Alice AKSOY
Stéphane BOISSIEUX
IUP MIAGe, 3ème année
université Joseph Fourier, Grenoble

# REALISATION D'UNE INTERFACE GRAPHIQUE SOUS JAVA POUR L'ENVIRONNEMENT DE PROGRAMMATION ORCCAD


Date : du 21/05/2001 au 07/09/2001.
Lieu : INIRIA Rhône-Alpes à Montbonnot (38).
Maître de Stage : Melle Soraya ARIAS, ingénieur expert au service Robotique.

**Remerciements :**

Nous tenons à remercier vivement Hervé Mathieu, responsable du service Robotique, Vision et Réalité Virtuelle de l'INRIA Rhône-Alpes de nous avoir accueillis pour effectuer notre stage de fin de cycle. Il nous a ainsi offert la possibilité d'acquérir une expérience professionnelle très enrichissante.

Nous remercions tout particulièrement Soraya Arias, notre responsable de stage, pour ses conseils et le temps qu'elle a bien voulu nous consacrer tout au long de ce stage.

Nous souhaitons également remercier toute l'équipe du service: Gérard Baille, Jean-François Cuniberto et Claudie Piscione pour leur aide et pour nous avoir accueillis au sein de leur équipe, ainsi que les thésards et stagiaires de la halle robotique et des projets Sharp et Bip, utilisateurs d'ORCCAD, pour nous avoir fait part de leurs impressions sur le logiciel initial et actuel. Ils nous ont ainsi permis de mieux répondre à leurs besoins.

Nous remercions aussi tout le personnel présent sur le site pour leur bonne humeur et leur enthousiasme à nous faire découvrir les travaux en cours qui touchent des domaines passionnants.

## Table des matières :

## Introduction :

Dans le cadre de nos études à l'IUP MIAGe de Grenoble, il est prévu de réaliser un stage applicatif en Entreprise d'une durée de 16 semaines.

Nous avons eu l'opportunité d'effectuer ce stage au sein du service Robotique, Vision et Réalité Virtuelle de l'Institut National de Recherche en Informatique et Automatique de la région Rhône-Alpes.
Ce service est notamment chargé de développer et maintenir des logiciels utilisés pour la recherche.

Le sujet de stage proposé s'inscrit dans ces travaux de développement et de maintenance; il consiste au passage de l'interface du logiciel ORCCAD (un environnement de programmation permettant de concevoir et de réaliser des commandes robots complexes) actuellement programmée un C++ via Ilog Views, en Java.

# I Contexte du stage :

## I-1 Présentation de l'INRIA :

Créé en 1967 à Rocquencourt près de Paris, l'INRIA, Institut National de Recherche en Informatique et en Automatique, est un établissement public à caractère scientifique et technologique qui mène des recherches avancées dans le domaine des sciences et technologies de l'information et de la communication. Ce domaine inclut l'Informatique et l'Automatique, mais aussi les Télécommunications et le Multimédia, la Robotique, le Traitement du signal et le Calcul scientifique. L'INRIA est placé sous la double tutelle du Ministère de la Recherche et du Ministère de l'Economie, des Finances et de l'Industrie.

L'INRIA a l'ambition d'être au plan mondial un institut de recherche au cœur de la société de l'information.
Sa volonté est de mettre en réseau les compétences et les talents de l'ensemble du dispositif de recherche français dans le domaine des STIC. Ce réseau permet de mettre l'excellence scientifique au service des progrès technologiques, créateurs d'emplois, de richesses et de nouveaux usages répondant à des besoins socio-économiques.

Son organisation décentralisée (5 unités de recherche), ses petites équipes autonomes et évaluées régulièrement permettent à l'INRIA d'amplifier ses partenariats : 7 projets de recherche sur 87 sont communs avec les universités, les grandes écoles et les organismes de recherche. Il renforce son implication dans les travaux de valorisation des résultats de recherche et le transfert technologique : 600 contrats R&D avec l'industrie et un peu moins d'une cinquantaine de sociétés sont issues de l'INRIA.

### a) Quelques chiffres (décembre 2000) :

- Ressources budgétaires :
  - dotation de l'Etat : 442 MF HT,
  - ressources propres : 174 MF HT.
- Ressources humaines :
  - titulaire INRIA : 724,
  - post-doctorants et contractuels : 256,
  - doctorants : 550,
  - chercheurs et enseignants d'autres organismes : 230,
  - conseillers, collaborateurs divers, invités : 430.
- Indicateurs :
  - contrats de recettes actifs : plus de 600,
  - contrats de recettes signés dans l'année : plus de 200,
  - un peu moins d'une cinquantaine de sociétés sont issues de l'INRIA, depuis Ilog, aujourd'hui coté Nasdacq, jusqu'aux toutes dernières, 5 en 1998, 6 en 1999, 11 en 2000,
  - 7 brevets initiaux déposés en 1999 : 1 est en pleine propriété INRIA, les autres sont en copropriétés, 5 avec des industriels et 1 avec une université.

### b) Gestion de projet :

Un projet de recherche est une équipe rassemblant de 15 à 20 personnes autour d'une thématique forte et sur des objectifs scientifiques précis. Ces équipes gèrent de façon autonome leur budget. Les résultats obtenus et les retombées industrielles qu'ils induisent sont régulièrement évaluées.

Une action de recherche est un groupe de chercheurs poursuivant un travail commun sur une thématique spécifique qui peut aboutir à la création d'un projet de recherche.

I-2 Présentation de l'INRIA Rhône-Alpes :

Créée en 1992, l'INRIA Rhône-Alpes est la plus récente des cinq unités de recherche. Menées au sein d'une région en plein essor technologique, les activités de l'unité de recherche INRIA Rhône-Alpes mobilisent plus de 350 personnes dont 220 chercheurs, géographiquement répartis sur trois sites : le site de l'INRIA à Montbonnot, le campus universitaire de Grenoble et l'Ecole Normale Supérieure de Lyon.

Ces activités s'inscrivent dans le cadre des missions que doit accomplir l'INRIA en tant qu'établissement de recherche national, tout en se focalisant sur les objectifs stratégiques poursuivis par l'institut dans le domaine des sciences et technologies de l'information et de la communication.

L'INRIA Rhône-Alpes accueille plus de 130 doctorants, ingénieurs et stagiaires. Ses chercheurs participent à l'enseignement supérieur au sein des universités et grandes écoles de la région (Institut National Polytechnique de Grenoble, université Joseph Fourrier, université Pierre Mendès France, université de Savoie, Ecole Nationale Supérieure de Lyon).

a) Missions de l'INRIA Rhône-Alpes :

En tant qu'unité de recherche de l'INRIA, les principales missions de l'INRIA Rhône-Alpes sont, selon le décret du 2 août 1985 portant sur l'organisation et le fonctionnement de l'institut :

- Entreprendre des recherches fondamentales et appliquées.
- Réaliser des systèmes expérimentaux.
- Organiser des échanges scientifiques internationaux.
- Assurer le transfert et la diffusion des connaissances et du savoir-faire.
- Contribuer à la valorisation des résultats de la recherche.
- Contribuer, notamment par la formation, à des programmes de coopération pour le développement.
- Effectuer des expertises scientifiques.
- Contribuer à des actions de normalisation.

b) Pôles de recherche :

L'INRIA Rhône-Alpes mène ses activités en étroite collaboration avec les laboratoires de recherche publics et privés, nationaux et internationaux, et il entretient des liens privilégiés avec l'institut d'Informatique et Mathématiques Appliquées de Grenoble (IMAG).
Ces activités sont organisées autour de quatre pôles de recherche :

- Maîtriser les systèmes et réseaux informatiques : Réseaux, Parallélisme et Systèmes Répartis.
- Aider à la conception et à la création : Bases de Connaissances, Documents Multimédias, Modèles Cognitifs.
- Percevoir, simuler et agir : Synthèse d'Images, Réalité Virtuelle, Vision par Ordinateur et Robotique.
- Modéliser les phénomènes complexes : Automatique et Calcul Scientifique.

I-3 Présentation du service :

Notre stage s'est déroulé au sein du Service Robotique, Vision et Réalité Virtuelle, de l'INRIA Rhône-Alpes dont le rôle est la mise en œuvre des outils matériels et logiciels pour les expérimentations robotiques des projets de recherche du site.

a) Les moyens humains :

Ce service comporte :

- 3 ingénieurs de recherche : Gérard Baille, Pascal Di Giacomo et Hervé Mathieu.
- 2 ingénieurs experts : Soraya Arias et Laurence Boissieux.
- 1 technicien : Jean-François Cuniberto.
- 1 assistante de service : Claudie Piscione.

b) Les missions du service :

Les missions qui lui sont attribuées sont de trois types :

- *Activités de service :*
  - maintenance des systèmes robotiques,
  - installation et maintenance de logiciels spécialisés,
  - interface entre les utilisateurs et le service informatique,
  - assistance aux utilisateurs.
- *Activités de développement :*
  - mise en place d'expérimentations,
  - développement de logiciels dédiés à la robotique.
- *Activités de recherche :*
  - conception de systèmes robotique,
  - confrontation théorie et expérimentation.

Le but du Service Robotique est de fédérer l'effort expérimental en favorisant :

- Les expérimentations inter-projets,
- La mise en commun des moyens expérimentaux,
- Les outils réutilisables (environnement de développement, machine de vision, …).

c) Les projets concernés :

Les moyens robotiques travaillent avec des projets tels que *« Interaction homme-machine, images, données, connaissance »* et *« Simulation et optimisation de systèmes complexes »* impliqués en robotique et vision :

- *SHARP* : Programmation Automatique et Systèmes Décisionnels en Robotique.
- *MOVI* : Modélisation, Localisation, Reconnaissance et Interprétation en Vision par Ordinateur.
- *BIP* : Conception et Contrôle de Robots Marcheurs et Applications.
- *IMAGIS* : Modèles, Algorithmes, Géométrie pour le Graphique et l'Image de Synthèse.
- *PRIMA* : Développement de Technique pour l'Intégration de la Perception et de l'Action en Robotique.

d) Les moyens techniques :

Le service a à sa disposition différents espaces permettant l'élaboration de ses missions :

- La Halle robotique, dispose des plates-formes suivantes :
  - un bras manipulateur,
  - trois véhicules électriques Cycab,
  - un robot portique dédié à la vision,
  - un robot bipède.

(cf. annexe 1 : Les équipements des Moyens Robotiques)

- Un espace aménagé pour des expérimentations en réalité virtuelle.
- Des systèmes d'acquisition d'images.

## II Présentation du sujet :

Notre stage s'est déroulé au sein des Moyens Robotiques de l'INRIA Rhône-Alpes. Nous avons pris part à ses activités de maintenance et de développement.
En effet, notre projet a concerné un logiciel dédié à la Robotique nommé ORCCAD, *« Open Robot Controller Computer Aided Design »*.

### I-1 Présentation synthétique du sujet :

Le service des Moyens Robotiques de l'INRIA Rhône-Alpes qui maintient le logiciel ORCCAD souhaitait porter sa partie graphique, initialement programmée sous Ilog Views, sous Java pour des raisons de portabilité et de facilité de maintenance.

Le travail proposé a donc consisté en l'étude de ce logiciel afin de le réaliser dans un autre langage, Java et, par là même, le rendre portable sur des stations de travail Linux ou Solaris. Ce travail s'est découpé en plusieurs phases :

- Phase d'analyse : il s'agit de prendre en main le logiciel, de discuter avec les utilisateurs afin d'en comprendre les principales fonctionnalités, d'étudier sa faisabilité et les spécificités du portage en Java,
- Phase de conception : il s'agit ici d'organiser les données et leur représentations graphiques afin de pouvoir construire un application cohérente, modifiable et réutilisable,
- Phase de réalisation : cette phase consiste au développement de l'interface proprement dite.

En outre, il ne s'agissait pas seulement de développer une interface graphique mais aussi de mettre en place les structures de données les mieux adaptées.

### I-2 Présentation du logiciel ORCCAD :

#### a) Utilité:

ORCCAD (**O**pen **R**obot **C**ontroller **C**omputer **A**ided **D**esign) désigne à la fois une méthodologie de programmation d'applications robotiques, et un environnement de programmation qui permet de mettre en œuvre cette méthodologie. Il permet de spécifier, valider et implanter une mission robotique complexe. Ce système s'inscrit dans le cadre des architectures de type « hybride » et est issu des travaux de laboratoire robotique de l'INRIA Rhône-Alpes et des équipes de recherche ICARE et BIP de l'INRIA spécialisées dans le domaine robotique.

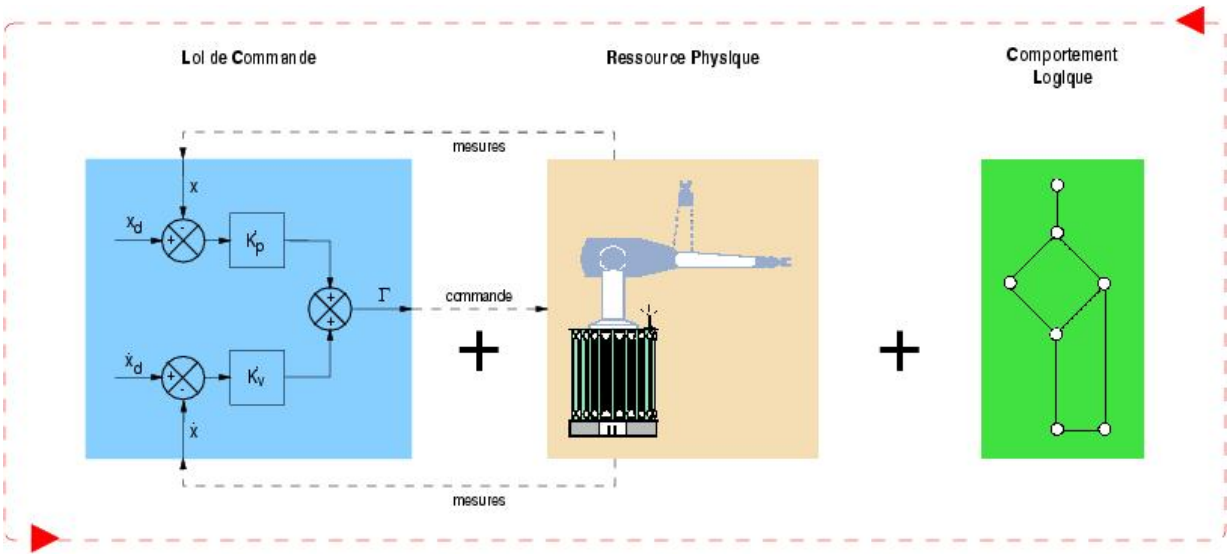
Les applications robotiques, visées par cet environnement, correspondent à des systèmes critiques tant au niveau temps réel que sur le plan de la *sûreté* de fonctionnement. En effet, ces systèmes interagissent fortement avec leur environnement aux travers d'actionneurs et capteurs. De plus, la commande du robot doit être assurée périodiquement suivant des contraintes temporelles fortes pour assurer sa stabilité, et d'autre part, il faut prévenir toute panne ou dysfonctionnement dans la mission à remplir par l'application.

L'objectif clairement établi d'ORCCAD est de permettre *l'implémentation la plus fidèle possible des lois de commandes issues de la théorie de l'automatique*. Pour ce faire, ORCCAD propose à l'utilisateur des outils dédiés qui facilitent la programmation d'une application robotique. En particulier, il permet de valider la logique du système robotique en se basant sur des méthodes formelles et la mise en œuvre temps réel de l'application est assurée par génération automatique de code correspondant à la plate-forme d'exécution. Ainsi, l'utilisateur se voit déchargé de la gestion explicite du temps réel et peut alors se consacrer à l'élaboration de la commande robotique proprement dite et à l'élaboration du scénario de la mission. (cf. annexe 2 : Plaquette de description d'ORCCAD).

b) Architecture:

D'un point de vue méthodologique, ORCCAD aborde la programmation d'applications robotiques suivant l'approche *« contrôle commande »*. Cette méthodologie s'articule suivant deux niveaux d'abstraction : le niveau *fonctionnel* (ou commande) et le niveau *contrôle*.

Le niveau fonctionnel va fournir des actions élémentaires qui fournissent la commande robotique, il manipule une entité, appelée **Tâche Robot** (ou TR) constituée d'une ressource physique, une action élémentaire ou loi de commande et d'un comportement logique, ces grains de base étant eux-mêmes modélisés par une entité appelée **Module**.
On a donc le schéma suivant :



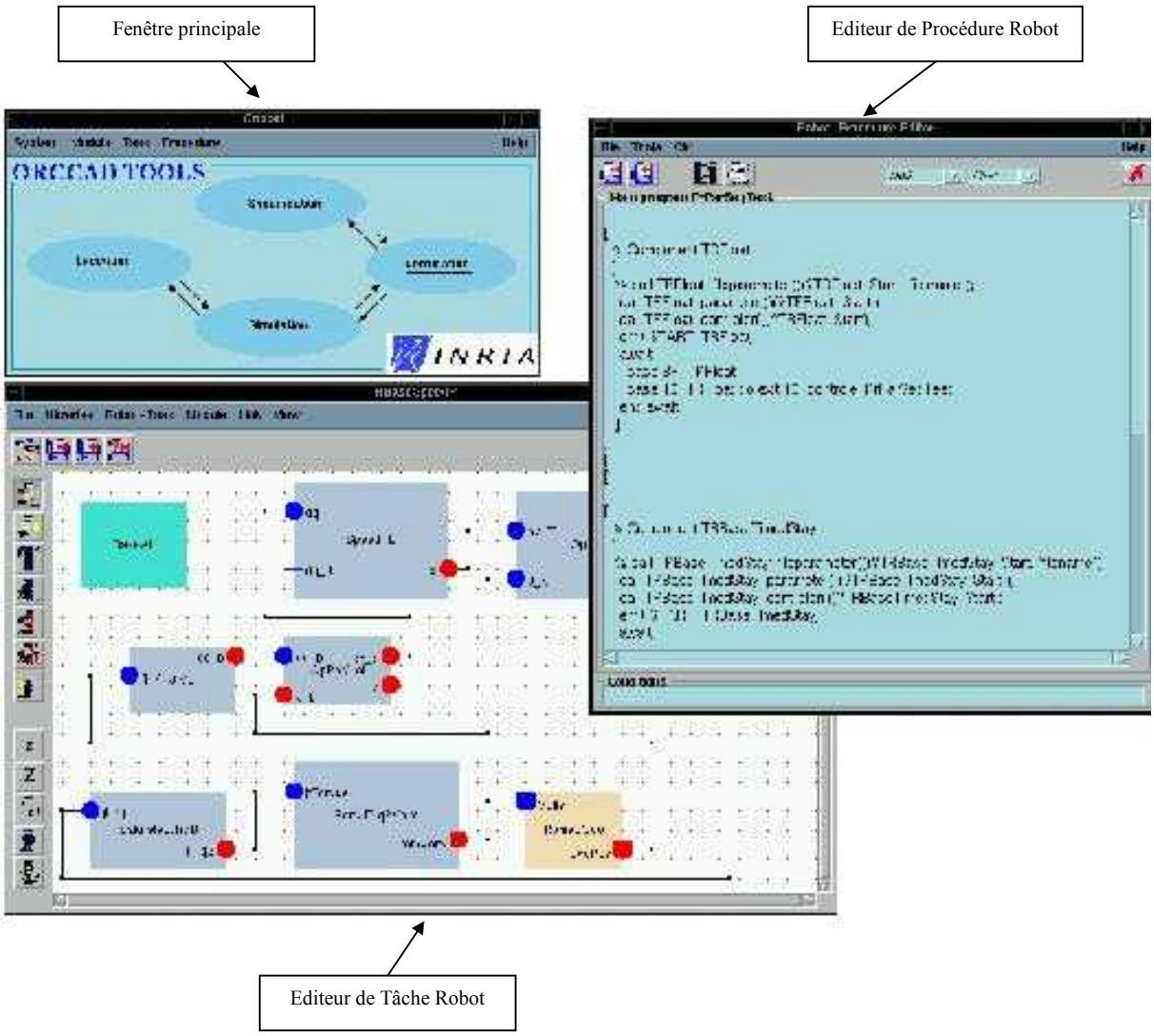

Le niveau contrôle va combiner ces actions élémentaires de manière logique et hiérarchique pour constituer des applications, il manipule une entité appelée **Procédure Robot** ou (PR).

Ces deux niveaux communiquent entre eux par des *événements discrets*. Ces événements permettent de lancer les actions, de les interrompre (par exemple, bonne fin, anomalie dans l'exécution, etc).

Cette méthodologie est mise en œuvre dans un environnement de programmation qui distingue strictement quatre étapes dans le cycle de programmation d'une application robotique, à savoir : la Spécification, la Vérification, la Simulation et l'Exécution.

Cet environnement a été bâti comme une boîte à outils extensible intégrant des outils dédiés à chacun de ces aspects de la programmation d'une application robotique : depuis les outils pour spécifier les TRs et les PRs, en passant par la vérification de la partie comportementale et la simulation, jusqu'à la génération de code automatique correspondant à l'application voulue. A chacune des étapes correspond une interface graphique particulière intégrant les outils dédiés. Ces outils communiquent par des classes C++ constituant aussi le noyau logiciel initial d'ORCCAD.

Aperçu de l'interface actuelle :



c) Pourquoi faire évoluer ORCCAD :

Le logiciel ORCCAD repose initialement sur l'environnement graphique de construction d'IHM Ilog Views qui génère un code en C++.

Or l'INRIA souhaite que ce logiciel soit disponible sur station Linux et Solaris afin d'en permettre une large diffusion.

Cependant, pour passer une interface programmée à l'aide d'Ilog Views de Linux à Solaris, il faut d'une part mettre à jour d'autre système (compilateurs, …) mais aussi dupliquer Ilog Views pour utiliser sur chaque type de stations une version adaptée à son architecture.

A ces problèmes de maintenance et de duplication de code, vient s'adjoindre le problème des licences.

En effet, l'utilisation de plusieurs versions d'Ilog Views nécessite l'acquisition de licences coûteuses qui sont autant de freins à une large diffusion du logiciel.

Passer cette interface en Java qui, au contraire, est un langage qui propose de nombreuses librairies graphiques portables et gratuites, semble alors être une solution, ce travail permettant de plus de réadapter quelques fonctionnalités d'ORCCAD qui étaient peu ou mal utilisées.

## III Présentation du travail effectivement réalisé :

### III-1 Approche du sujet :

#### a) Prise en main d'ORCCAD

Pour bien comprendre la philosophie de ce logiciel, nous avons consacré notre première semaine de travail à la lecture du manuel utilisateur, de différents tutoriaux, de documents de spécification, de notes explicatives sur l'architecture même…
(cf. annexe 3 : Tutorial ORCCAD & annexe 5 : Spécification ORCCAD)
Notre responsable de stage était présente pour répondre à nos questions techniques, nombreuses du fait que ce logiciel s'applique au domaine de la Robotique et que nous n'avions aucune connaissance en la matière.

Nous avons ensuite passé plusieurs jours sur le logiciel en lui-même, étudiant le processus de réalisation de chaque étape de la spécification d'une Procédure Robot via l'interface actuelle.

Par exemple, pour la spécification d'un Module, nous avons repéré les symboles associés à chaque action (quelle icône représente quelle action ?), les feedbacks des événements générés par la souris et/ou par le clavier (que se passe-t-il lorsque je clique sur tel objet ?), comment spécifier ses attributs sémantiques (où changer le nom d'un Module ?), comment sont traitées les erreurs de saisies de valeurs incohérentes (que se passe-t-il si je rentre un nom de Module incorrect ?), comment sont gérées les sauvegardes de fichiers (où et comment sauver un Module ?), comment les liens entre un Module et une Tâche Robot sont-ils modélisés (une Tâche Robot a-t-elle ses propres Modules définis uniquement à son niveau ou fait-elle appel à une librairie de Modules partagés ?) …

#### b) Les utilisateurs

Nous avons ensuite étudié le profil des utilisateurs d'ORCCAD.
Comme expliqué dans le manuel, ce logiciel est destiné à être utilisé par des chercheurs en Informatique et en Automatique. Ils sont donc, par leur profession, habitués au maniement de logiciels en tous genres.

La contrainte qui nous a paru la plus importante au regard de ces utilisateurs est que, étant donné que notre travail était de rénover un de leurs outils, il fallait se soucier de ce qu'ils appréciaient ou non dans son interface de manière à proposer une version effectivement améliorée et qui ne nécessite pas de « ré-appropriation » du logiciel.

Pour ce faire, nous avons élaboré un questionnaire à leur attention et nous les avons interviewés. (cf. annexe 4 : Questionnaire utilisateur)
Les résultats ont montré quelques manquements aux règles d'ergonomie de base, une certaine rigidité au niveau de différentes fonctionnalités et divers dysfonctionnements dûs soit à l'utilisation d'Ilog Views, soit à des processus indépendants de l'interface.
De plus, nous avons soumis différentes versions de notre interface à l'approbation des utilisateurs.

c) Les outils

Il nous a fallu par la suite nous poser la question des outils à utiliser pour programmer cette interface.
En ce qui concerne les librairies graphiques construites au-dessus de la couche AWT (Abstract Window Toolkit) de la librairie Java, trois d'entre elles ont particulièrement attiré notre attention : Ilog JViews, Koala toolkit et Java Swing.
Ilog JViews semblait proposer des fonctionnalités très intéressantes et faciles d'implémentation mais ne résolvait pas le problème posé par la diffusion du logiciel.
La librairie Koala toolkit, bien que simple d'utilisation, proposait quant à elle un nombre limité de fonctionnalités.
La boîte Java Swing, que nous avions déjà eu l'occasion de manipuler, nous a donc paru être le meilleur compromis.

En ce qui concerne l'utilisation d'un outil de développement tel que JBuilder, cela ne nous semblait pas très indiqué dans notre cas. En effet, un des buts de ce travail était de fournir un code clair et réutilisable, et ceci nous paraissait plus simple à réaliser en programmant directement, sans passer par un générateur de code.
Par contre, pour ces mêmes raisons, l'utilisation de générateurs de documents tels que Javadoc ou Doxygen paraissait indispensable.

d) Conception et Réalisation

Après avoir fait l'étude approfondie du logiciel, de ses utilisateurs, et des outils nécessaires à la réalisation du projet, nous nous sommes attaqués à la phase de conception puis de réalisation.

Tout d'abord, nous avons réfléchi en étroite collaboration à la mise en place de l'éditeur de Module et de la structure de données nécessaires à la modélisation d'un Module.

Notre responsable tenait à ce que l'on fasse un éditeur de Module en premier lieu afin de pouvoir évaluer la faisabilité du logiciel. Il est certain que, par rapport à cette première version, notre modélisation a été remise en cause à plusieurs reprises puisqu'en intégrant les traitements des Modules à ceux de la Tâche Robot, nous nous sommes trouvés confrontés à des problèmes de structure, notamment au niveau de la distinction entre la sémantique et le graphique.

En ce qui concerne la programmation elle-même, nous nous sommes répartis le travail au jour le jour, l'un s'occupant plus particulièrement de la partie structure de données, et l'autre de la partie interface.

Nous avons ensuite remanié notre code de sorte que les éditeurs soient eux-mêmes modulaires et réutilisables.

Il nous reste à ce jour à finaliser l'éditeur de Tâche Robot, à programmer l'éditeur de Procédure Robot et à réactualiser la structure de données en y intégrant l'entité Procédure Robot.

III-2 Réalisation :

Maintenant que nous avons exposé notre approche du sujet, nous allons nous appliquer à décrire plus techniquement notre travail.

Pour ce faire, nous allons exposer clairement comment nous avons modélisé l'application ORCCAD du point de vue de la structure des données, puis de l'interface en elle-même. nous verrons ensuite comment nous avons développé l'application en expliquant la structure du programme réalisé et l'utilisation de l'outil graphique choisi.

a) Modélisation :

Le principal motif de rénovation d'ORCCAD était de le porter sous l'environnement Java afin de pouvoir s'affranchir d'Ilog Views mais il nous a aussi été demandé d'anticiper au mieux les problèmes de maintenance liés aux besoins de modifier certains paramètres du logiciel.
La modélisation de l'application s'est donc faite avec le soucis de distinguer deux niveaux de modélisation : une partie propre aux données et leur organisation et une partie propre aux différentes interfaces, ceci permettant de pouvoir modifier la structure de données sans avoir à actualiser la partie graphique en conséquence.
Précisément, chaque entité est représentée par un objet qui lui-même est représentable graphiquement.

- Structure de données :

  - *Les Modules :*

Un Module est donc un bloc élémentaire, situé à la base de la spécification de la Tâche Robot. Initialement, ORCCAD propose des Modules de trois types : Module Algorithmique (ModuleAlgo), Module Ressource Physique (ModulePhy), et Module Automate. L'expérience sur l'utilisation du logiciel a montré que le Module Automate en tant que tel n'était pas indispensable à la spécification d'une Tâche Robot. En effet, il servait uniquement à récupérer des événements produits par les Modules et ces événements peuvent être récupérés directement par le système. De plus, c'était une entité dont les caractéristiques sont fortement liées à une Tâche Robot donnée et, par conséquent, difficilement réutilisable. Il nous a donc été demandé de supprimer ce type de Module.
Par ailleurs, il semblait utile de mettre en place un nouveau type de Module : le Module Capteur (ModuleSens). C'est un Module similaire à la Ressource Physique mais qui n'est pas commandé. Sa seule utilité est de restituer son état.

*Exemple d'un Module Ressource Physique :*

Un moteur renvoie la vitesse de rotation de l'essieu qu'il active. Cette vitesse est une fonction de la tension qu'il reçoit et peut , par conséquent, être contrôlée.

*Exemple d'un Module Capteur :*

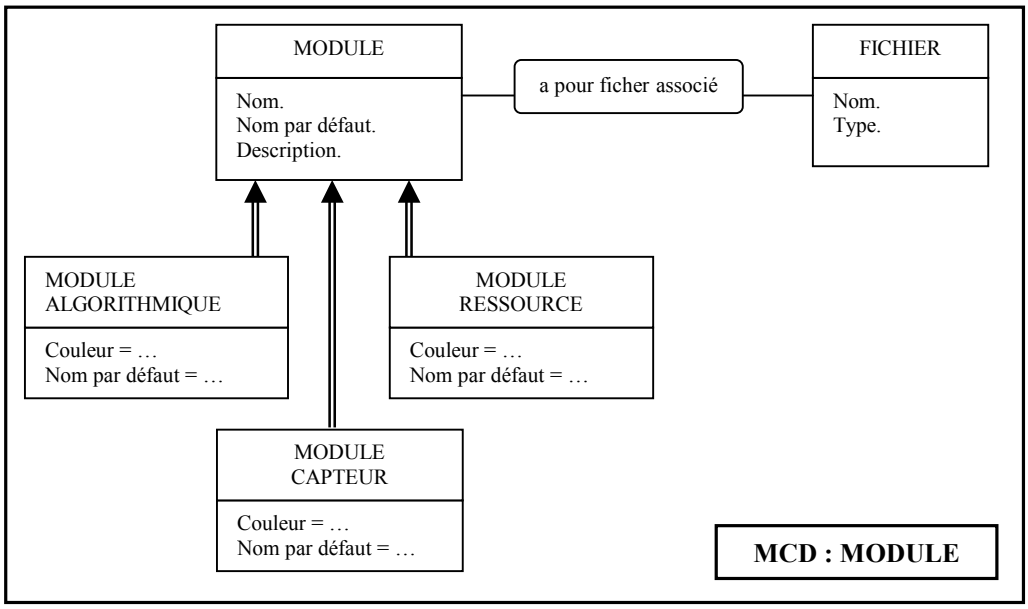
Un thermomètre renvoie uniquement la température ambiante de son environnement avec une périodicité spécifiée. On peut éventuellement générer un événement si la température est trop basse ou trop élevée mais on ne peut agir directement dessus.

*Exemple d'un Module Algorithmique :*

Pour qu'un moteur tourne à une certaine vitesse tout en ayant une température acceptable, il faut calculer à quelle tension le soumettre en fonction de sa vitesse et de sa température actuelle et lui envoyer cette tension.

Ces modifications des éléments de base du logiciel nous ont amenés à réfléchir à une structure des entités Module telle qu'elle puisse évoluer facilement avec les besoins de la Robotique. Plus exactement, il nous a fallu trouver une solution suffisamment modulaire afin de pouvoir ajouter, supprimer ou même modifier un type de Module facilement sans remettre en cause le logiciel.

Nous sommes arrivés à la structure mettant en œuvre de l'héritage simple suivante :




Ce niveau d'abstraction permet de plus de pouvoir traiter un Module sans se préoccuper de son type. Ils ont une représentation graphique, ils peuvent être édités, ils peuvent avoir des fichiers C++ associés, ils peuvent avoir des Ports d'entrée et/ou de sortie (rappelons qu'un Port est une entrée ou une sortie de données d'un Module) et peuvent donc être traités, tout du moins en partie, de manière analogue.

- *Les Ports :*

Un Module contient donc des Ports qui peuvent eux-mêmes être de plusieurs types : Port de Donnée (PortData), Port de Driver (PortDriver), Port de Paramètre (PortParam) et Port d'Evénement (PortEvent). Leur description est disponible dans les documentations d'ORCCAD.

Un Port modélise les données que peut recevoir ou envoyer un Module.

*Exemple d'un Port de Données :*

Pour contrôler un moteur de sorte qu'il ait une vitesse donnée et une température acceptable, un Module de type Algorithmique doit lui envoyer la tension calculée en fonction de ces données.

Ce Module Algorithmique doit donc avoir un Port de Données.

*Exemple d'un Port Driver :*

Pour être contrôler par un Module Algorithmique restituant une donnée de tension, un Module Ressource Physique représentant un moteur doit pouvoir recevoir et interpréter cette donnée.

Ce Module Ressource Physique doit donc avoir un Port de Driver.

*Exemple d'un Port Paramètre :*

Pour pouvoir calculé la tension à appliquer à un moteur de telle sorte qu'il ait une vitesse donnée et une température acceptable, un Module Algorithmique doit avoir connaissance de la précision avec laquelle il doit fournir cette tension.

Ce Module Algorithmique doit donc avoir un Port de Paramètre.

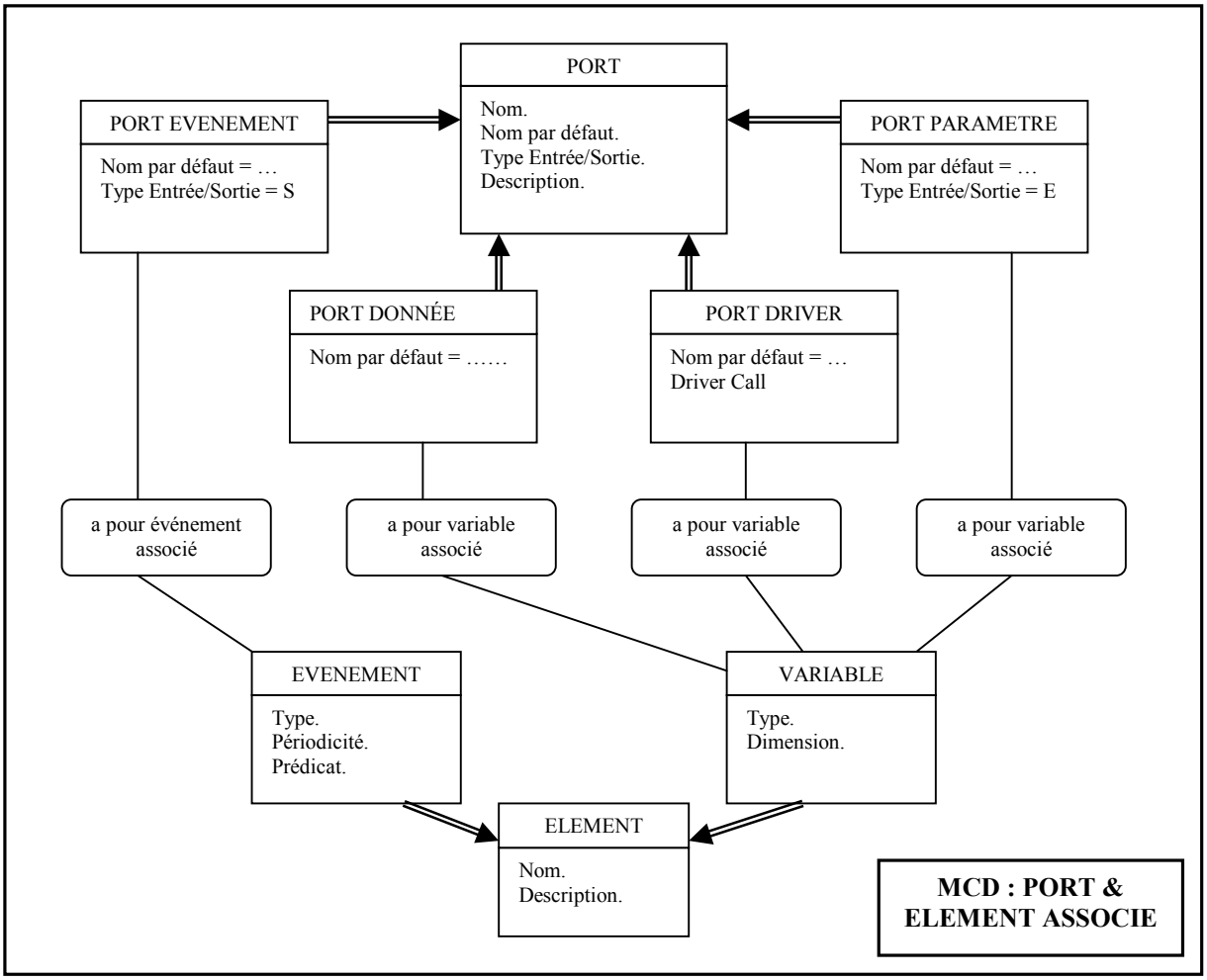
*Exemple d'un Port Evénement :*

Pour pouvoir bloquer un moteur si sa température a atteint une valeur trop élevée, le Module Capteur modélisant le thermomètre doit pouvoir émettre cette information.

Ce Module Capteur doit donc avoir un Port d'Evénement.

En outre, un Port est associé à un élément : soit une Variable (PortAssociatedVariable) avec un nom, un type et une dimension, soit un Evénement (PortAssociatedEvent) avec un nom, un type et une périodicité. Ces éléments représentent le type de données qu'un Module pourvu d'un Port peut échanger avec d'autres Modules. Seul le Port d'Evénement a pour élément associé un Evénement, les autres ports ont une Variable.

Pour les mêmes raisons qui nous ont fait créer une classe abstraite pour les Modules et les Ports, nous avons décidé de modéliser ainsi ces éléments :

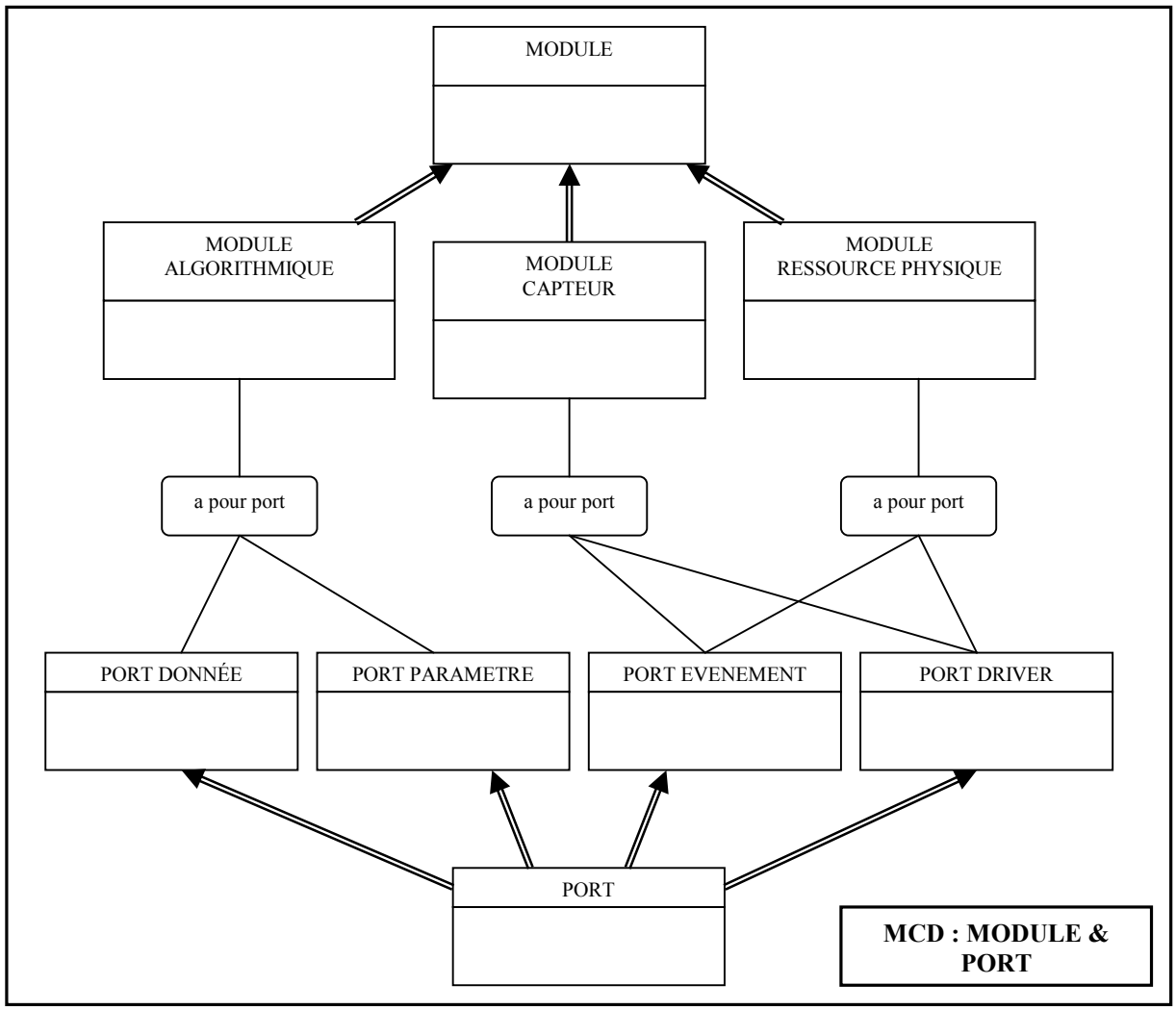



Tous ces Ports ne peuvent être rattachés à tous les types de Modules. Les contraintes suivantes sont à respecter :

| | Module Algorithmique | Module Capteur | Module Ressource physique |
|---|---|---|---|
| Port de Donnée en entrée | **Oui** | **Non** | **Non** |
| Port de Donnée en sortie | **Oui** | **Non** | **Non** |
| Port de Driver en entrée | **Non** | **Non** | **Oui** |
| Port de Driver en sortie | **Non** | **Oui** | **Oui** |
| Port d'Evénement en entrée | **Non** | **Non** | **Non** |
| Port d'Evénement en sortie | **Oui** | **Oui** | **Oui** |
| Port de Paramètre en entrée | **Oui** | **Non** | **Non** |
| Port de Paramètre en sortie | **Non** | **Non** | **Non** |

Ces contraintes nous ont amenés à organiser la structure des données de façon à pouvoir traiter un Module ou un Port sans connaître les détails sur les propriétés de chacune de ces entités lors de la programmation de l'éditeur.

De plus, au même titre que les Modules, les Ports peuvent subir différentes modifications. Nous avons donc opté pour une structure similaire, intégrant de l'héritage :




- *Les Tâches Robots :*

La Tâche Robot (RobotTask) est une action robotique élémentaire (par exemple : faire tourner un moteur à sa vitesse maximale en évitant une surchauffe). Elle est caractérisée par une loi de commande, une ressource physique et un comportement logique, le tout soumis à des contraintes temporelles fortes (le but est de fournir une nouvelle commande à la ressource physique suivant une fréquence particulière).

Elle combine ainsi différents types de Module dont un et un seul est un Module Ressource Physique sur lequel l'action spécifiée s'applique.

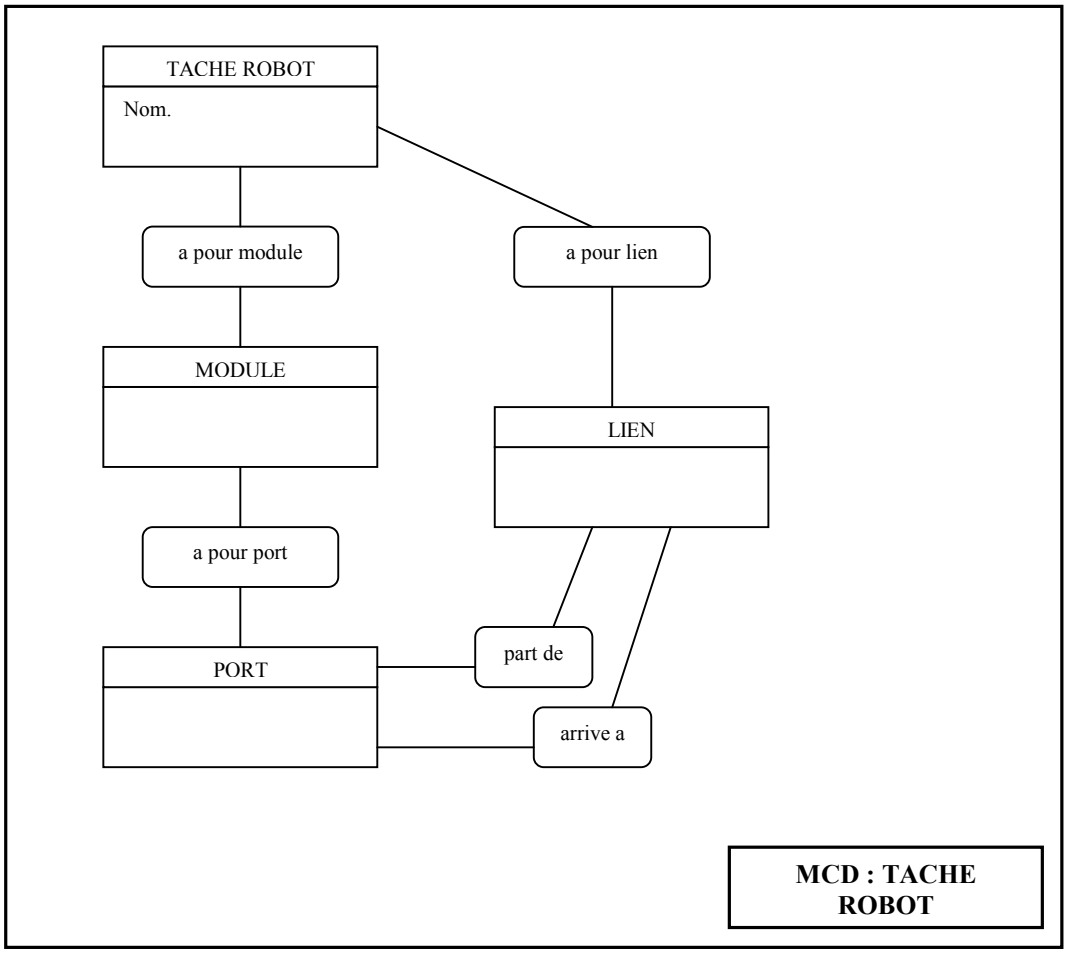
Ces Modules sont reliés entre eux par des Liens (Link). Les Liens sont les objets qui modélisent la propriété qu'ont les Modules de communiquer entre eux via leurs Ports. Ces Liens sont soumis à des contraintes très précises.

En effet, seuls deux Ports de type Données ou Driver et dont les Variables associés ont les mêmes caractéristiques (même type et même dimension), peuvent être reliés avec comme condition supplémentaire que l'un soit un Port de sortie et l'autre un Port d'entrée.

De plus, un Module Algorithmique ayant un Port de Donnée en entrée ne peut effectuer de calcul qu'à la condition que ce Port de Donnée ait été affecté. C'est pourquoi un bouclage au niveau des liens rend l'exécution d'une Tâche Robot impossible. En effet, si un Module attend des données d'un autre Module, lui-même en attente des données du premier, le système est bloqué.

Par ailleurs, un Port d'entrée ne peut recevoir d'information que d'un seul Port de sortie, alors qu'un Port de sortie peut fournir une information à plusieurs Port d'entrée.

D'où la modélisation (ici simplifiée) suivante :




- *La Procédure Robot :*

La Procédure Robot met en place une mission robot complexe, elle constitue la dernière étape de l'élaboration d'une application robotique. Elle est spécifiée comme étant la composition structurée et hiérarchisée d'un ensemble de Tâches Robot et d'autres Procédures Robot.

Cette entité n'est pas représentée graphiquement mais textuellement. L'utilisateur détermine les Tâches Robots et Procédures qu'il veut utiliser, puis, en utilisant le langage Esterel, compose logiquement et temporellement ces différentes entités.

Au jour d'aujourd'hui, cette entité n'a pas encore été implémentée mais nous avons préparé tous les éléments pour pouvoir l'intégrer au logiciel.

- *Modélisation des propriétés graphiques :*

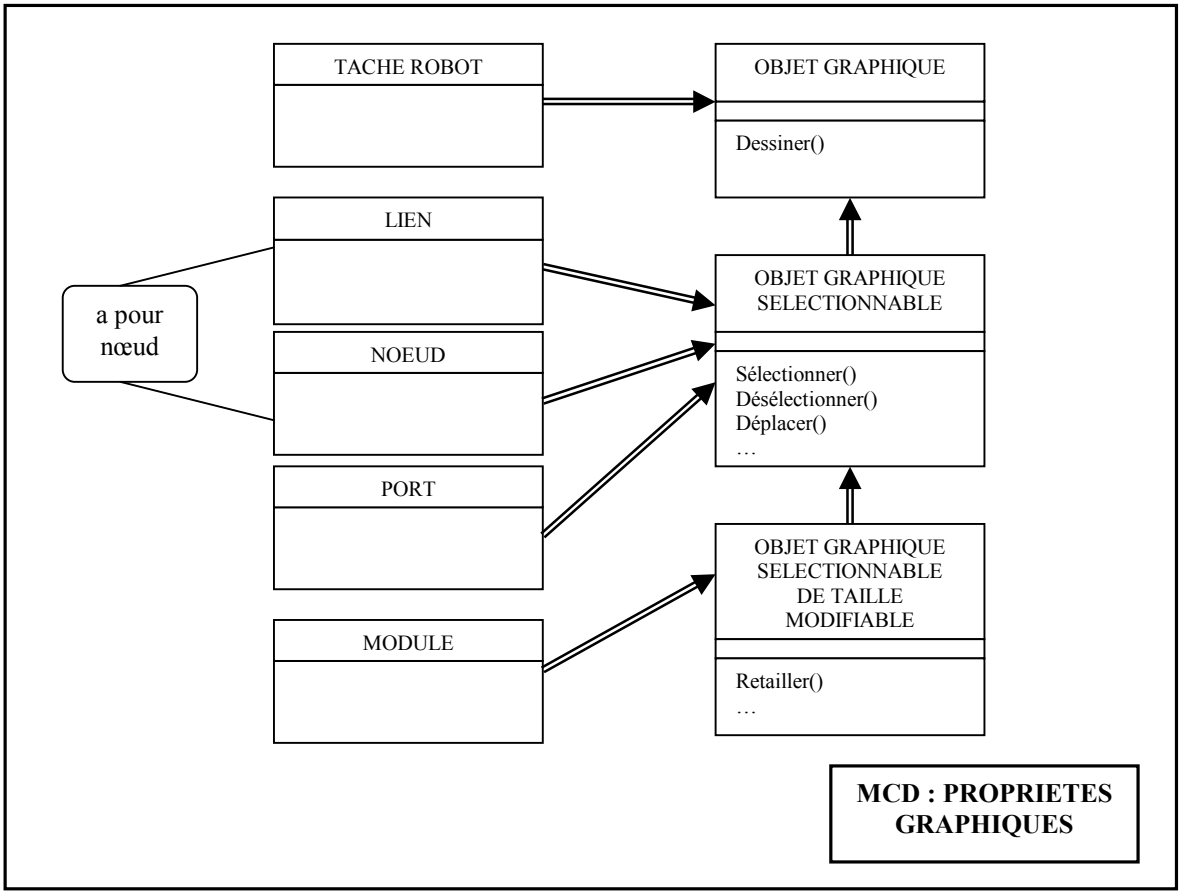
Chaque entité est interprétable graphiquement (sauf la Procédure Robot) dans un éditeur. Chaque objet peut donc être considéré comme un objet graphique.
De plus, certains de ces objets graphiques ont la particularité d'être sélectionnable par l'utilisateur dans un contexte graphique.

Plus en détail, la représentation du Module est un rectangle sélectionnable, modifiable en taille et dont la couleur dépend de son type.

La représentation du Port dépend elle aussi de son type. Le Port de Données est représenté par un disque, le Port de Driver par un carré arrondi, le Port d'Evénement par un losange et le Port de Paramètre par une étoile. Ils sont sélectionnables et de même taille fixe. Leur couleur est variable : rouge pour un Port de sortie, bleue pour un Port d'entrée et jaune si cette propriété n'a pas encore été définie.

Les Liens sont eux aussi sélectionnables au même titre que les Nœuds qui les composent (un Nœud (Node) est un objet graphique nécessaire à la représentation d'un Lien en multi-lignes).

La Tâche Robot est également un objet interprétable graphiquement. Seulement, il est la composition de plusieurs Modules (et donc de Ports) et de Liens (et donc de Nœud). La TR n'est donc pas un objet sélectionnable.



- Interface graphique :

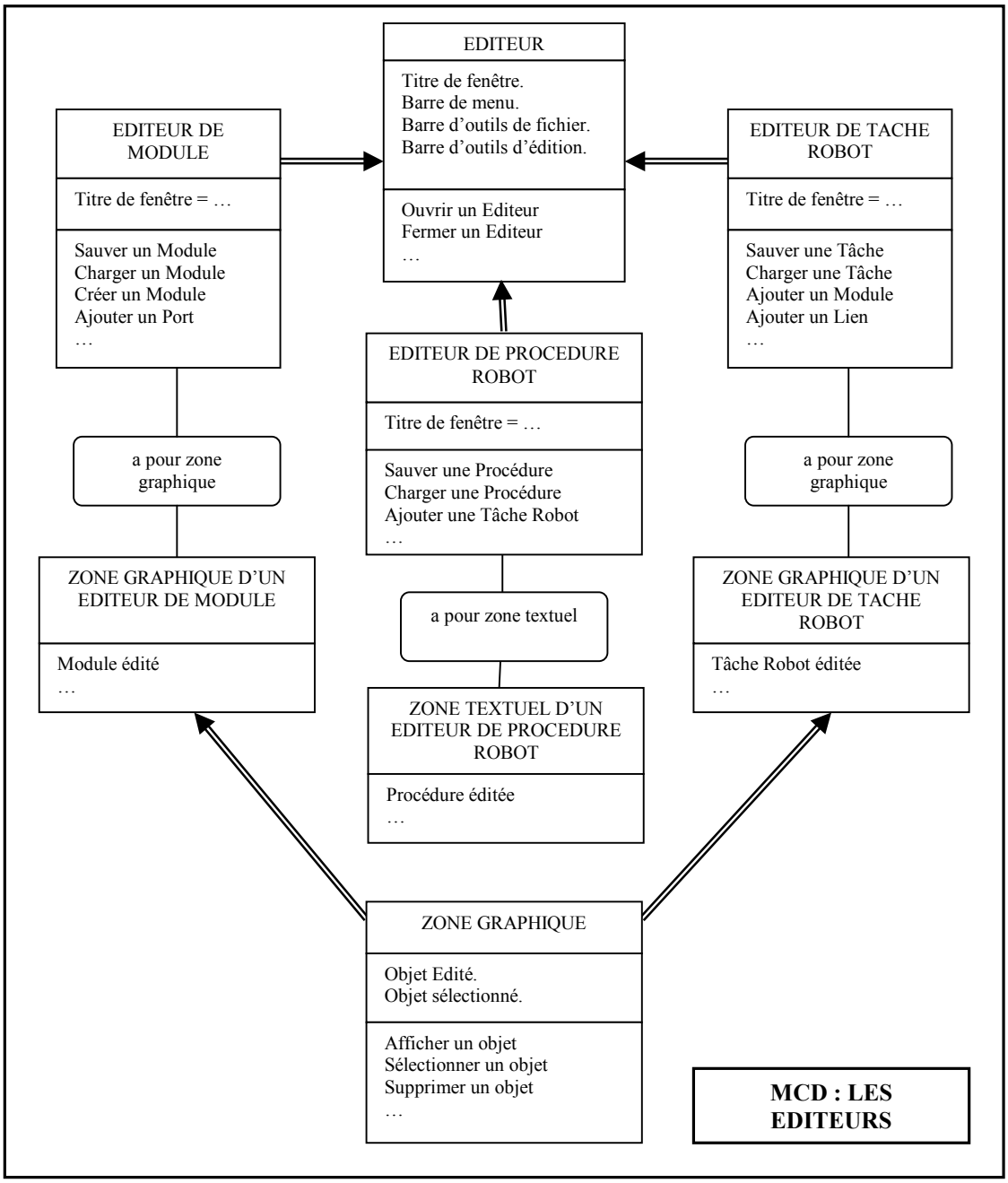
- *Les Editeurs :*

Les différents éditeurs nécessaires à la spécification d'une Procédure Robot sont au nombre de trois : l'éditeur de Module qui permet de spécifier un Module et ses Ports, l'éditeur de Tâche Robot qui permet d'établir des Liens et des contraintes temporelles entre les Modules qui la composent et l'éditeur de Procédure Robot qui permet de relier logiquement et temporellement les Tâches et les Procédures Robot qui la composent.

Bien que devant proposer des fonctionnalités différentes, ces éditeurs ont plusieurs caractéristiques en commun . En effet, ils ont tous trois besoin d'une barre de menu , de deux barres d'outils (une pour les actions de gestion de fichier et une pour les actions d'édition), ….

D'autre part, les éditeurs de Module et de Tâche Robot ont tous deux des éléments graphiques à gérer. Ils ont donc besoin d'une zone dans laquelle un objet graphique est interprétable et modifiable, la seule différence étant que l'une devra gérer un Module et l'autre une Tâche Robot.

Toutes ces considérations nous ont menées à modéliser ces éditeurs comme le décrit le schéma suivant :



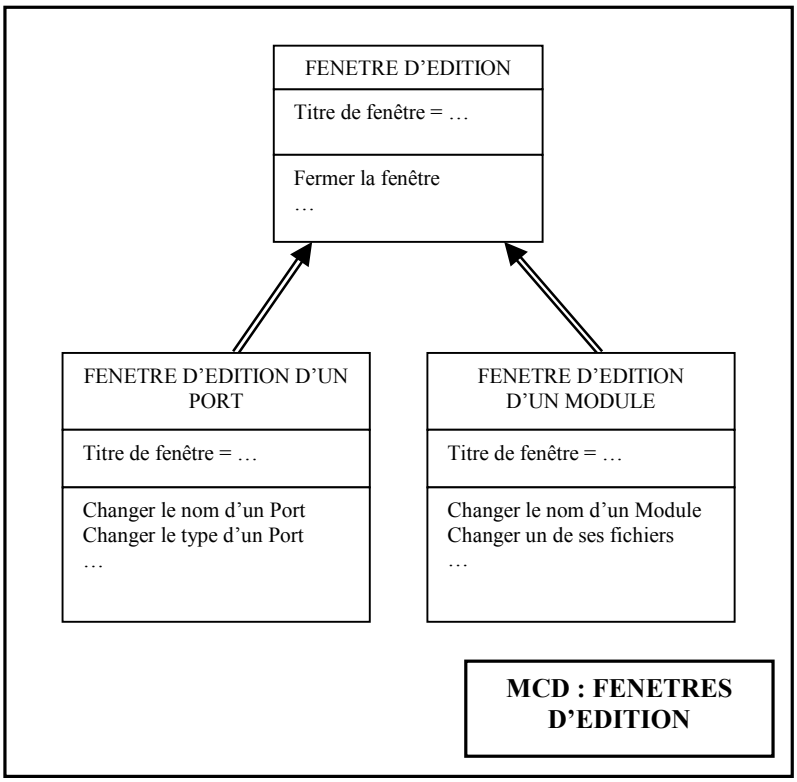
- *Les Fenêtres d'édition :*

Pour permettre l'édition de certaines caractéristiques d'entités représentées graphiquement, il nous a fallu prévoir un système de fenêtres d'édition.

Chacune de ces fenêtres, de la même que les éditeurs, ont des points communs (elles sont de taille fixe, elles peuvent se fermer, elles ont un titres …).

En notre état d'avancement actuel, nous avons modélisé deux fenêtres d'édition : une permettant la spécification des attributs d'un Module (son nom, le nom de ses fichiers associés, …) et l'autre ceux d'un Port (son nom, le nom de son élément associé, …).

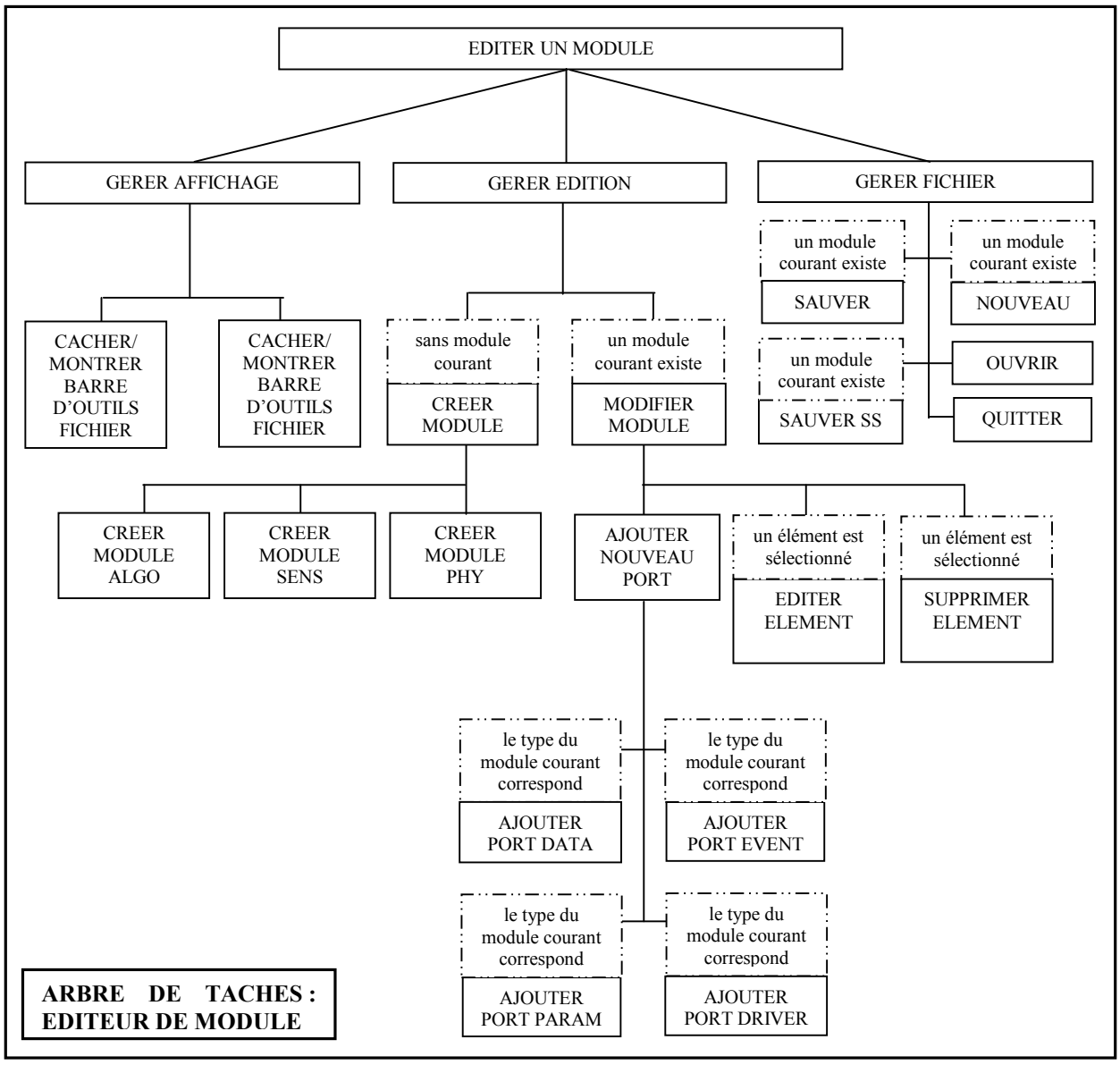
Ce qui nous donne le schéma suivant :

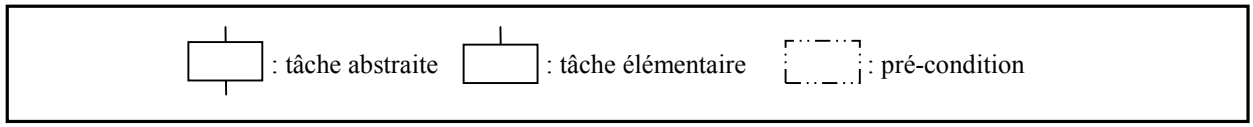


- *Organisation des tâches :*

Pour hiérarchiser les tâches à réaliser depuis ces éditeurs, nous avons étudié les objectifs de chacun en élaborant des scénarios puis nous leurs avons associés un arbre des tâches.

Ces arbres ayant de nombreuses similitudes, nous avons choisi de ne présenter que celui de l'éditeur de Module :

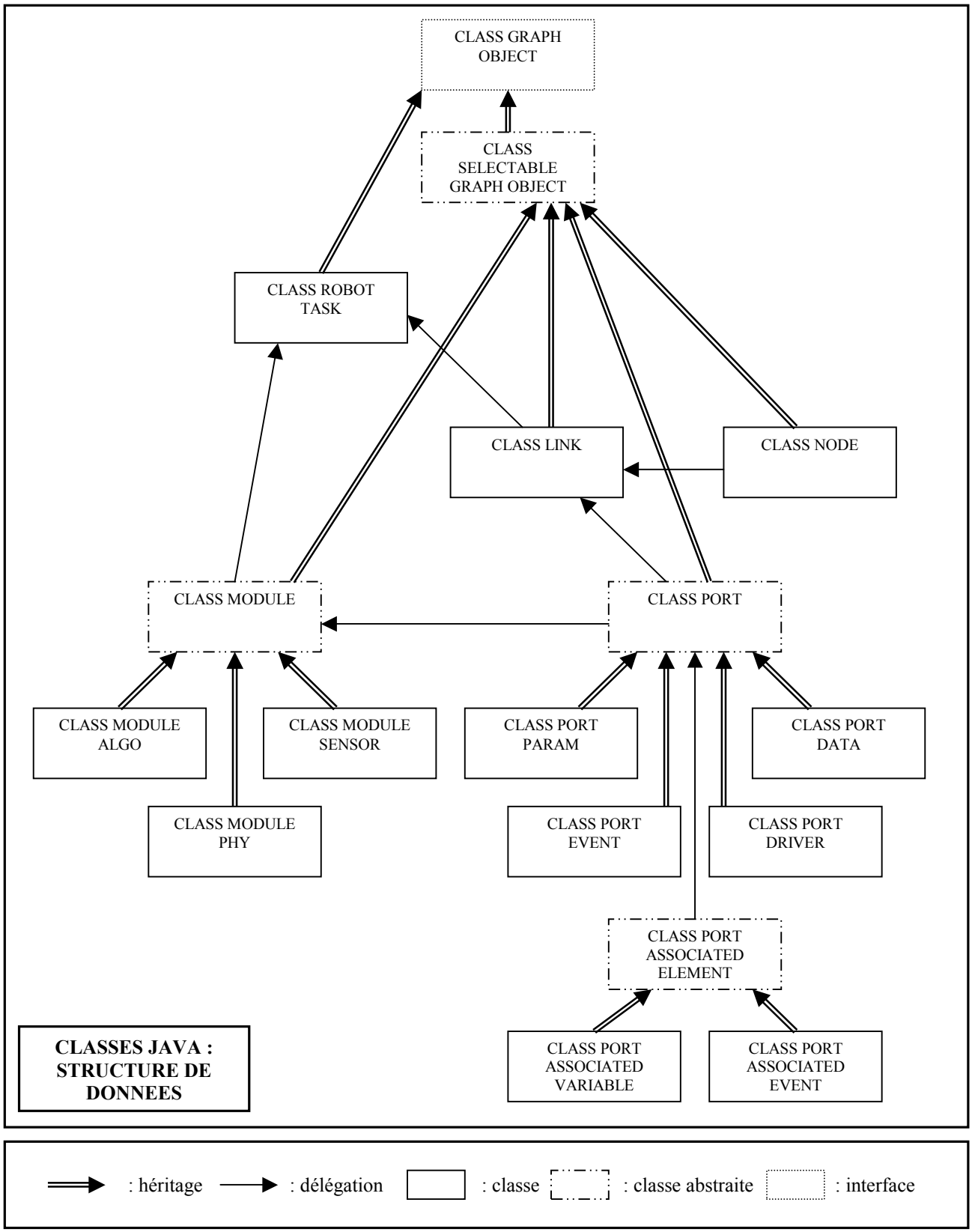


b) Programmation :

- Structure de données :

La modélisation finalement adoptée nous a menés à organiser nos classes Java comme décrit dans la figure suivante :



La marche à suivre pour faire évoluer cette structure est la suivante :

- *Pour ajouter un nouveau type de Module :*

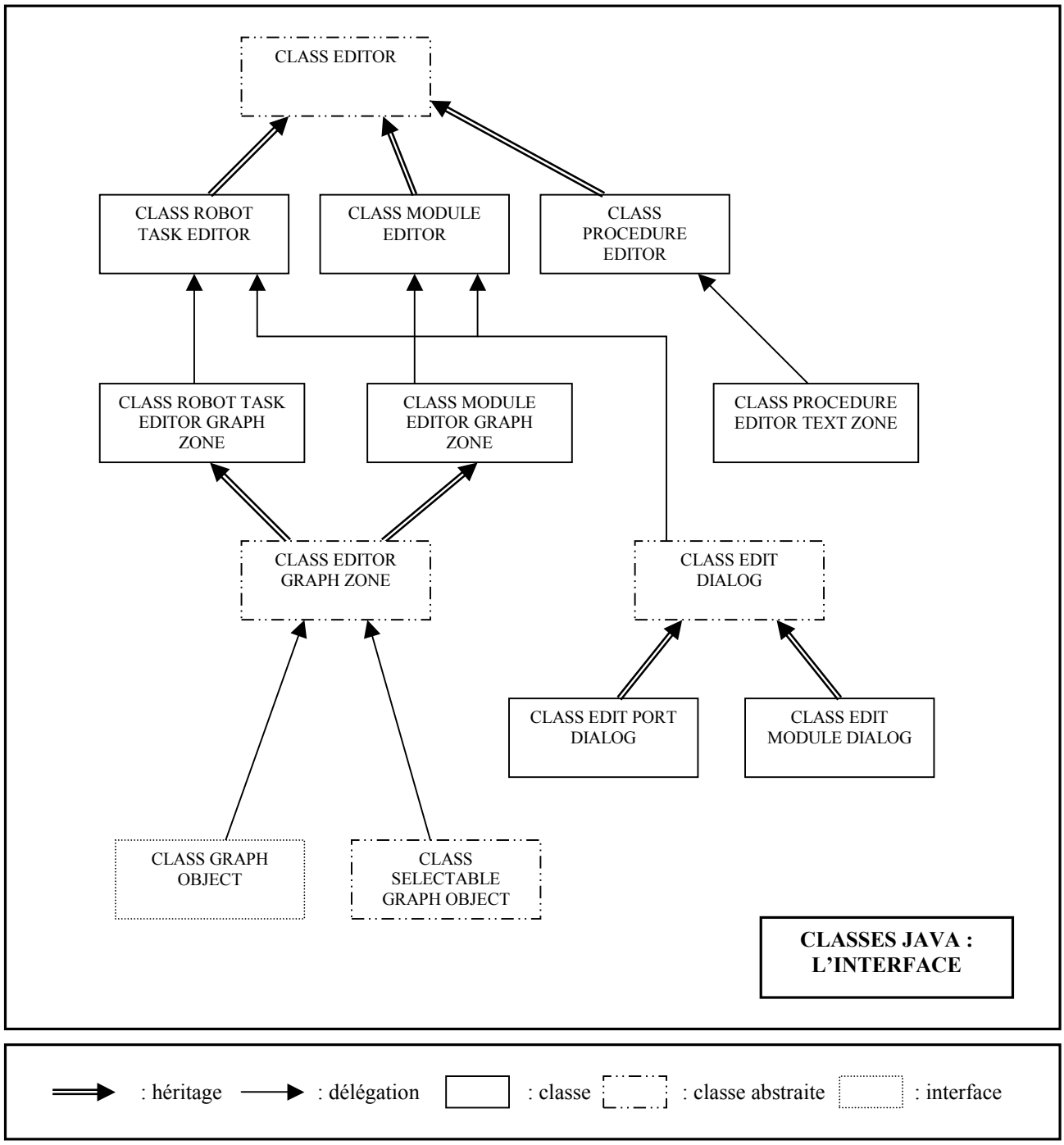
1. Créer un nouvelle sous-classe de la classe Module.
   Ce faisant, cette nouvelle entité hérite de toutes les propriétés communes aux différents types de Module.
2. Définir toutes ses données dont on a fait abstraction dans la classe Module.
   Ce sont les donnés modélisant :
   - La couleur de la représentation graphique du nouveau Module,
   (il faut définir un nouvelle variable statique de type java.awt.Color et l'utiliser dans la méthode Draw(java.awt.Graphics g) pour définir la couleur de dessin de son paramètre g, puis utiliser la méthode de la classe Module qui dessine un objet de cette classe)
   - Les types de Ports qui peuvent lui être associés,
   (il faut programmer les méthodes acceptsPortData(), acceptsPortEvent(), acceptsPortParam() et acceptsPortDriver())
   - Les fichiers qui lui sont associés,
   (il faut utiliser dans son constructeur la méthode addFile(java.lang.String TypeDeFichier, java.lang.String NomInitialDeFichier))
   - Les propriétés spécifiques quant au Port associé.
   (si, par exemple, ce nouveau module accepte des Ports de Données uniquement en entrée, il faut surcharger la méthode addPort(Port NouveauPort) en appliquant sur NouveauPort, si c'est un Port de Données, les méthodes setModifiableType(false) et setType(Port.INPUT_TYPE) )

- *Pour ajouter un nouveau type de Port :*

1. Créer une nouvelle sous-classe de la classe Port.
   Ainsi, les objets de cette classe se verront pourvus des caractéristiques communes aux différents types de Port.
2. Définir toutes ses données dont on a fait abstraction dans la classe Port.
   Ce sont les données concernant :
   - La forme de la représentation graphique du nouveau Port,
   (il faut programmer la méthode draw(java.awt.Graphics g) et créer une forme de taille Port.WIDTH, Port.HEIGHT)
   - Le type de son élément associé,
   (il faut, dans le constructeur de cette nouvelle classe, passer en argument du constructeur de la classe Port une instance d'un objet héritant de la classe PortAssociatedElement)
   - Les propriétés relatives à l'émission ou la réception de données.
   (si par exemple ce nouveau Port ne peut être qu'une sortie de donnée, il faut lui appliquer, à sa création les méthodes setModifiableType(false) et setType(Port.OUT_TYPE))
3. Mettre à jour le comportement de tous les Modules face à ce nouveau Port.
   (il faut rajouter une méthode similaire à acceptsDataPort() à la classe Module et l'implémenter dans toutes les sous-classes de la classe Module puis modifier la méthode addPort(Port NouveauPort) de sorte qu'elle prenne en compte ce nouveau Port)

- Interface graphique :

Nous avons décidé d'organiser les classes Java modélisant l'interface graphique comme suit :



Cette structure à également été programmée de manière à supporter de légers changements sans une remise en cause totale :

- *Pour ajouter une action à un éditeur :*

1. Créer une nouvelle sous-classe de la classe javax.swing.AbstractAction,
   Toutes les actions d'un éditeur sont modélisées par une instance d'une telle classe.
   Pour en définir une nouvelle il faut :
   - Déclarer dans la classe EditorConstants une instance finale de la classe java.lang.String contenant la description de la nouvelle action,
   - Déclarer dans la classe EditorConstants une instance finale de la classe java.lang.String contenant le chemin d'accès à l'icône associée à la nouvelle action,
   - Dans le constructeur de la nouvelle classe héritant de AbstractAction, utiliser les deux constantes précédemment définies pour spécifier les attributs de l'action,
   - Réécrire la méthode ActionPerformed(javax.swing.awt.ActionEvent e), méthode appelée dès que l'action est déclenchée.
2. Déclarer comme champ de la classe définissant l'éditeur un objet de la nouvelle sous-classe de javax.swing.AbstractAction créée au point précédent.
3. Dans le constructeur de la classe définissant l'éditeur, instancier ce champ et l'ajouter au composant adéquat.
   Par exemple, s'il s'agit d'une action concernant la gestion des fichiers, il faut l'ajouter au menu correspondant en appliquant sur l'instance de la classe javax.swing.JMenu représentant ce menu la méthode add(javax.swing.Action a).
   De plus, si cette nouvelle action est destinée à être fréquemment utilisée, il convient de l'ajouter à la barre d'outils fichier en appliquant sur l'instance de la classe javax .swing.JToolBar représentant cette barre la méthode add(Action a).
4. Compléter la méthode refreshButtonAccessibility() de la classe définissant l'éditeur de manière à ce que la nouvelle action soit prise en compte.
   C'est au niveau de cette méthode que sont codées les conditions que le système doit vérifier pour qu'une action soit déclenchable ou non.
5. De même pour la méthode privée de la même classe disableAllAction().
6. Si cette nouvelle action implique la gestion d'un événement par un autre élément de l'éditeur, mettre à jour la classe définissant cet élément.
   Par exemple, pour créer une action dans l'éditeur de Module dont le rôle serait d'informer le système que le prochain click dans la zone graphique aura tel effet, il faut ajouter à la classe ModuleEditorGraphZone un objet traitant ce click :
   - Créer une nouvelle classe héritant de la classe javax.swing.event.MouseInputAdapter et coder les traitements associés aux événements attendus en réécrivant les méthodes MousePressed(MouseEvent e), MouseReleased(MouseEvent e), etc.,
   - Dans la classe ModuleEditorGraphZone, créer une constante de type entier correspondant à la nouvelle action,
   - Compléter la méthode setAction(int Action) de la classe ModuleEditorGraphZone de sorte qu'elle prenne en compte cette nouvelle action,
   - De même pour la méthode privée removeAllMouseListener(),

- *Pour créer une nouvelle fenêtre d'édition :*

1. Créer une nouvelle sous-classe de la classe EditDialog.
   Cette classe est en fait une sorte de « ToolKit » dédiée à la création d'une fenêtre d'édition.
   Y sont définis sous forme de classe tous les panneaux pouvant servir à l'édition de données dans ORCCAD, du panneau contenant le champs nom d'un Module à celui contenant les boutons Apply et Cancel.
2. Déclarer comme champs de cette nouvelle classe, des objets de classes définissant les panneaux contenant les informations à éditer par la nouvelle fenêtre.
3. Si un nouveau panneau doit être construit, le faire en déclarant une classe interne à la classe EditDialog héritant des propriétés de la classe javax.swing.JPanel, sans oublier de la munir de méthode permettant d'accéder aux données rentrées par l'utilisateur.
4. Construire l'action associée au bouton Apply de la fenêtre d'édition.
   C'est dans cet objet qu'est vérifié la cohérence des données rentrées par l'utilisateur.
   Pour ce faire, il faut créer une nouvelle classe implémentant l'interface ActionListener et passer une instance de cette classe en argument du constructeur du panneau contenant ces boutons (il s'agit du panneau définit par la classe EditDialog.ApplyCancelPanel).

La documentation de notre programme sous format JavaDoc est disponible en annexe. (cf. annexe 6 : Documentation du programme)

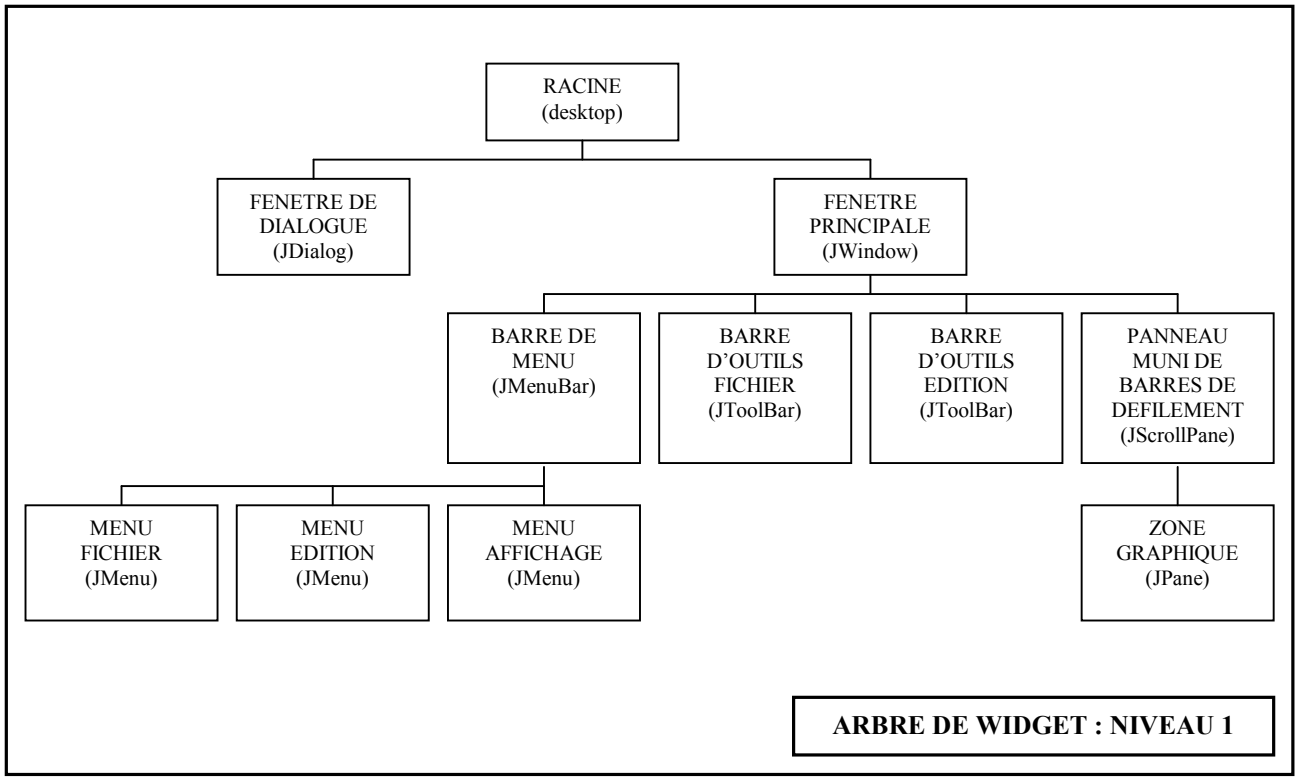
c) Ergonomie :

En ce qui concerne l'ergonomie de nos éditeurs, nous avons autant que faire ce peut implémenté un système de « feedback-proactif ». C'est-à-dire que toutes actions possibles lors de la spécification d'un Module, d'une Tâche Robot ou d'une Procédure Robot sont désactivées, et ce visiblement, lorsque, si déclenchées, elles entraîneraient des erreurs. Par exemple, le bouton de suppression est grisé tant qu'aucun élément n'est sélectionné.

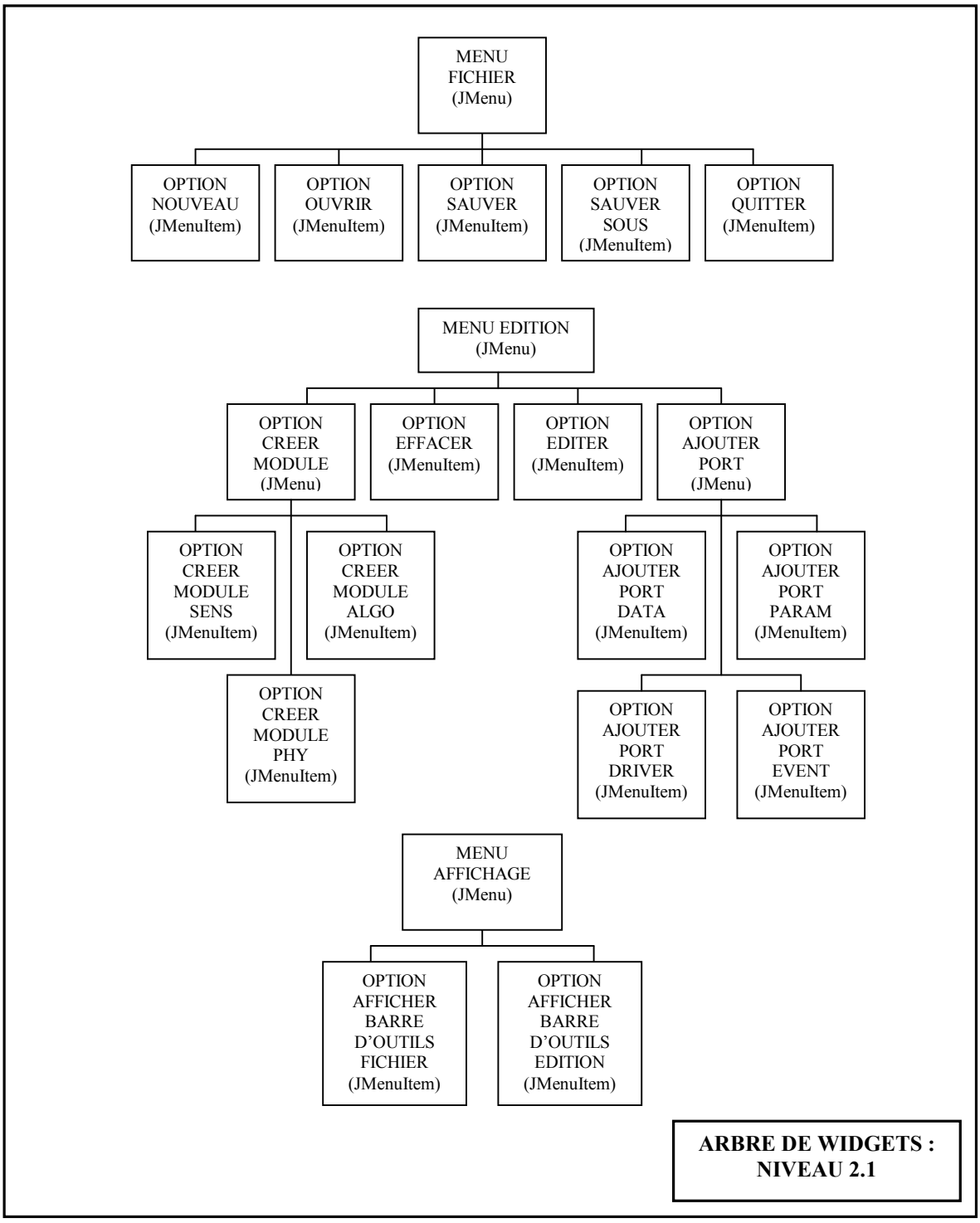
En outre, dans le souci de prévenir tous changements irréversibles commis par erreur, nous avons mis en place un système d'alerte lorsque l'utilisateur est sur le point d'écraser des données. Par exemple, lorsqu'il tente de sauvegarder un fichier et que ce fichier existe déjà, une fenêtre de dialogue apparaît. Elle l'informe qu'il risque de perdre des données et lui propose d'annuler cette action.

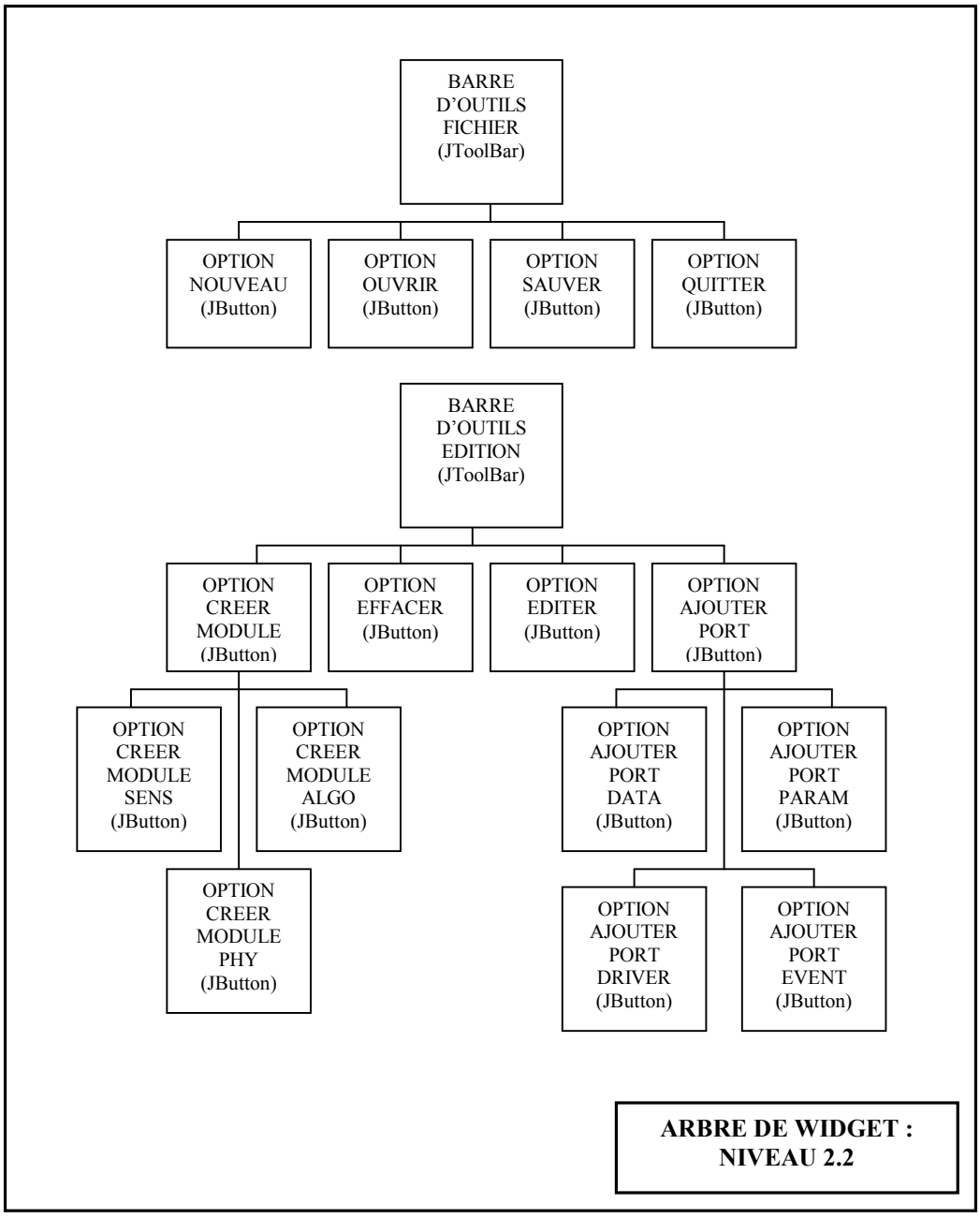
Toutes les actions disponibles sont accessibles par la barre de menu. Pour les plus fréquentes, nous avons mis en place un système de raccourci par les barres d'outils. Nous envisageons d'implémenter également un système de raccourci clavier.

Nos différentes interfaces présentant de grandes similitudes dans leurs structures, nous allons détailler uniquement celles de l'éditeur de Module.
En utilisant la boîte à outils Java Swing, et à partir de l'arbre de tâche que nous avons modélisé, nous sommes arrivés à la structure de « widget » (composant graphique) suivante :




**ARBRE DE WIDGET : NIVEAU 1**

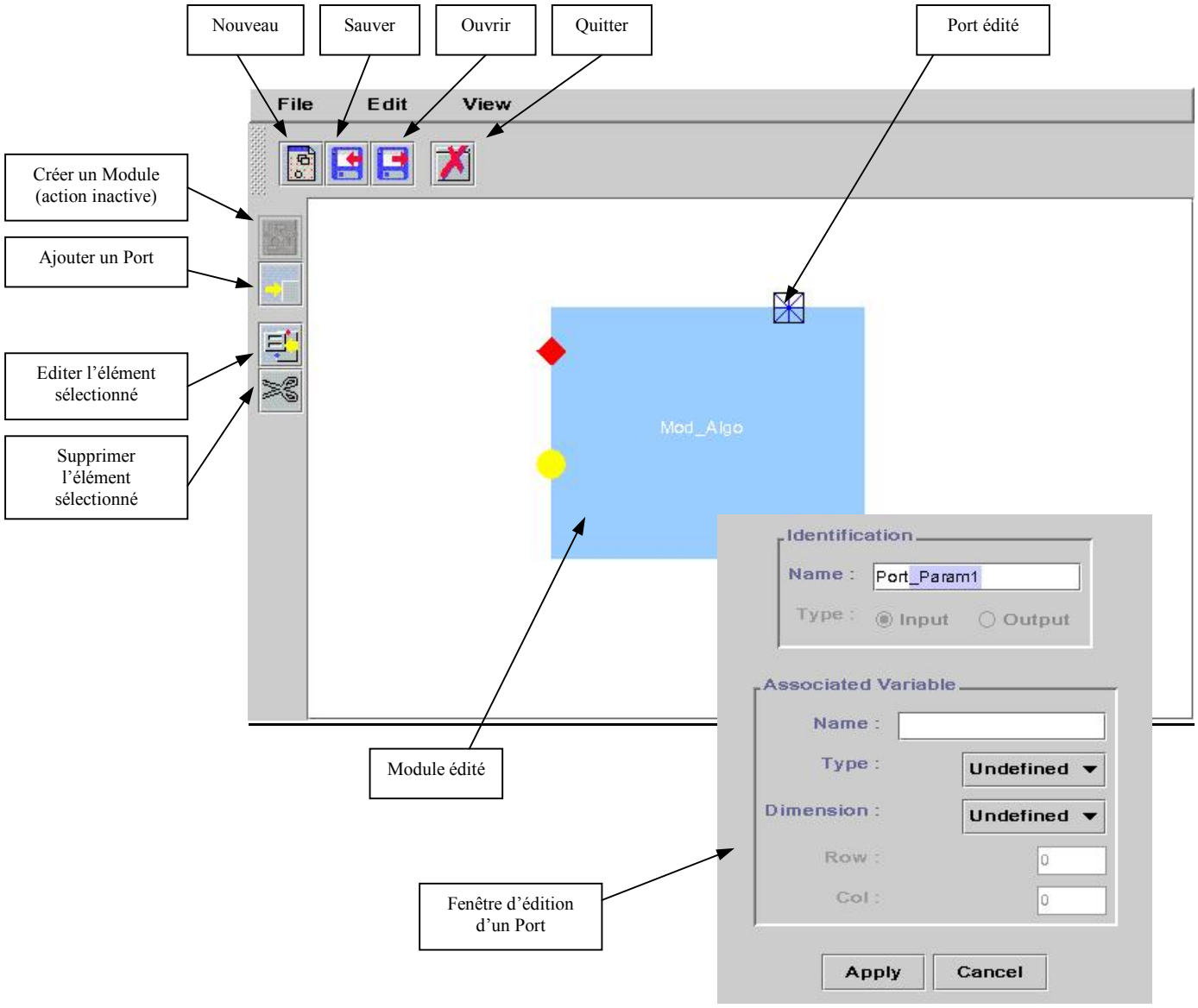

MENU FICHIER (JMenu)
OPTION NOUVEAU (JMenuItem)
OPTION OUVRIR (JMenuItem)
OPTION SAUVER (JMenuItem)
OPTION SAUVER SOUS (JMenuItem)
OPTION QUITTER (JMenuItem)
MENU EDITION (JMenu)
OPTION CREER MODULE (JMenu)
OPTION EFFACER (JMenuItem)
OPTION EDITER (JMenuItem)
OPTION AJOUTER PORT (JMenu)
OPTION CREER MODULE SENS (JMenuItem)
OPTION CREER MODULE ALGO (JMenuItem)
OPTION CREER MODULE PHY (JMenuItem)
OPTION AJOUTER PORT DATA (JMenuItem)
OPTION AJOUTER PORT PARAM (JMenuItem)
OPTION AJOUTER PORT DRIVER (JMenuItem)
OPTION AJOUTER PORT EVENT (JMenuItem)
MENU AFFICHAGE (JMenu)
OPTION AFFICHER BARRE D'OUTILS FICHIER (JMenuItem)
OPTION AFFICHER BARRE D'OUTILS EDITION (JMenuItem)
ARBRE DE WIDGETS : NIVEAU 2.1



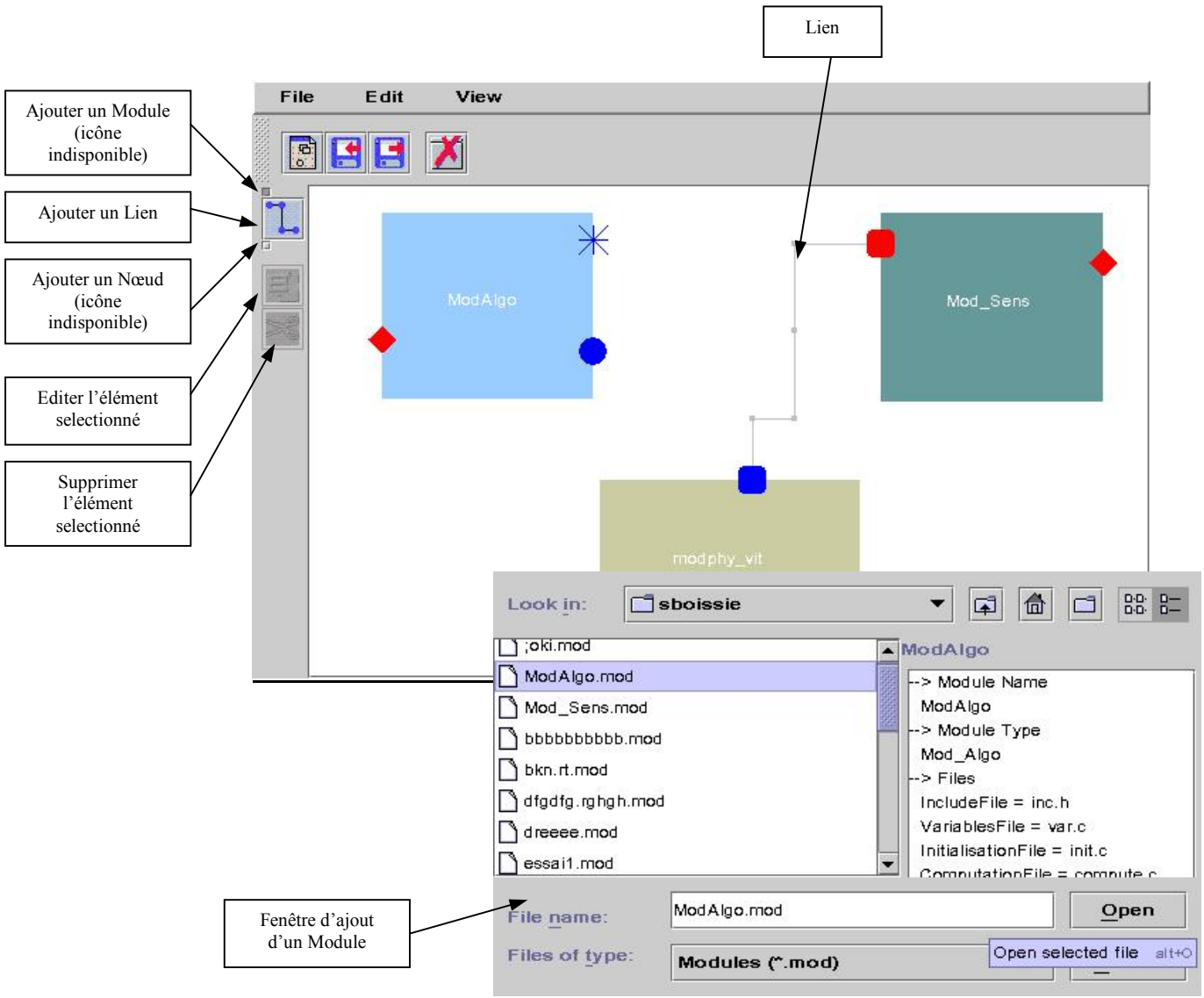

**ARBRE DE WIDGET : NIVEAU 2.2**

Résultat graphique :

*- éditeur de Module :*



- *éditeur de Tâche Robot :*



I-3 Ce qu'il reste à faire :

Le sujet qui nous a été proposé était, selon notre responsable, difficilement réalisable intégralement en 16 semaines. C'est pour cela que nous nous sommes restreints à la réalisation de la partie Spécification des entités gérées par le logiciel, cette dernière étant la plus importante.

Néanmoins, les parties Vérification et Simulation, demandent également à être remaniées car à ce jour, elles ne donnent pas entière satisfaction. Mais, pour ce faire, il faudrait mettre en place un cahier des charges détaillé avec l'aide des chercheurs. En effet, ces deux blocs d'ORCCAD font appel à des connaissances poussées en Automatique et Robotique.

La partie Exécution reste elle aussi à actualiser. Elle consiste à récupérer tout le code généré durant la phase de Spécification et de produire du code directement exécutable par les robots. Cette tâche ne nécessitera pas de lourd travail de conception et de réalisation puisque tous les éléments nécessaires sont présents dans la partie Spécification ; reste à regrouper ces éléments.

Pour la partie Spécification qui se trouve donc être l'objet même de notre stage, il nous reste à finaliser l'éditeur de Tâche Robot.

Plus précisément, il nous faut intégrer une structure de Contrainte Temporelle.
Cette structure est un élément propre aux Modules Algorithmiques. Elle correspond à la fréquence à laquelle un Module doit produire les données en sortie. Elle sera représentée comme un attribut d'un Module Algorithmique à définir au niveau de l'éditeur de Tâche Robot. Elle nécessitera la création de nouvelles fenêtres d'édition.

Il nous faut également finaliser les Liens entre les Modules. En effet, la vérification de la validité du réseau créé n'est pas encore tout à fait au point, il nous reste des tests à programmer sur les types de Ports qui sont liés et sur l'enchaînement des Modules.

Toujours dans la partie Spécification, il reste à intégrer la génération du code relatif à chaque entité qui sera inclus dans le code final.

D'autre part, il nous reste la partie Procédure Robot à réaliser. La structure de l'éditeur est faite, il suffit d'implémenter les différents traitements : la correspondance en langage Esterel de la Procédure construite, la sauvegarde et le chargement d'une Procédure, la génération du code Esterel final au moment de la sauvegarde…

Néanmoins, il semble que la plupart de ces fonctionnalités, étant très proches de celles déjà programmées au niveau des Modules et des Tâches Robots, soient réalisables dans les temps.

Dans les deux semaines de stage qu'il nous reste pour terminer notre projet, nous avons choisi de nous concentrer dans un premier temps sur l'éditeur de Tâche Robot. Nous verrons ensuite si le temps nous permet d'intégrer l'éditeur de Procédure Robot.

# IV Bilan :

## IV-1 La solution apportée :

Ces 16 semaines nous ont parus très courtes. Comme nous l'avons déjà précisé, le sujet proposé était difficilement réalisable dans le temps imparti.

Nous nous sommes appliqué à concevoir une application robuste et dont le code soit aussi explicite que possible afin que notre travail puisse être facilement réutilisable et étendu. Nous avons résolu les problèmes de compatibilité Linux-Solaris grâce au passage en Java. Nous avons bien sûr testé notre application sur les deux types d'environnement.
Dans cette optique et d'un point de vue plus technique, notre programme ne contient aucune interdépendance entre les différentes classes Java.

Le résultat est modulaire, il sera très facile de créer un nouveau type de Port ou de Module. Par contre, la modification de leur structure impliquera la perte des Modules et Ports déjà enregistrés, puisque la sauvegarde et le chargement se fait sur les objets. Cette méthode est très simple mais elle pourrait être amélioré pour faire face à ce genre de problèmes. La solution la plus adaptée serait l'utilisation de XML.

Nous espérons que notre travail portera ses fruits au sein de l'INRIA Rhône-Alpes, et qu'il satisfera ses futurs utilisateurs. Même si l'interface n'a pas été réalisée de façon exhaustive, une base solide a été mise en place.

## IV-2 Intérêt du stage :

Ce stage a été notre première longue expérience en entreprise et nous lui avons porté un grand intérêt.

Le sujet était très intéressant puisqu'il nous a permis d'avoir à notre charge un projet important : la réalisation de l'interface d'un logiciel dédié à la Robotique. Il nous a permis de mettre en application les connaissances que la MIAGe nous a apportées. Entre autre, nous avons pu profiter des notions d'ergonomie acquises en cours et TP d'Interaction Homme-Machine. Il nous a également permis d'appliquer les différentes méthodes de modélisation que nous avons vues dans le cadre de la conception de systèmes d'information. Bien sûr nous les avons adaptées au contexte du stage et aux méthodes de travail déjà mises en place au sein de l'INRIA Rhône-Alpes.

Par ailleurs, ce stage nous a permis de nous perfectionner pendant presque 4 mois dans le développement en Java que nous avions tous deux découvert cette année. Cela nous a permis de mettre en pratique nos connaissances sur les concepts et la manipulation de ce langage orienté objet.

Enfin, du point de vue de notre expérience professionnelle, ce stage nous a permis de découvrir ce qu'est un laboratoire de recherche et comment ce genre d'institut fonctionne. C'est une expérience que nous n'aurons peut-être pas l'occasion de renouveler dans le futur puisque notre formation va plus certainement nous amener à évoluer dans le monde de l'industrie. Notre curiosité nous a également amenés à découvrir tous les travaux réalisés au sein de cet institut et ainsi de découvrir les dernière avancées technologiques en matière d'informatique appliquée et d'automatisme.

IV-3 Bilan connaissance/compétence :

La principale compétence que nous avons acquise au cours de ce stage est donc la maîtrise du langage Java et par conséquent toute l'application des concepts.

Nous avons pu aussi découvrir à quel point il est difficile de mener un projet à terme, de la phase de conception à la phase de réalisation. Notre responsable a été présente tout au long de nos travaux et son aide a porté essentiellement sur les spécifications du projet et sur les détails des fonctionnalités à mettre en place. D'un point de vue organisationnel, méthodologique et technique, nous avions toutes les libertés de mener nos travaux comme nous le souhaitions.

C'est d'ailleurs à ce niveau que cela nous a présenté le plus de difficultés puisque nous n'avions pas ou peu d'expérience en matière de gestion et réalisation de projet. Cette expérience nous aura appris que pour bien gérer le temps, il faut être rigoureux au niveau de la programmation, mais surtout au niveau de la méthodologie. En effet, il est facile de se laisser porter par le projet et c'est pourquoi nous avons défini des objectifs intermédiaires qui nous ont permis d'avancer étapes par étapes.

Ce travail au sein de l'INRIA, institut connu et reconnu dans le monde de l'informatique, sera certainement très profitable à notre avenir professionnel.

# Bibliographie / Webliographie :


- **http://www.inrialpes.fr**
  Présentation de l'Institut National de Recherche en Informatique et Automatique. Description du service des Moyens Robotiques (http://www.inrialpes.fr/iramr/), notamment toutes les documentations concernant ORCCAD.

- **Manuel utilisateur – ORCCAD, version 3.0**
  *Ecrit par Roger Pissard-Gibolet et konstantinos Kapellos, INRIA.*

- **Formalisation et intégration en vision par ordinateur temps réel, thèse de sciences de l´ ingénieur ( 1999).**
*Ecrite par Soraya Arias.*

- **http://www.ilog.com**
  Site proposant tous les produits Ilog.

- **http://www.java.sun.com/docs/**
  Toutes les documentations sur les librairies Java.

- **http://koala.ilog.fr**
  Description de la librairie graphique koala.