

# Mes premiers pas avec `librtl.so`

Christophe LOHR

18 août 2003

## Table des matières

<b>1</b>	<b>Généralités</b>	<b>3</b>
1.1	Pourquoi des types C++ et Java ?	3
1.2	Description du mécanisme d'utilisation des types	3
1.2.1	Version dynamique	3
1.2.2	Version statique	3
1.3	Technique d'implémentation	3
1.4	Conseil	4
<b>2</b>	<b>Les types en C++</b>	<b>4</b>
2.1	Principe	4
2.2	Déclaration des types dans RT-LOTOS	5
2.3	Génération de l'interface <code>RCCLib_lipsync.cc</code>	5
2.3.1	La marche à suivre :	5
2.4	Implémentation des types	6
<b>3</b>	<b>Compilation et création de <code>librtl.so</code></b>	<b>6</b>
<b>4</b>	<b>Les types en Java</b>	<b>7</b>
4.1	Principe	7
4.2	Déclaration des types dans RT-LOTOS	7
4.3	Génération de l'interface <code>RJLib.java</code>	7
4.3.1	La marche à suivre :	9
4.4	Implémentation des types	9
4.5	Compilation	9
4.5.1	Remarques :	10
<b>5</b>	<b>Choix entre la version statique et la version dynamique</b>	<b>10</b>
<b>A</b>	<code>lipsync.lot</code>	<b>12</b>
<b>B</b>	<code>RCCLib_lipsync.cc</code>	<b>13</b>
<b>C</b>	<code>tl_lib_buffer.h</code>	<b>14</b>
<b>D</b>	<code>tl_lib_buffer.cc</code>	<b>15</b>
<b>E</b>	<code>Makefile c++ dynamique</code>	<b>17</b>

<b>F</b>	<b>Makefile c++ statique</b>	<b>18</b>
<b>G</b>	<b>RJLib.java</b>	<b>19</b>
<b>H</b>	<b>RJLib_lipsync.java</b>	<b>19</b>
<b>I</b>	<b>jt_int.java</b>	<b>21</b>
<b>J</b>	<b>jt_pkt.java</b>	<b>21</b>
<b>K</b>	<b>jt_pktbuf.java</b>	<b>22</b>
<b>L</b>	<b>Makefile java dynamique</b>	<b>24</b>
<b>M</b>	<b>Makefile java statique</b>	<b>24</b>

# 1 Généralités

## 1.1 Pourquoi des types C++ et Java ?

Dans le langage LOTOS, ACT-ONE, formalisme de types de données abstraits algébriques, permet de définir la signature des types et leur sémantique.

Dans les spécifications RT-LOTOS, on utilise la syntaxe ACT-ONE pour définir la signature des types. Par-contre, leur sémantique est implémentée par des méthodes en C++ ou en Java, l'approche ACT-ONE sous la forme d'un ensemble d'équations étant, en milieu industriel, considérée comme étant beaucoup trop lourde et complexe.

L'outil `rtl` étant écrit en C++, la manière naturelle d'implémenter les types est d'utiliser C++.

Le développement en C++ étant parfois délicat (problèmes d'allocation de la mémoire à la main, «segmentation fault» surprenants,...), on a voulu pouvoir utiliser un langage plus simple tel que Java.

Le but de cette note est d'expliquer comment RT-LOTOS, lors de la simulation, 'communique' avec ces autres langages pour utiliser les types. <sup>1</sup>

## 1.2 Description du mécanisme d'utilisation des types

Vous avez la possibilité d'utiliser soit une *distribution dynamique* (i.e. avec des bibliothèques dynamiques), soit une *distribution statique* (i.e. le code des bibliothèques précédentes est compilé statiquement avec le code de `rtl`).

### 1.2.1 Version dynamique

Afin d'utiliser `rtl` indifféremment avec des types C++ ou Java on utilise la modularité qu'offre le système des bibliothèques dynamiques. Concrètement, cela signifie qu'en toutes circonstances, `rtl` fait appel à la bibliothèque `librtl.so` pour manipuler les types.

Lorsque l'on implémente des types en C++ on construit une nouvelle bibliothèque `librtl.so` qui contient le code C++ (cf Fig.1). Et lorsque l'on implémente des types en Java on utilise une bibliothèque également nommée `librtl.so` mais qui est déjà existante et qui effectue la communication entre `rtl` et le code Java (cf Fig.3 et Fig.4).

### 1.2.2 Version statique

Lorsque l'on implémente des types en C++ on construit une nouvelle bibliothèque `librtl.so` qui contient le code C++ (cf Fig.1) que l'on utilisera avec le programme `rtl`. Et lorsque l'on implémente des types en Java on utilisera le programme `rtl-java-shmem` ou `rtl-java-socket` (cf Fig.5 et Fig.6).

## 1.3 Technique d'implémentation

La séquence des opérations à réaliser est identique dans leur principe pour Java et C++

---

<sup>1</sup>N'oubliez pas de lire attentivement le "RTL User Guide".

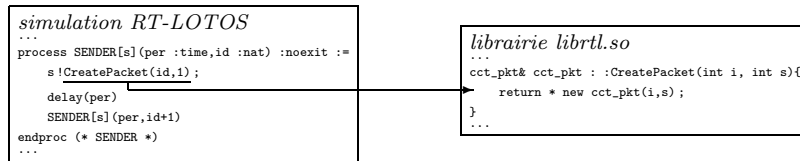


FIG. 1 – Architecture du simulateur utilisant des types C++, version dynamique

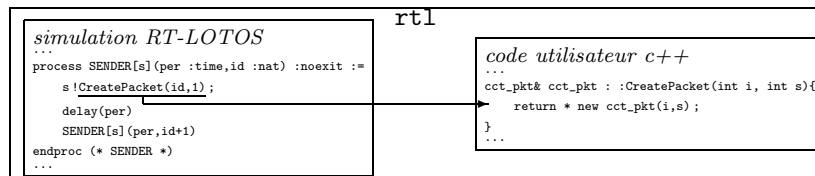


FIG. 2 – Architecture du simulateur utilisant des types C++, version statique

1. déclaration de la signature des types en ACT-ONE,
2. génération automatique d'un handler de fonctions utilisateur, on utilise la commande `rtl -make-lib specification.lot` qui génère soit un fichier `RCC_lib_specification.cc` soit les fichiers `RJLib.java` et `RJLib_specification.java`,
3. écriture du code implémentant les types de l'utilisateur,
4. compilation de ce handler avec le code des types.

## 1.4 Conseil

Je conseille de prévoir un répertoire pour chaque projet RT-LOTOS et d'y créer un répertoire `lib/` dans lequel on placera la bibliothèque `librtl.so` générée pour ce projet. Pensez à rajouter `lib/` dans votre variable d'environnement `LD_LIBRARY_PATH`. Vérifiez qu'il n'y a pas d'ambiguïté avec d'autres `librtl.so`. Lorsque l'on aura réalisé une première bibliothèque, on pourra utiliser la commande `ldd rtl` pour vérifier que l'éditeur de lien dynamique va chercher la bonne `librtl.so`.

## 2 Les types en C++

### 2.1 Principe

- Avec la version dynamique, lorsque l'on implémente des types en C++ on construit une nouvelle bibliothèque `librtl.so` qui contient le code C++ des types de l'utilisateur (cf Fig.1).
- Avec la version statique, pour implémenter des types en C++, il va falloir recompiler un nouveau `rtl` (cf Fig.2) incorporant les objets des types de l'utilisateur qui autrement se seraient trouvés dans `librtl.so`. L'archive `./lib/libobjrtl.a` contient tous les objets nécessaires à la recompilation de `rtl`

## 2.2 Déclaration des types dans RT-LOTOS

La première étape consiste à écrire une spécification RT-LOTOS avec la signature des types en ACT-ONE. Cette spécification peut très bien être votre projet lui même ou n'importe quel autre du moment qu'elle contienne les bonnes déclarations ACT-ONE des types que vous voulez utiliser.

Regardez le fichier `lipsync.lot` donné en exemple en annexe A.

Notez que les types `nat`, `bool` et `int` sont déjà implémentés dans `rtl`. Cependant, on a enrichi `int` de la fonction `mult`.

## 2.3 Génération de l'interface `RCCLib_lipsync.cc`

À partir du fichier `lipsync.lot` que l'on vient d'écrire, on va générer un fichier C++ portant un nom de la forme `RCCLib_nom-du-fichier-lotos.cc`, c'est à dire ici : `RCCLib_lipsync.cc`.

Le code de cette classe C++ réalise l'interface entre `rtl` et les types implémentés en C++. À chaque nouvelle opération sur les types est associé un numéro d'appel que va utiliser `rtl` pour lancer l'exécution de telle ou telle fonction lors de la simulation. Le code de cette classe récupère donc ce numéro et lance l'exécution de la fonction correspondante, fonction que vous aurez définie.

### 2.3.1 La marche à suivre :

- Si vous utilisez la version dynamique :
  1. Récupérez et copiez `librtl.so.cc` dans le `lib/` correspondant à votre projet sous le nom `librtl.so`. (Et bien oui!, il faut bien un `librtl.so` pour commencer. Celui-ci contient tout ce qu'il faut pour générer le `RCCLib_lipsync.cc` en question).
  2. Exécutez `rtl -make-lib lipsync.lot`. Notez aussi que cette opération peut s'achever sur un *bus error* ; normalement ce bug est corrigé, mais même s'il se produit, c'est sans conséquence car le fichier a été correctement généré.
- Si vous utilisez la version statique :

Exécutez `rtl-cc -make-lib lipsync.lot`. Notez que cette opération peut aussi s'achever sur un *bus error*.

Vous remarquerez que la première ligne du fichier `RCCLib_lipsync.cc`<sup>2</sup> ainsi créé est : `#include "tl_lib_lipsync.h"`. En effet, `rtl` suppose que vos types seront implémentés par les fichiers `tl_lib_nom-du-fichier-lotos.h` et `tl_lib_nom-du-fichier-lotos.cc`. Bien entendu vous pourrez choisir le nom qui vous plait et changer le `#include` en question dans `RCCLib_lipsync.cc`.

Par exemple ici, on prendra `tl_lib_buffer.h`. Notre Makefile comportera donc des commandes du genre :

```
sed -e "s/tl_lib_lipsync.h/tl_lib_buffer.h/g" RCCLib_lipsync.cc > tmp.cc
mv tmp.cc RCCLib_lipsync.cc
```

Un détail peut aussi vous troubler : en effet, si le nom de votre fichier RT-LOTOS est trop long, le nom du fichier `RCCLib_nom-du-fichier-lotos.cc` sera tronqué à 200 caractères (ce qui laisse tout de même de la marge...) lors de sa génération. Si c'est le cas, redonnez-lui son nom correct.

<sup>2</sup>voir annexe B

## 2.4 Implémentation des types

Il va maintenant falloir écrire les fichiers `tl_lib_buffer.h` et `tl_lib_buffer.cc`. Ils sont donnés en annexe C.

À chaque type correspond une classe. Le nom de la classe est de la forme `cct_type` (par exemple ici `cct_pkt`).

### Important

Chaque type doit comporter :

- un constructeur
- un destructeur
- un opérateur de comparaison `==`
- une méthode pour afficher l'objet `print()`
- une méthode pour copier l'objet `fait_copie()`
- une méthode qui retourne un code de hachage `fonction_hashing()`

Vous devez gérer vous-même l'allocation de la mémoire. Cependant, vous avez à votre disposition la classe `user_obj` dans `tl_user.h` qui propose deux méthodes :

- `add_ref()` qui incrémente le nombre de références à l'objet appelé
- `del_ref()` qui décrémente le nombre de références à l'objet appelé, et le détruit si ce nombre devient nul.

**Attention :** il est impératif que vos méthodes ne modifient en aucun cas les objets passés en paramètre. Vous vous exposeriez alors à de bien mauvaises surprises...

Par ailleurs, les méthodes correspondant aux opérations sur les types que vous avez définis doivent être `static`.

## 3 Compilation et création de `librtl.so`

Il ne reste plus qu'à :

- 1 compiler le fichier contenant les classes des types, ici `tl_lib_buffer.cc`
- 2 compiler l'interface `RCCLib_lipsync.cc`
- si vous utilisez la version dynamique :
  - 3 linker `tl_lib_buffer.o` et `RCCLib_lipsync.o` pour créer `librtl.so`
  - 4 stripper `librtl.so` si vous êtes perfectionniste
  - 5 déplacer `librtl.so` dans `lib/`
- si vous utilisez la version statique :
  - 3 linker `tl_lib_buffer.o` et `RCCLib_lipsync.o` avec `lib/libobjrtl.a` pour créer un nouveau `rtl`
  - 4 stripper `rtl` si vous êtes perfectionniste

Vous pouvez consulter le `Makefile` donné en exemple en annexe E.

Vous pouvez consulter le `Makefile` donné en exemple en annexe F.

Voilà, c'est tout ! Vous pouvez jouer maintenant avec vos nouveaux types.

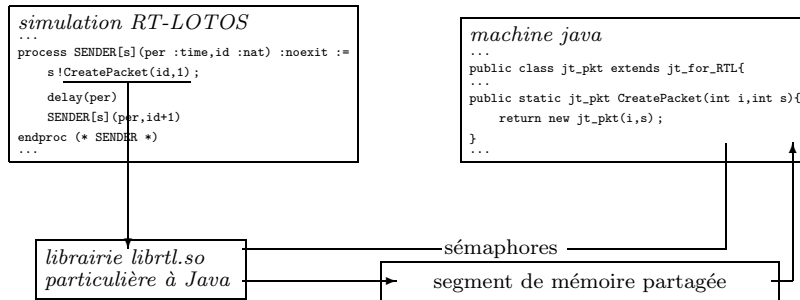


FIG. 3 – Architecture du simulateur utilisant des types Java et communiquant par mémoire partagée, version dynamique

## 4 Les types en Java

### 4.1 Principe

Contrairement à l'utilisation des types en C++, l'implémentation des types Java ne peut pas se faire directement par la bibliothèque `librt1.so` car le code Java ne permet pas de créer de bibliothèques dynamiques...

Le code des types est exécuté par l'interpréteur Java et communique avec `rtl` par mémoire partagée et sémaphores ou bien par socket unix. Les deux techniques sont disponibles.

- Si l'on opte pour une communication par mémoire partagée, ce segment de mémoire partagée est gérée du côté Java par `RJInterfaceShMem.class` et du côté `rtl` par une autre bibliothèque `librt1.so`. Cette bibliothèque est déjà disponible, l'utilisateur n'a pas à la recompiler. Cette bibliothèque sert d'intermédiaire entre le simulateur et la machine Java. Dans la version statique cette bibliothèque est intégrée au code de `rtl` qui devient `rtl-java-shmem`.
- Si l'on opte pour une communication par socket, le socket est gérée du côté Java par `RJInterfaceSocket.class` et du côté `rtl` par une autre version de la bibliothèque `librt1.so`. Dans la version statique cette bibliothèque est intégrée au code de `rtl` qui devient `rtl-java-socket`.

Le choix entre communication par mémoire partagée ou par socket est indifférent pour le reste des opérations. Par ailleurs, si la communication par socket unix est en principe un peu plus rapide, on n'observera pas de différence notable ici, la majeure partie du temps étant utilisée par la machine Java.

### 4.2 Déclaration des types dans RT-LOTOS

Comme pour les types en C++, il faut tout d'abord donner une spécification RT-LOTOS des types <sup>3</sup>.

### 4.3 Génération de l'interface `RJLib.java`

À partir de la spécification RT-LOTOS que l'on vient d'écrire, on va générer deux fichiers JAVA : `RJLib.java` et `RJLib_nom-de-fichier-lotos.java` (ici `RJLib_lipsync.java`)

<sup>3</sup>voir le paragraphe 2.2

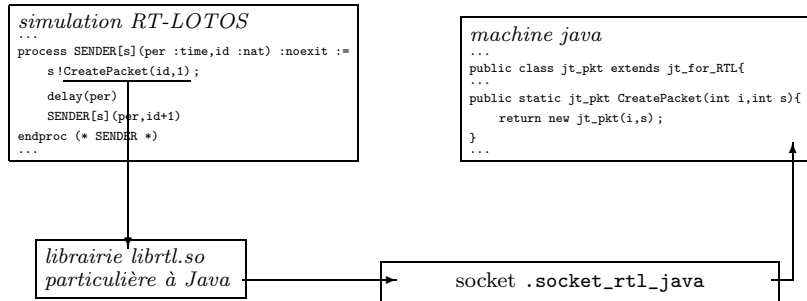


FIG. 4 – Architecture du simulateur utilisant des types Java et communiquant par socket unix, version dynamique

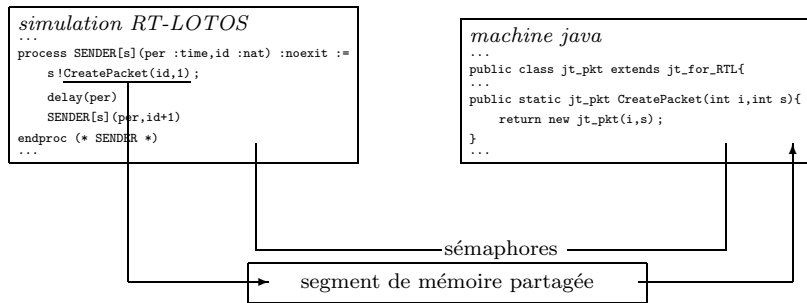


FIG. 5 – Architecture du simulateur utilisant des types Java et communiquant par mémoire partagée, version statique

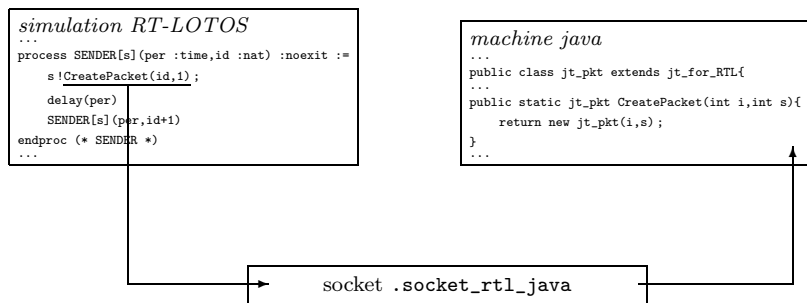


FIG. 6 – Architecture du simulateur utilisant des types Java et communiquant par socket unix, version statique

Lorsque le gestionnaire de mémoire partagée du côté JAVA note que `rtl` fait un appel à une opération sur des types, le gestionnaire sollicite la classe `RJLib`. Cette classe consiste uniquement en un appel à la classe `RJLib_lipsync`. Cette classe est l'interface qui associe à chaque numéro d'appel la bonne opération sur les types, comme en C++<sup>4</sup>.

#### 4.3.1 La marche à suivre :

- Si vous utilisez la version dynamique :
  1. Si vous optez pour la communication par mémoire partagée, récupérez et copiez `librtl.so.java-shmem` dans le `lib/` correspondant a votre projet sous le nom `librtl.so`.
  2. Si vous optez pour la communication par socket, récupérez et copiez `librtl.so.java-socket` dans le `lib/` correspondant a votre projet sous le nom `librtl.so`.
  3. Exécutez `rtl -make-lib lipsync.lot`.
- Si vous utilisez la version statique :
  1. Si vous optez pour la communication par mémoire partagée, exécutez `rtl-java-shmem -make-lib lipsync.lot`.
  2. Si vous optez pour la communication par socket, exécutez `rtl-java-socket -make-lib lipsync.lot`.

Des exemples sont donnés en annexe G.

## 4.4 Implémentation des types

Vous allez devoir créer une classe par type défini. Le nom des classes doit être de la forme `jt_nom-de-type` (et pour les fichiers : `jt_nom-de-type.java`). Ici on aura donc `jt_int.java`, `jt_pkt.java` et `jt_pktbuf.java`.

Vous devez suivre les mêmes consignes qu'en C++<sup>5</sup> pour implémenter vos types à ceci près que la gestion de la mémoire est grandement simplifiée en JAVA!

Les classes correspondant aux types doivent hériter de la classe `jt_for_RTL`. Vous pouvez regarder les exemples donnés en annexe I.

## 4.5 Compilation

Le fichier à compiler est `RJLib.java`. En fait, l'exécutable JAVA est `RJInterfaceShMem.class` (pour une communication par socket c'est `RJInterfaceSocket.class`), mais il passe par l'intermédiaire de `RJLib.java`. Ainsi il a pu être compilé une bonne fois pour toutes. Soyez prudent si vous entreprenez de recompiler `RJInterfaceShMem.java` car cette classe utilise des fonctions natives en C pour gérer la communication.

Un exemple de `Makefile` pour distribution dynamique est donné en annexe L, et un autre pour distribution statique en annexe M.

Notez que vous allez avoir besoin des classes correspondant au gestionnaire de mémoire partagée. Il faut : `RJDecoder.class`,

---

<sup>4</sup>voir le paragraphe 2.3

<sup>5</sup>voir paragraphe 2.4

`RJInterfaceShMem.class`, `RJPacket.class`, `RJQueue.class`,  
`jt_for_RTL.class`, `t_special_for_RTL.class`. Placez-les toutes dans  
un répertoire sur lequel pointe votre variable d'environnement `CLASSPATH`.

#### 4.5.1 Remarques :

- Les sémaphores et le segment mémoire sont choisis automatiquement par `rtl` (on peut les lui imposer par les options `-semkey-N` et `-shmkey-N`). Son choix porte sur les premières clefs libres. Généralement, il s'agit de la clef de sémaphore 1 et de la clef de segment partagé 1. Les trois sémaphores nécessaires sont associées à la même clef. Vous pourrez vous en convaincre par un `ipcs -sa`.
- Sur certains systèmes d'exploitation tels que Solaris il n'est pas possible d'avoir plus de 10 clefs (ce qui n'est pas le cas d'autres systèmes tels que Linux). Vous ne pourrez pas par conséquent effectuer plus de 10 simulations simultanément. Ceci dit, il n'est pas rare qu'une seule simulation nécessite 30Mo pour s'exécuter. Le nombre de clefs ne devrait donc pas être le facteur limitant....
- Si votre simulation s'achève sur une erreur quelconque, il se peut que les sémaphores que vous avez utilisés restent actifs dans le noyau du système d'exploitation. Ceci peut poser des problèmes si vous relancez immédiatement une autre simulation avec des types JAVA sur ces mêmes sémaphores. Normalement `rtl` devrait alors choisir d'autres sémaphores, mais on n'est jamais trop prudent ! Pensez donc à vérifier de temps en temps ce qu'il en est avec la commande `ipcs`. Vous pouvez détruire ces sémaphores avec la commande `ipcrm -S 1 -M 1` (s'il s'agit de ces numéros de sémaphores, comme l'indique `ipcs`).
- Dans le cas d'une communication par socket, le socket porte le nom de `./socket_rtl_java`. C'est un fichier spécial unix (un 'pipe'), créé dans le répertoire de travail au début de la simulation et détruit à la fin.

## 5 Choix entre la version statique et la version dynamique

Que vous utilisiez `rtl` compilé en *dynamic* ou en *static*, cela ne change en rien les principes énoncés plus haut concernant l'écriture de types utilisateurs ; seule la manipulation de l'outil `rtl` est modifiée.

Dans la version dynamique `rtl` fait appel à la librairie dynamique `librtl.so`. Il y a quatre versions de cette librairie `librtl.so`, `librtl.so.cc`, `librtl.so.java-shmem` et `librtl.so.java-socket`. On place alors l'une ou l'autre de ses librairies dans son `LD_LIBRARY_PATH` suivant ses besoins.

Dans la version statique, `rtl` a été compilé quatre fois intégrant les différents versions de `librtl.so`. Dans le répertoire `./bin` se trouvent alors quatre versions de `rtl` :

- `rtl` : équivalent à `rtl + librtl.so`  
pour utiliser des spécifications sans types utilisateur
- `rtl-cc` : équivalent à `rtl + librtl.so.cc`  
pour générer l'interface C++ des types utilisateurs

- `rtl-java-shmem` : équivalent à `rtl + librtl.so.java-shmem`  
pour générer l'interface Java des types utilisateurs et simuler des spécification en utilisant la mémoire partagée
- `rtl-java-socket` : équivalent à `rtl + librtl.so.java-socket`  
pour générer l'interface Java des types utilisateurs et simuler des spécification en utilisant des sockets

Pour implémenter des types en C++, on re-créeait une nouvelle librairie `librtl.so`. Avec la version static, il va falloir recompiler un nouveau `rtl` incorporant les objets des types de l'utilisateur qui autrement se seraient trouvés dans `librtl.so`. L'archive `./lib/libobjrtl.a` contient tous les objets nécessaires à la recompilation de `rtl` ; il suffit de linker `./lib/libobjrtl.a` avec son propre `RCCLib_xxx.o`.

## A lipsync.lot

specification LipSynchro : noexit

```
type boolean is
  sorts bool
  opns
    not :nat->nat
    and :nat,nat->nat
    or :nat,nat->nat
endtype

type natural is boolean
  sorts nat
  opns
    + :nat,nat->nat
    - :nat,nat->nat
    * :nat,nat->nat
    min :nat,nat->nat
    max :nat,nat->nat
    < :nat,nat->nat
    > :nat,nat->nat
    <= :nat,nat->nat
    >= :nat,nat->nat
    div :nat,nat->nat
    mod :nat,nat->nat
    divs :nat,nat->nat
endtype

type integer is boolean, natural
  sorts int
  opns
    + :int,int->int
    - :int,int->int
    * :int,int->int
    min :int,int->int
    max :int,int->int
    < :int,int->int
    > :int,int->int
    <= :int,int->int
    >= :int,int->int
    divs :int,int->int
    div :int,int->int
    mod :int,int->int
    mult :int,int,int->int
endtype

type packet is natural
  sorts pkt
  opns
    CreatePacket : nat,nat->pkt
    PacketId : pkt->nat
endtype

type buffer is natural,boolean,packet
  sorts pktbuf
  opns
    CreateBuffer : nat->pktbuf
    AddPacket : pkt,pktbuf->pktbuf
    PlacePacket : pkt,pktbuf->pktbuf
    RemovePacket : pktbuf->pktbuf
    FirstPacket : pktbuf->pkt
    BufferEmpty : pktbuf->bool
    BufferFull : pktbuf->bool
    BufferLoad : pktbuf->nat
endtype

behaviour
  exit
endspec (* LipSynchro *)
```

## B RCCLib\_lipsync.cc

```
// user-defined types interface made by rtl -make-lib

#include "tl_lib_lipsync.h"

#define id_mult 200
#define id_CreatePacket 201
#define id_PacketId 202
#define id_CreateBuffer 203
#define id_AddPacket 204
#define id_PlacePacket 205
#define id_RemovePacket 206
#define id_FirstPacket 207
#define id_BufferEmpty 208
#define id_BufferFull 209
#define id_BufferLoad 210

void user_obj::user_init () {
}

void user_obj::user_destroy(){
}

int user::prend_n_fonction_user ( char* str_fonction,
i_liste_sort_char& i_sort) {
    if ( strcmp (str_fonction,"mult") == 0 )
        return id_mult;
    else if ( strcmp (str_fonction,"CreatePacket") == 0 )
        return id_CreatePacket;
    else if ( strcmp (str_fonction,"PacketId") == 0 )
        return id_PacketId;
    else if ( strcmp (str_fonction,"CreateBuffer") == 0 )
        return id_CreateBuffer;
    else if ( strcmp (str_fonction,"AddPacket") == 0 )
        return id_AddPacket;
    else if ( strcmp (str_fonction,"PlacePacket") == 0 )
        return id_PlacePacket;
    else if ( strcmp (str_fonction,"RemovePacket") == 0 )
        return id_RemovePacket;
    else if ( strcmp (str_fonction,"FirstPacket") == 0 )
        return id_FirstPacket;
    else if ( strcmp (str_fonction,"BufferEmpty") == 0 )
        return id_BufferEmpty;
    else if ( strcmp (str_fonction,"BufferFull") == 0 )
        return id_BufferFull;
    else if ( strcmp (str_fonction,"BufferLoad") == 0 )
        return id_BufferLoad;
    else return 0;
}

user_obj* user::execute_fonction_user (int n, i_liste_sort_char& i_sort,
i_liste_sem& i_param) {
    switch (n) {
        case id_mult: {
            int par1 = ((int_lib *)i_param.elem_1_de_1())->prend_int();
            ++i_param;
            int par2 = ((int_lib *)i_param.elem_1_de_1())->prend_int();
            ++i_param;
            int par3 = ((int_lib *)i_param.elem_1_de_1())->prend_int();
            return new int_lib(cct_int::mult(par1,par2,par3));
        }
        case id_CreatePacket: {
            int par1 = ((int_lib *)i_param.elem_1_de_1())->prend_int();
            ++i_param;
            int par2 = ((int_lib *)i_param.elem_1_de_1())->prend_int();
            return &(cct_pkt::CreatePacket(par1,par2));
        }
        case id_PacketId: {
            cct_pkt *par1 = (cct_pkt*)i_param.elem_1_de_1();
            return new int_lib(cct_pkt::PacketId(*par1));
        }
        case id_CreateBuffer: {
            int par1 = ((int_lib *)i_param.elem_1_de_1())->prend_int();
    
```



```

typedef cct_pkt* p_Packet;

class cct_pktbuf:public user_obj{
    DList<p_Packet> ListPacket;
    int size;
public:
    cct_pktbuf(int s);
    virtual ~cct_pktbuf();
    virtual char operator == (objet& un_objet);
    virtual ostream& print(ostream& out) const;
    virtual t_offptr fonction_hashing();
    virtual user_obj& fait_copie();
    void AddPacket(cct_pkt&);
    void PlacePacket(cct_pkt&);
    void RemovePacket();
    cct_pkt& FirstPacket(){ return *ListPacket.front(); }
    char BufferEmpty(){ return ListPacket.empty(); }
    int BufferLoad(){ return ListPacket.length(); }
    char BufferFull(){ return ListPacket.length() == size; }

    static cct_pktbuf& CreateBuffer(int);
    static cct_pktbuf& AddPacket(cct_pkt&, cct_pktbuf&);
    static cct_pktbuf& PlacePacket(cct_pkt&, cct_pktbuf&);
    static cct_pktbuf& RemovePacket(cct_pktbuf&);
    static cct_pkt& FirstPacket(cct_pktbuf&);
    static char BufferEmpty(cct_pktbuf&);
    static char BufferFull(cct_pktbuf&);
    static int BufferLoad(cct_pktbuf&);
};

class cct_int:public user_obj{
public:
    cct_int(){ };
    static int mult(int n1,int d1,int d2){
        return (int)((float)n1*(float)d1/(float)d2);
    };
};

```

## D t1\_lib\_buffer.cc

```

#include <strstream.h>
#include <string.h>

#include "t1_lib_buffer.h"

//+++++ Class cct_pkt +++++
cct_pkt::cct_pkt(int i,int s){
    id=i;
    size=s;
}

cct_pkt::~cct_pkt(){
}

char cct_pkt::operator == (objet &un_objet){
    cct_pkt &a_packet = (cct_pkt &) un_objet;
    if (this == &a_packet)
        return 1;
    return (size == a_packet.size) && (id == a_packet.id);
}

ostream& cct_pkt::print(ostream& out) const{
    out << id;
    return out;
}

t_offptr cct_pkt::fonction_hashing(){
    return 0;
}

```

```

user_obj& cct_pkt::fait_copie(){
    return *(user_obj *) new cct_pkt(id,size);
}

cct_pkt& cct_pkt::CreatePacket(int i, int s){
    return * new cct_pkt(i,s);
}

int cct_pkt::PacketId(cct_pkt &P){
    return P.id;
}

int cct_pkt::PacketSize(cct_pkt &P){
    return P.size;
}

//+++++++ Class cct_pktbuf ++++++++

cct_pktbuf::cct_pktbuf(int s):ListPacket(){
    size = s;
}

cct_pktbuf::~cct_pktbuf(){
    if (ListPacket.length())
        for (Pix i=ListPacket.first();i != 0;ListPacket.next(i))
            ListPacket(i)->delete_ref();
}

char cct_pktbuf::operator == (objet& un_objet){
    cct_pktbuf& a_cct_pktbuf = (cct_pktbuf&) un_objet;
    if ( this == &a_cct_pktbuf )
        return 1;
    Pix i=ListPacket.first();
    Pix j=a_cct_pktbuf.ListPacket.first();
    int res= (ListPacket.length() == a_cct_pktbuf.ListPacket.length());
    for (;i != 0; ListPacket.next(i),a_cct_pktbuf.ListPacket.next(j))
        res = res && (*ListPacket(i) == *a_cct_pktbuf.ListPacket(j));
    return res;
}

ostream& cct_pktbuf::print(ostream& out) const{
    DLList<p_Packet> *p_list = (DLList<p_Packet> *)&ListPacket;
    out << "[";
    for (Pix i=p_list->first();i != 0;p_list->next(i))
        out<<*(*p_list)(i)<<": ";
    out <<"]";
    return out;
}

user_obj& cct_pktbuf::fait_copie() {
    cct_pktbuf* p_copie_cct_pktbuf = new cct_pktbuf(size);
    for (Pix i=ListPacket.first();i != 0;ListPacket.next(i)) {
        cct_pkt* a_Packet = ListPacket(i);
        a_Packet->add_ref ();
        p_copie_cct_pktbuf->ListPacket.append(a_Packet);
    }
    return (user_obj&) *p_copie_cct_pktbuf;
}

t_offptr cct_pktbuf::fonction_hashing() {
    return 0;
}

void cct_pktbuf::PlacePacket (cct_pkt& pkt) {
    pkt.add_ref();
    Pix i;
    for ( i = ListPacket.first();i !=0 ;ListPacket.next(i)){
        cct_pkt* a_Packet = ListPacket(i);
        if (pkt.id < a_Packet->id){

```

```

        ListPacket.ins_before(i,&pkt);
        break;
    }
}
if (i==0)
    ListPacket.append(&pkt);
}

void cct_pktbuf::AddPacket (cct_pkt& pkt) {
    pkt.add_ref();
    ListPacket.append (&pkt);
}

void cct_pktbuf::RemovePacket () {
    cct_pkt& a_Packet = FirstPacket ();
    a_Packet.delete_ref ();
    ListPacket.del_front ();
}

cct_pktbuf& cct_pktbuf::CreateBuffer(int s){
    return * new cct_pktbuf(s);
}

cct_pktbuf& cct_pktbuf::AddPacket(cct_pkt& P, cct_pktbuf& B){
    cct_pktbuf &copy = (cct_pktbuf &)B.fait_copie();
    copy.AddPacket(P);
    return copy;
}

cct_pktbuf& cct_pktbuf::PlacePacket(cct_pkt& P, cct_pktbuf& B){
    cct_pktbuf &copy = (cct_pktbuf &)B.fait_copie();
    copy.PlacePacket(P);
    return copy;
}

cct_pktbuf& cct_pktbuf::RemovePacket(cct_pktbuf& B){
    cct_pktbuf &copy = (cct_pktbuf &)B.fait_copie();
    copy.RemovePacket();
    return copy;
}

cct_pkt& cct_pktbuf::FirstPacket(cct_pktbuf& B){
    B.FirstPacket().add_ref();
    return B.FirstPacket();
}

char cct_pktbuf::BufferEmpty(cct_pktbuf& B){
    return B.BufferEmpty();
}

char cct_pktbuf::BufferFull(cct_pktbuf& B){
    return B.BufferFull();
}

int cct_pktbuf::BufferLoad(cct_pktbuf& B){
    return B.BufferLoad();
}

```

## E Makefile c++ dynamique

```

# ce qu'il faut configurer
NAME=lypsync
TYPELIB=t1_lib_buffer

# ce qu'il vaut mieux regarder
RTLHOME=${HOME}/RT-LOTOS
RTL=${RTLHOME}/bin/rtl -cray
RTLIB=${RTLHOME}/bin/rtl -make-lib
INCLUDES= -I ${RTLHOME}/include
LIBPATH=lib

```

```

#####
RLIB=RCCLib_$(NAME).cc
CC=gcc
SPEC=$(NAME).lot
SIM=$(SPEC).sim

$(SIM) : $(SPEC)
    @hostname
    @date
    -time $(RTL) -max-spec-t-200000000 $(SPEC)
    @date
    @echo "rtl ($(HOST):'pwd'): $(SIM) is done." | mail $(LOGNAME)

clean :
    -rm *~ *.fig *.sim
    (cd Plot ; make clean)

cclib : librtl.so
    mv librtl.so $(LIBPATH)/librtl.so

librtl.so : RCCLib_$(NAME).o $(TYPELIB).o
    ld -G -o librtl.so RCCLib_$(NAME).o $(TYPELIB).o

RCCLib_$(NAME).o : $(TYPELIB).h $(RLIB)
    gcc -c $(RLIB) $(INCLUDES)

$(TYPELIB).o : $(TYPELIB).cc $(TYPELIB).h
    gcc -c $(TYPELIB).cc $(INCLUDES)

$(RLIB) : $(SPEC)
    cp $(LIBPATH)/librtl.so.cc $(LIBPATH)/librtl.so
    -$(RTL) $(SPEC)
    sed -e "s/tl_lib_$(NAME).h/$(TYPELIB).h/g" $(RLIB) > tmp.cc
    mv tmp.cc $(RLIB)

libclean :
    -rm *~ *.o librtl.so $(RLIB)

```

## F Makefile c++ statique

```

# ce qu'il faut configurer
NAME=lypsync
TYPELIB=tl_lib_buffer

# ce qu'il vaut mieux regarder
RTLHOME=${HOME}/RT-LOTOS
RTL=./rtl -cray
RTLIB=${RTLHOME}/bin/rtl-cc -make-lib
INCLUDES= -I ${RTLHOME}/include

#####
RLIB=RCCLib_$(NAME).cc
CC=gcc
SPEC=$(NAME).lot
SIM=$(SPEC).sim

$(SIM) : $(SPEC) $(RTL)
    @hostname
    @date
    -time $(RTL) -max-spec-t-200000000 $(SPEC)
    @date
    @echo "rtl ($(HOST):'pwd'): $(SIM) is done." | mail $(LOGNAME)

clean :
    -rm *~ *.fig *.sim
    (cd Plot ; make clean)

```

```

cclib : $(RTL)
$(RTL) : RCCLib_$(NAME).o $(TYPELIB).o
        ld -o rtl RCCLib_$(NAME).o $(TYPELIB).o ${RTLHOME}/lib/libobjrt.o

RCCLib_$(NAME).o : $(TYPELIB).h $(RLIB)
        gcc -c $(RLIB) $(INCLUDES)

$(TYPELIB).o : $(TYPELIB).cc $(TYPELIB).h
        gcc -c $(TYPELIB).cc $(INCLUDES)

$(RLIB) : $(SPEC)
        cp $(LIBPATH)/librtl.so.cc $(LIBPATH)/librtl.so
        -$(RTL) $(SPEC)
        sed -e "s/tl_lib_$(NAME).h/$(TYPELIB).h/g" $(RLIB) > tmp.cc
        mv tmp.cc $(RLIB)

libclean :
        -rm *~ *.o rtl $(RLIB)

```

## G RJLib.java

```

// user-defined types interface made by rtl -make-lib

public abstract class RJLib {

    static RJLib createLib () {
        return (RJLib) new RJLib_lipsync();
    }

    abstract int getUserFunctionCode (String UserFunctionName);
    abstract RJPacket executeUserFunction (RJPacket aPacket);

}

```

## H RJLib\_lipsync.java

```

// user-defined types interface made by rtl -make-lib

public class RJLib_lipsync extends RJLib {

    final int code_mult = 200;
    final String str_mult = "mult";
    final int code_CreatePacket = 201;
    final String str_CreatePacket = "CreatePacket";
    final int code_PacketId = 202;
    final String str_PacketId = "PacketId";
    final int code_CreateBuffer = 203;
    final String str_CreateBuffer = "CreateBuffer";
    final int code_AddPacket = 204;
    final String str_AddPacket = "AddPacket";
    final int code_PlacePacket = 205;
    final String str_PlacePacket = "PlacePacket";
    final int code_RemovePacket = 206;
    final String str_RemovePacket = "RemovePacket";
    final int code_FirstPacket = 207;
    final String str_FirstPacket = "FirstPacket";
    final int code_BufferEmpty = 208;
    final String str_BufferEmpty = "BufferEmpty";
    final int code_BufferFull = 209;
    final String str_BufferFull = "BufferFull";
    final int code_BufferLoad = 210;
    final String str_BufferLoad = "BufferLoad";

    int getUserFunctionCode (String UserFunctionName) {
        if ( UserFunctionName.compareTo (str_mult) == 0 )
            return code_mult;
        if ( UserFunctionName.compareTo (str_CreatePacket) == 0 )
            return code_CreatePacket;
    }
}

```

```

if ( UserFunctionName.compareTo (str_PacketId) == 0 )
    return code_PacketId;
if ( UserFunctionName.compareTo (str_CreateBuffer) == 0 )
    return code_CreateBuffer;
if ( UserFunctionName.compareTo (str_AddPacket) == 0 )
    return code_AddPacket;
if ( UserFunctionName.compareTo (str_PlacePacket) == 0 )
    return code_PlacePacket;
if ( UserFunctionName.compareTo (str_RemovePacket) == 0 )
    return code_RemovePacket;
if ( UserFunctionName.compareTo (str_FirstPacket) == 0 )
    return code_FirstPacket;
if ( UserFunctionName.compareTo (str_BufferEmpty) == 0 )
    return code_BufferEmpty;
if ( UserFunctionName.compareTo (str_BufferFull) == 0 )
    return code_BufferFull;
if ( UserFunctionName.compareTo (str_BufferLoad) == 0 )
    return code_BufferLoad;
return 0;
}

RJPacket executeUserFunction (RJPacket aPacket) {
int func_code = aPacket.Extract_int(1);
switch (func_code) {
    case code_mult: {
        int par1 = aPacket.Extract_int(5);
        int par2 = aPacket.Extract_int(9);
        int par3 = aPacket.Extract_int(13);
        int res = jt_int.mult(par1,par2,par3);
        RJPacket returnPacket = new RJPacket(4);
        returnPacket.Insert_int(res,0);
        return returnPacket;
    }
    case code_CreatePacket: {
        int par1 = aPacket.Extract_int(5);
        int par2 = aPacket.Extract_int(9);
        jt_pkt res = jt_pkt.CreatePacket(par1,par2);
        RJPacket returnPacket = new RJPacket(4);
        returnPacket.Insert_int(RJDecoder.getObjectCode((jt_for_RTL)res),0);
        return returnPacket;
    }
    case code_PacketId: {
        jt_pkt par1 = (jt_pkt)RJDecoder.getObject(aPacket.Extract_int(5));
        int res = jt_pkt.PacketId(par1);
        RJPacket returnPacket = new RJPacket(4);
        returnPacket.Insert_int(res,0);
        return returnPacket;
    }
    case code_CreateBuffer: {
        int par1 = aPacket.Extract_int(5);
        jt_pktbuf res = jt_pktbuf.CreateBuffer(par1);
        RJPacket returnPacket = new RJPacket(4);
        returnPacket.Insert_int(RJDecoder.getObjectCode((jt_for_RTL)res),0);
        return returnPacket;
    }
    case code_AddPacket: {
        jt_pkt par1 = (jt_pkt)RJDecoder.getObject(aPacket.Extract_int(5));
        jt_pktbuf par2 = (jt_pktbuf)RJDecoder.getObject(aPacket.Extract_int(9));
        jt_pktbuf res = jt_pktbuf.AddPacket(par1,par2);
        RJPacket returnPacket = new RJPacket(4);
        returnPacket.Insert_int(RJDecoder.getObjectCode((jt_for_RTL)res),0);
        return returnPacket;
    }
    case code_PlacePacket: {
        jt_pkt par1 = (jt_pkt)RJDecoder.getObject(aPacket.Extract_int(5));
        jt_pktbuf par2 = (jt_pktbuf)RJDecoder.getObject(aPacket.Extract_int(9));
        jt_pktbuf res = jt_pktbuf.PlacePacket(par1,par2);
        RJPacket returnPacket = new RJPacket(4);
        returnPacket.Insert_int(RJDecoder.getObjectCode((jt_for_RTL)res),0);
        return returnPacket;
    }
    case code_RemovePacket: {
        jt_pktbuf par1 = (jt_pktbuf)RJDecoder.getObject(aPacket.Extract_int(5));
        jt_pktbuf res = jt_pktbuf.RemovePacket(par1);

```

```

        RJPacket returnPacket = new RJPacket(4);
        returnPacket.Insert_int(RJDecoder.getObjectCode((jt_for_RTL)res),0);
        return returnPacket;
    }
    case code_FirstPacket: {
        jt_pktbuf par1 = (jt_pktbuf)RJDecoder.getObject(aPacket.Extract_int(5));
        jt_pkt res = jt_pktbuf.FirstPacket(par1);
        RJPacket returnPacket = new RJPacket(4);
        returnPacket.Insert_int(RJDecoder.getObjectCode((jt_for_RTL)res),0);
        return returnPacket;
    }
    case code_BufferEmpty: {
        jt_pktbuf par1 = (jt_pktbuf)RJDecoder.getObject(aPacket.Extract_int(5));
        boolean res = jt_pktbuf.BufferEmpty(par1);
        RJPacket returnPacket = new RJPacket(1);
        returnPacket.Insert_boolean(res,0);
        return returnPacket;
    }
    case code_BufferFull: {
        jt_pktbuf par1 = (jt_pktbuf)RJDecoder.getObject(aPacket.Extract_int(5));
        boolean res = jt_pktbuf.BufferFull(par1);
        RJPacket returnPacket = new RJPacket(1);
        returnPacket.Insert_boolean(res,0);
        return returnPacket;
    }
    case code_BufferLoad: {
        jt_pktbuf par1 = (jt_pktbuf)RJDecoder.getObject(aPacket.Extract_int(5));
        int res = jt_pktbuf.BufferLoad(par1);
        RJPacket returnPacket = new RJPacket(4);
        returnPacket.Insert_int(res,0);
        return returnPacket;
    }
    default : return null;
}
}
}

```

## I jt\_int.java

```

// RTL packet type

public class jt_int extends jt_for_RTL{

    jt_int(){

    }

    public static int mult(int n1,int d1,int d2){
        return (int)((float)n1*(float)d1/(float)d2);
    }

}

```

## J jt\_pkt.java

```

// RTL packet type

public class jt_pkt extends jt_for_RTL{
    public int Id;
    public int TS;
    public int Delay;

    jt_pkt(int i,int s){
        Id = i;
        TS = s;
        Delay = 0;
    }

    public Object clone(){
        return new jt_pkt(Id,TS);
    }
}

```

```

}

public String toString(){
    return ("Pkt("+Id+", "+TS+", "+Delay+"");
}

public int hashCode(){
    return Id+TS+Delay;
}

public boolean equals(Object O){
    jt_pkt P = (jt_pkt)O;
    return (P.TS == TS) && (P.Id == Id) && (P.Delay == Delay);
}

public static jt_pkt CreatePacket(int i,int s){
    return new jt_pkt(i,s);
}

public static int PacketId(jt_pkt P){
    return P.Id;
}

public static int PacketTS(jt_pkt P){
    return P.TS;
}

public static int PacketD(jt_pkt P){
    return P.Delay;
}

public static jt_pkt SetDelay(int D,jt_pkt P){
    jt_pkt Pkt = (jt_pkt)P.clone();
    Pkt.Delay = D;
    return Pkt;
}

}

```

## K jt\_pktbuf.java

```

// file stimulus type

public class jt_pktbuf extends jt_for_RTL{
    LinkedList Container;
    int Size;
    int Load;

    jt_pktbuf(int s){
        Size=s;
        Load=0;
    }

    public Object clone(){
        jt_pktbuf NewBuff = new jt_pktbuf(Size);
        if(Container != null)
            NewBuff.Container = (LinkedList)Container.clone();
        NewBuff.Load = Load;
        return NewBuff;
    }

    public String toString(){
        if(Load == 0)
            return new String("[]");
        i_LinkedList a_i_list = new i_LinkedList(Container);
        StringBuffer Out = new StringBuffer("");
        while(!(a_i_list.end())){
            jt_pkt elt = (jt_pkt)a_i_list.Item();
            Out.append(elt.toString());
            Out.append(":");
            a_i_list.next();
        }
    }
}

```

```

    }
    Out.append("]");
    return Out.toString();
}

public int hashCode(){
    return 0;
}

public boolean equals(Object Obj){
    jt_pktbuf B = (jt_pktbuf)Obj;
    if((B.Size != Size) && (B.Load != Load))
        return false;
    else
        if(Load > 0)
            return Container.equals(B.Container);
        else return true;
}

public static jt_pktbuf CreateBuffer(int size){
    return new jt_pktbuf(size);
}

public static jt_pktbuf AddPacket(jt_pkt P,jt_pktbuf Buffer){
    jt_pktbuf Copy = (jt_pktbuf)Buffer.clone();
    if(Copy.Container != null)
        Copy.Container.Add(P);
    else
        Copy.Container = new LinkedList(P);
    Copy.Load++;
    return Copy;
}

public static jt_pktbuf PlacePacket(jt_pkt P,jt_pktbuf Buffer){
    jt_pktbuf Copy = (jt_pktbuf)Buffer.clone();
    if(Copy.Container != null)
        Copy.Container.Add(P);
    else
        Copy.Container = new LinkedList(P);
    Copy.Load++;
    return Copy;
}

public static jt_pktbuf RemovePacket(jt_pktbuf Buffer){
    jt_pktbuf Copy = (jt_pktbuf)Buffer.clone();
    if(Copy.Load > 1){
        Copy.Container.Remove();
        Copy.Load--;
    }
    else{
        Copy.Container = null;
        Copy.Load = 0;
    }
    return Copy;
}

public static jt_pkt FirstPacket(jt_pktbuf Buffer){
    jt_pkt P = (jt_pkt)Buffer.Container.first();
    return P;
}

public static int BufferLoad(jt_pktbuf Buffer){
    return Buffer.Load;
}

public static boolean BufferFull(jt_pktbuf Buffer){
    return Buffer.Size == Buffer.Load;
}

public static boolean BufferEmpty(jt_pktbuf Buffer){

```

```

    return Buffer.Load == 0;
}
}

```

## L Makefile java dynamique

```

# ce qu'il faut configurer
NAME = lipsync

# ce qu'il vaut mieux regarder
RTLHOME = ${HOME}/RT-LOTOS
RTL = ${RTLHOME}/bin/rtl -cray
RTLIB = ${RTLHOME}/bin/rtl -make-lib
LIBPATH = lib

CLASSPATH = ${RTLHOME}/class/./
JPARMS = -verbose

#####
SPEC=${NAME}.lot
SIM=${SPEC}.sim

$(SIM) : $(SPEC)
    @hostname
    @date
    -time $(RTL) -max-spec-t-200000000 $(SPEC)
    @date
    @echo "rtl ($(HOST):'pwd'): $(SIM) is done." | mail $(LOGNAME)

clean :
    -rm *~ $(SIM)

javakill :
    -ipcrm -S 100 -S 101 -S 102 -M 103

#####

jlib : RLib.class

RLib.class : RLib.java RLib_$(NAME).class
    javac $(JPARMS) RLib.java

RLib_$(NAME).class : RLib_$(NAME).java
    javac $(JPARMS) RLib_$(NAME).java

RLib.java : $(SPEC)
    -$(RTLIB) $(SPEC)

libclean :
    -rm -f *~ *.class RLib*.java

```

## M Makefile java statique

```

# ce qu'il faut configurer
NAME = lipsync

# ce qu'il vaut mieux regarder
RTLHOME = ${HOME}/RT-LOTOS
RTL = ${RTLHOME}/bin/rtl-java-shmem -cray
RTLIB = ${RTLHOME}/bin/rtl-java-shmem -make-lib

CLASSPATH = ${RTLHOME}/class/./
JPARMS = -verbose

#####

```

```

SPEC=$(NAME).lot
SIM=$(SPEC).sim

$(SIM) : $(SPEC)
    @hostname
    @date
    -time $(RTL) -max-spec-t-200000000 $(SPEC)
    @date
    @echo "rtl ($(HOST):'pwd'): $(SIM) is done." | mail $(LOGNAME)

clean :
    -rm *~ $(SIM)

#####

jlib : RLib.class

RLib.class : RLib.java RLib_$(NAME).class
    javac $(JPARMS) RLib.java

RLib_$(NAME).class : RLib_$(NAME).java
    javac $(JPARMS) RLib_$(NAME).java

RLib.java : $(SPEC)
    -$(RTL) $(SPEC)

libclean :
    -rm -f *~ *.class RLib*.java

```