaque rectangle correspond à un état donné.

Il est possible d'avoir des états imbriqués les uns dans les autres.

Ce symbole correspond à l'état initial du bloc ou de lamachine

Ce symbole correspond à l'état final du bloc ou de lamachine

A l'intérieur de chaque rectangle d'état apparaît l'action à faire dans cet état

Pour chaque transition entre les blocs on indique l'évènement conduisant à cette transition ainsi que l'action à mener lors de la transition. Ceci avec la syntaxe suivante :

Evènement provoquant la transition / Action à faire lors de la transition

Conclusion : Dans ce chapitre nous avons créé les diagrammes d'exigences en spécifiant les critères de spécification de réalisation issus du cahier des charges de réalisation d'un système embarqué d'un lave-linge de 5kg sur la plate-forme logicielle MagicDraw. Ensuite, pour modéliser l'aspect statique du système, nous avons procédé à la création des diagrammes d'utilisation, de définition externe en relation avec l'environnement et interne avec les sous-ensembles qui le composent et les différentes liaisons physiques qui existent entre eux. Ensuite pour modéliser l'aspect dynamique du système nous avons créé le diagramme de séquence qui décrit le déroulement chronologique du fonctionnement du système en respectant les contraintes temps réel ainsi que le diagramme d'état qui détermine tous les états du système ainsi que transitions inter états.

CONCLUSION GENERALE

Comme nous l'avons vu selon les différents points de vue, SysML apparu en 2007, est un langage utilisant plusieurs outils graphiques offrant au concepteur un modèle global cohérent. Il permet de spécifier les systèmes, de concevoir, de définir et d'analyser leur structure et leur fonctionnement dynamique, de simuler leur comportement afin de valider leur faisabilité avant la réalisation et même le cahier des charges.
Quels sont les fonctionnalités de l'outil de représentation SysML ?

• Il facilite la collaboration transdisciplinaire des différents corps de métiers intégrés au système

• Il permet la mise à jour, le stockage et, surtout, le partage ainsi que l'interprétation facile de toutes les informations nécessaires à la connaissance du système (exemple : le développement de l'A380 d'Airbus a duré 15 ans. Il serait intéressant d'archiver la description totale du système en continu et ainsi connaître l'historique global de son développement. Pour une automobile, le développement dure entre 1 et 2 ans).

• Il permet la modélisation du système à toutes les étapes de son cycle de vie selon tous les points de vue : expression du besoin/contraintes, organisation structurée des composants, définition précise de chaque composant, description attendu du comportement.

• Il permet la mise en relation des différents composants techniques.

• Il permet la validation de solutions technologique (choix) grâce à une simulation basée sur des diagrammes paramétriques dans lesquels on peut saisir les équations de comportement (exemple : choix d'un vérin électrique ou d'un moteur électrique + chaine pour tirer le chariot de la codeuse)

SysML est surtout utilisé par les grandes entreprises et, de plus en plus par leurs entreprises sous-traitantes. Les grandes entreprises faisant office de locomotive et les sous-traitants jouant le rôle de relais pour ce nouvel outil. Quelques exemples :

EADS (EuropeanAeronauticDefence and Spacecompany, 120 000 personnes) : Société européenne, spatial militaire et civil.

Peugeot (200 000 personnes), **Renault** (120 000 personnes): construction automobile française

SKF 45 000 personnes): fabricant suédois de roulements

Bombardier (65 400 personnes): société canadienne, matériel de transport (terrestre, aérien, nautique)

Le travail développé dans ce projet consistait à mettre en pratique un cas d'étude de système embarqué que nous avons modélisé à l'aide de SysML sur une plate-forme de développement logicielle MagicDraw version 18.5 de démonstration. Le système objet de notre étude est un lave-linge ménager de 5kg. Nous avons créés les diagrammes d'utilisation, de définition externe en relation avec l'environnement et interne avec les sous-ensembles qui le composent et les différentes liaisons physiques qui existent entre eux suivant un cahier des charges prédéfini. Ces derniers diagrammes représentent le modèle statique du système embarqué du fait qu'ils projettent juste son architecture, sa composition physique et les fonctionnalités auxquelles il doit répondre. Ensuite nous avons procédé à la création de deux diagrammes modélisant le comportement dynamique du système embarqué à bord du lave-linge à savoir le diagramme de séquence et le diagramme d'état. Le premier modélisant le déroulement chronologique des taches et le second les différents états / transitions du système.

Comme perspective nous proposons d'approfondir la modélisation jusqu'à la génération des codes et la définition précise des architectures matérielles et ce en ajoutant d'autres profils d'UML tel que MARTE (Modeling and Analysis of Real-Time and Embedded Systems) qui modélise comme son nom l'indique le temps réel des systèmes embarqués aussi bien du côté logiciel que matériel avec SysML. AADL (Architecture Analysis and Design Language) est aussi un profil d'UML qui mixé avec MARTE permettent de vérifier des propriétés sur l'occupation mémoire, l'ordonnancement, la consommation d'énergie, la génération de code et donc de pouvoir assister les concepteurs de systèmes embarqués critiques lors du processus de leurs développements.

Bibliographie

[1] B.P.Douglass,**DoingHardTime:DevelopingReal-TimeSystemswithUML,Objects,FrameworksandPatterns**,AddisonWesley,1999

[2] B. P. Douglass**, Real-Time UML :Developping Efficient Objects for Embedded Systems, Addison Wesley, 1999**

[3] J.Rumbaugh,I.Jacobson,G.Booch,**UML2.0GuidedeRéférence**,**Campus Press,2004**

[4] C.Larman,**UML2etlesdesignpatterns,PearsonEducation,2005**

[5] P.Roques,F.Vallée,**UML2enaction:Del'analysedesbesoinsàlaconceptionJ2EE,Eyrolles,2004**

[6] http://www.omg.org

[7] T. Bahill, J. Daniels**, Using Object-oriented and UML tools for hardware design: A case study, Systems engineering vol6 n°1 p28-48. 2003**

[8] J. Bansiya, C. Davis, **A hierarchical model for object oriented design quality assessment. IEEE Transactions on software engineering, vol. 28, No. 1, page. 4-17, 2002**

[9] P. Andersson, M. Host**, UML and SystemC – a Comparison and Mapping Rules for Automatic Code Generation, Proceedings of the Forum on specification and Design Languages conference (FDL), Barcelona, Spain, September 18 - 20, 2007**

[10] B.P. Douglass**, Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems, The Addison-Wesley Object Technology Series, ISBN-13: 978-0201699562, 3 octobre 2002**

[11] OMG**, "Portal of the Model Driven Engineering Community," 2007, http://www.planetmde.org.**

[12] S. Sendall and W. Kozaczynski**, "Model Transformation: The Heart and Soul of Model-Driven Software Development," IEEE Software, vol. 20, no. 5, pp. 42–45, 2003.**

[13] A. Bagnato et al**, "MADES: Embedded systems engineering approach in the avionics domain," in First Workshop on Hands-on Platforms and tools for model-based engineering of Embedded Systems (HoPES), 2010.**

[14] MADES, "EU FP7 Project," 2011, http://www.mades-project.org/.

[15] Object Management Group Inc**, "Final Adopted OMG SysML Specification," mai 2006, http://www.omg.org/cgi-bin/doc?ptc/06-0504.**

[16] OMG**, "Modeling and Analysis of Real-time and Embedded systems**

(MARTE)," 2010, http://www.omg.org/spec/MARTE/1.0/PDF.

[17] W. Mueller et al**., "The SATURN Approach to SysML-based HW/SW Codesign," in IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2010.**

[18] M. Mura et al**, "Model-based Design Space Exploration for RTES with SysML and MARTE," in Forum on Specification, Verification and Design Languages (FDL 2008), 2008, pp. 203–208.**

[19] L. Ober et al.**, "Projet Omega : Un profil UML et un outil pour la modélisation et la validation de systèmes temps réel" 2005, pp. 73: 33–38.**

[20] H. Espinoza et al**, "Challenges in Combining SysML and MARTE for Model-Based Design of Embedded Systems," in ECMDA-FA'09. Springer-Verlag, 2009, pp. 98–113.**

[21] P.H. Feiler, D.P. Gluch, J.J. Hudak**, The Architecture Analysis & Design Language (AADL): An Introduction, Performance-Critical Systems, February 2006, Technical Note CMU/SEI-2006-TN-011, disponiblesur le site official http://www.aadl.info/**

[22] W Ross Ashby **Introduction to cybernetics Science Editions - John Wiley & Sons Inc, Year: 1966**

Résumé :

Les systèmes embarqués ont pratiquement conquis tous les objets technologiques que l'homme à construits jusqu'à maintenant, grâce au développement spectaculaire de l'électronique numérique et des microprocesseurs. La complexité de plus en plus croissante de leur mise en œuvre nécessite d'utiliser des outils de modélisation puissants, orienté objet pour atteindre un haut niveau d'abstraction. Ces modèles permettent d'appréhender tous les aspects statiques et dynamiques des systèmes embarqués qui doivent répondre aux exigences définis par le cahier des charges. UML (UnifiedModelingLanguage) est un outil puissant de modélisation des systèmes. SysML dérivé d'UML permet de modéliser les systèmes embarqués. Dans le cadre de ce projet nous avons créés les diagrammes Sysml de modélisation d'un système embarqué d'un lave-linge ménager comme exemple de cas d'étude. Ces diagrammes portent assistance aux concepteurs et réalisateurs du système embarqué.

Mots clés : UML, SysML, Systèmes embarqués

Abstract:

Embedded systems have conquered virtually every technological object that man has built up to now, thanks to the spectacular development of digital electronics and microprocessors. The increasing complexity of their implementation requires the use of powerful, object-oriented modeling tools to achieve a high level of abstraction. These models make it possible to understand all the static and dynamic aspects of on-board systems which must meet the requirements defined by the specifications. UML (Unified Modeling Language) is a powerful systems modeling tool. SysML derived from UML allows to model embedded systems. As part of this project we created the Sysml modeling diagrams of an on-board system of a household washing machine as an example of a case study. These diagrams provide assistance to the designers and implementers of the on-board system.

Key words :UML, SysML, Embedded systems

ملخص :

لقدغزتالأنظمةالمدمجةتقريبًاكلكائنتكنولوجيصنعهاالإنسانحتىالآن،وذلكبفضلالتطورالمذهلللإلكترونياتالرقميةوا لمعالجاتالدقيقة.

يتطلبالتعقيدالمتزايدبشكلمتزايدفيتنفيذهااستخداماأدواتنمذجةقويةموجهةللكائناتلتحقيقمستوىعالٍمنالتجريد.

تتيحهذهالنماذجفهمجميعالجوانبالثابتةوالديناميكيةللأنظمةالموجودةعلىمتنالأنظمةوالتييجبأنتفيبالمتطلباتالتيتحددها المواصفات. UML (لغةالنمذجةالموحدة) هيأداةقويةلنمذجةالأنظمة.

يسمحSysMLالمشتقمنUMLبنمذجةالأنظمةالمدمجة.

كجزءمنهذاالمشروع،أنشأنامخططاتنمذجةSysmlلنظامعلىمتنالغسالةالمنزليةكمثالعلىدراسةحالة.

توفرهذهالمخططاتالمساعدةلمصممييومطوريالنظامالداخلي.

الكلماتالرئيسية:،UML،SysML، الأنظمةالمدمجة