# A.P.I. Kuka

# Manuel du devellopeur

# Version 0.1.0

Paul Chavent

Kubrick

12 février 2004

# Table des matières

# 1 Introduction

Ce projet à pour objectif la réalisation d'une interface de programation au robot Kuka d'IMERIR (KR6/KRC2).

Elle doit permettre de :

– commander le bras manipulateur grâce à un langage répandu.

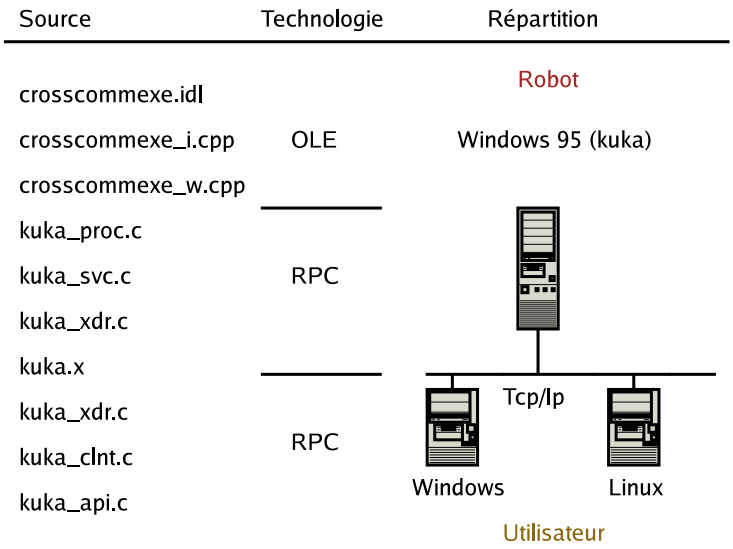
– d'exporter son interface sur le réseau de l'école.

Ce projet implémente ces services à l'aide des RPC et du langage de programmation c.

# 2 Maintenance de l'api

## 2.1 Décomposition du code du projet

La figure 1 représente les différents niveaux de l'implémentation de l'interface entre l'utilisateur et le robot.




Voici le détail de chaque niveau :

**crosscommexe.idl** est la spécification d'interface du crosscomm. Le crosscomm est un composant OLE fournis par Kuka.

**crosscommexe_i.c crosscommexe.h** sont **générées par midl** à partir du fichier crosscommexe.idl. Crosscommexe_i signifie crosscommexe_interface.

**crosscommexe_w.c et crosscommexe_w.h** doivent être **maintenus par les développeurs**. Il faudra modifier le moins possible leur interface. Ces sources servent :

- à garder la même interface d'utilisation pour le crosscommexe si celle-ci venait à être changée,
- à simplifier l'utilisation des fonctions du crosscommexe,
- à exporter en c l'interface du crosscommexe qui est en c++.

Crosscommexe_w signifie crosscommexe_warpper, c'est à dire traducteur.

**kuka_proc.c** doit être **maintenu par les développeurs**. Il implémente les fonctions qui seront exportées sur le réseaux en utilisant crosscommexe_w. Kkuka_proc signifie kuka_procédures.

**kuka.x** doit être **maintenus par les développeurs**. C'est une spécification d'interface en idl (Interface Définition Langage). **Attention** : une partie de ce fichier déclare les types et structures. Or ce sont les même que ceux exporté par l'api (définis dans kuka_api.h), ce qui implique de surveiller la concordance entre le fichier kuka.h (généré) et kuka_api.h (écrit).

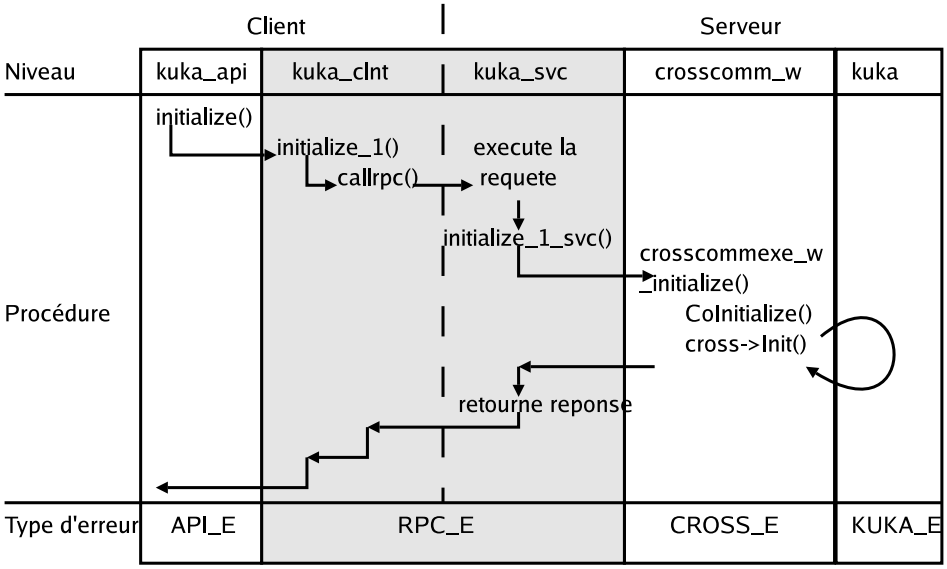
**kuka.h, kuka_svc.c, kuka_clnt.c et kuka_xdr.c** sont **générés par rpcgen** à partir du fichier kuka.x qui est la spécification d'interface utilisée pour communiquer sur le réseau.

**kuka_api.h et kuka_api.c** doivent être **maintenus par les développeurs**. Ils spécifient l'api qui permettra à l'utilisateur de dialoguer avec Kuka.

## 2.2 Gestion des erreurs

### 2.2.1 Généralité

Comme nous venons de le voir l'API appelle différentes couches. Par exemple lors de l'appel de la fonction `initialize` les appels aux couches inférieures permettent de faire remonter les erreurs (voir figure).




De cette manière nous pouvons faire remonter les erreurs.

### 2.2.2 Implémentation

Localement (c'est à dire entre l'api Kuka et le client rpc ou entre le crosscomm et le serveur rpc), la dernière erreur générée est enregistrée dans une variable statique nommée kukaError (de type `kukaVar_t` dont la valeur est de type `kukaError_t`). D'autre part, il est convenu que toutes les fonctions qui échouent retournent un `int` ou un `long` différent de zéro. Ainsi nous savons qu'il faut consulter la variable kukaError lorsque une fonction retourne -1. A distance ( c'est à dire entre le client rpc et le serveur rpc), il est convenu que toutes les procédures retournes une variable de type kukaVar_t . Si la valeur de cette variable est de type kukaError_t cela signifie qu'il y a eu une erreur dans un des appels de fonction sous-jacent.

# 3 Maintenace du programme de commande

Comme cela est expliqué dans l'introduction, un programme devra en permanence tourner sur Kuka. Ce programme peut être différent selon les besoins de l'utilisateur, et bien que l'api soit fournie avec un programme simple, les développeurs pourront le modifier pour le faire évoluer. Aussi, il pourra etre livré une interface qui encapsulera l'api Kuka afin de l'adapter à son contexte (voir l'exemple du manuel d'utilisateur).

Le schéma suivant met en évidence la répartition de chaque composant .