UNIVERSITÀ DI BRESCIA
FACOLTÀ DI INGEGNERIA
Dipartimento di Elettronica per l'Automazione

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica
(Prof. Riccardo Cassinis)

Nuove librerie per Saphira 8.x

Elaborato di esame di: **Alberto Bettini, Enrico Mansini**

Consegnato il: **13 Maggio 2002**

Sommario

Il lavoro si propone di fornire una metodologia generale per la creazione di librerie personalizzate per Saphira 8.x da parte dell'utente in modo da rendere agevole l'implementazione di nuove funzioni 'ad hoc' in base alle sue necessità, utilizzabili attraverso l'interfaccia Colbert.

Attua quindi le modalità fornite attraverso un lavoro di creazione di una nuova libreria, denominata 'betman', che ha lo scopo di rendere disponibili agli utenti funzionalità presenti nelle versioni precedenti di Saphira (in particolare si fa riferimento alla versione 6.2), ma assenti in quella attualmente in uso. Queste utility riguardano in modo particolare il controllo dei dispositivi di campo sonar montati su robot Pioneer, di cui abbiamo potuto approfondire la conoscenza attraverso gli esemplari presenti nel Laboratorio di Robotica Avanzata.

Oltre alle funzionalità enunciate, ne sono state implementate altre che presentavano la caratteristica di essere potenzialmente utili agli utenti, su indicazione del tesista Trevisson che ha fornito un metro di giudizio grazie alla propria approfondita conoscenza nel settore in questione.

1. Introduzione

Quest'elaborato si propone in primo luogo di documentare in modo approfondito tutti i passi necessari alla realizzazione di nuove librerie per Saphira 8.x, inoltre di realizzare una nuova libreria contenente funzionalità utili all'utente, utilizzabili attraverso l'interfaccia Colbert, in modo da fornire un esempio pratico che possa risultare delucidante per la creazione di future librerie.

Per fornire all'utente una preliminare panoramica del dominio in cui il lavoro è stato svolto, andremo ad esporre una serie di nozioni generali su Saphira e Colbert, per ulteriori approfondimenti rimandiamo il lettore alla manualistica a cui è presente riferimento nelle note bibliografiche.

1.1. Saphira

1.1.1. Saphira Client/Server

Saphira è un ambiente di sviluppo per applicazioni robotiche scritto e costantemente aggiornato da 'SRI International's Artificial Intelligence Center', che ha sviluppato la piattaforma per il robot mobile Pioneer.

Saphira opera in un ambiente client/server, le sue librerie sono costituite da un set di routine per la costruzione di client. Le librerie integrano una serie di funzioni utili per

la spedizione di messaggi al server, l'acquisizione d'informazioni dai sensori del robot e la rappresentazione di essi attraverso un'interfaccia grafica utente basata su finestre. Inoltre Saphira supporta funzioni di più alto livello per il controllo del robot e l'interpretazione dei dati da sensori, tra cui anche un sistema di localizzazion e navigazione basato su mappe.

Il client Saphira attua la connessione al server su robot che possiede i componenti basilari per la navigazione ed il recepimento di segnali esterni: interagisce con motori e ruote, encoder di posizione e sensori. Il server acquisisce dettagli di basso livello riguardanti i sensori e la gestione della guida, spedisce informazioni, e risponde a Saphira attraverso l'interfaccia robot.

Saphira/Aria costituisce quindi un'architettura a tre livelli per il controllo di un robot che mostriamo qui di seguito:

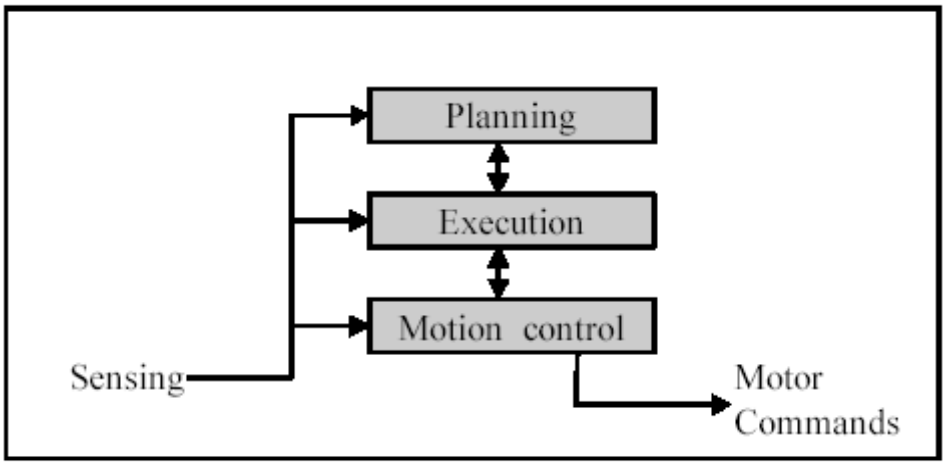

Figura 1.1 – Architettura a tre livelli per il controllo di un robot

Le librerie del client Saphira sono disponibili per Microsoft Windows 98/ME/2000, e per sistemi Linux, il codice sorgente e le librerie di Saphira sono scritte in ANSI C++. Esiste un'interfaccia applicativa per programmatori (API) che contiene le chiamate alle librerie di Saphira.

1.1.2. Saphira e Aria

Saphira 8.x è basato su Aria, un sistema di basso livello per il controllo di robot sviluppato da ActivMedia Robotics. Aria contiene la maggior parte delle funzionalità di basso livello che in precedenza erano incorporate in Saphira ed inoltre una serie di nuove possibilità come un controllo multiplo di robot.

Una buona comprensione di Aria è assolutamente necessaria per lavorare con Saphira in quanto quest'ultima utilizza molte delle classi di Aria. Mentre lo scopo di Aria è fornire un accesso flessibile alle funzionalità di basso livello del robot, Saphira semplifica il compito degli sviluppatori mettendo a disposizione un'interfaccia semplice a tali funzionalità.

Una documentazione approfondita su Aria è elencata nelle note bibliografiche a cui rimandiamo il lettore per un ulteriore approfondimento.

1.2. Colbert

1.2.1. Interazione Colbert/Saphira

Colbert è un linguaggio di programmazione per robot la cui funzione principale è quella di mettere a disposizione funzioni di alto livello interfacciandosi con l'architettura a tre livelli di Saphira/Aria, mostrata in figura 1.1. In particolare rende possibile richiamare comandi motori, costruire sequenze di azioni del robot ed attuare strategie a complessità variabile utilizzando il repertorio di routine sensoriali e di controllo di Saphira.

Colbert è un linguaggio eseguibile, il che significa che le invocazioni delle sue funzioni possono essere direttamente eseguite da un interprete. La prima interazione che la maggior parte degli utenti avranno con il robot avverrà attraverso i comandi Colbert, che permettono di caricare file, definire attività ad hoc per il robot ('activity'), avviare e controllare programmi per esso; abilitano inoltre l'utente a controllare lo stato di un sistema Saphira in esecuzione.

Le strategie di controllo del robot sono sviluppate in Colbert sotto forma di activity, le quali sono semplicemente listati costituiti dai comandi e dalle strutture di controllo di Colbert. Le activity possono essere richiamate ed eseguite attraverso l'interprete d'ambiente e possono richiamare l'intera gamma di comandi messa a disposizione da Colbert.

Colbert è estendibile in quanto le funzioni e gli oggetti interni a Saphira/Aria possono essere resi disponibili attraverso la sua interfaccia; questo include anche oggetti e funzioni definiti direttamente dall'utente e routine predefinite di Saphira/Aria.

Per implementare ed eseguire programmi per il robot sarà quindi sufficiente scriverli in linguaggio C++, creare activity che si interfacciano ad essi richiamando le loro funzioni e quindi eseguire operazioni di start, stop ed interrogazione di stato attraverso i comandi interprete.

1.2.2. Finestra d'interazione Colbert

Nella maggior parte dei casi gli utenti invocheranno comandi per il robot attraverso la finestra d'interazione Colbert; quando Saphira è avviato e carica la propria interfaccia utente, essa crea una finestra principale suddivisa in tre parti:

1. La visualizzazione grafica della geometria del robot nello spazio (parte centrale)
2. Una lista di parametri sullo stato del robot e della comunicazione (parte sinistra)
3. La finestra d'interazione Colbert (parte inferiore)

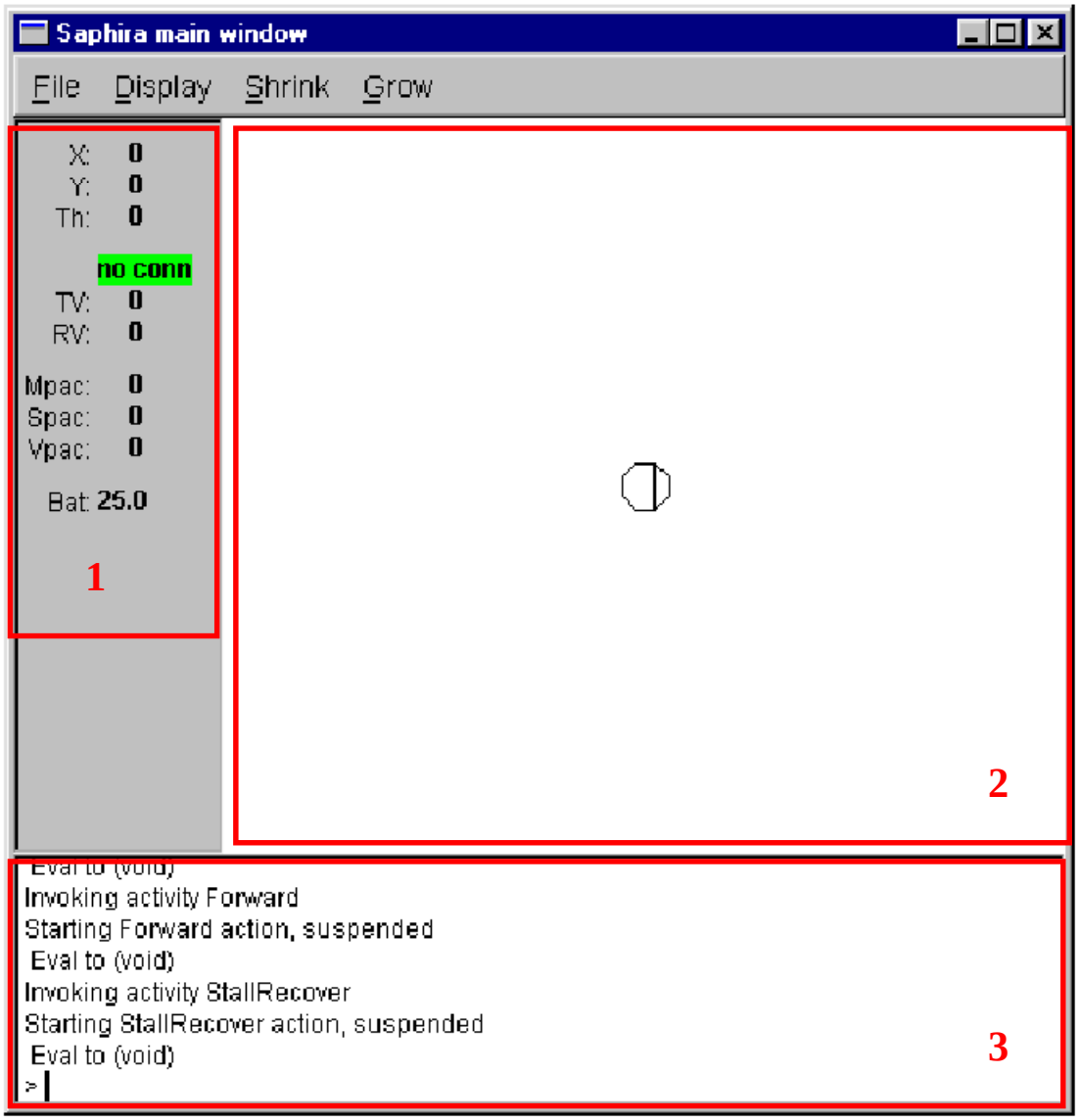

Figura 1.2 – Interfaccia utente di Saphira

Attraverso la finestra d'interazione gli utenti digitano i comandi di Colbert, ognuno di essi è eseguito a seguito della pressione del tasto invio, senza cui esso continua anche nella linea successiva con l'inserimento del carattere backslash ('\'). Il risultato della chiamata viene visualizzato nella stessa finestra, in questo modo gli utenti hanno la possibilità di interrogare lo stato delle variabili interne a Saphira/Aria, rese disponibili da Colbert.

Tutti i comandi richiamati attraverso la linea di comando sono eseguiti in modalità 'noblock' (rimandiamo il lettore ad un relativo approfondimento), per esempio se viene richiesto al robot di muoversi in avanti di un metro ('move(1000)'), viene restituito immediatamente il controllo all'interfaccia anche se il robot impiegherà un certo lasso di tempo per coprire la distanza richiesta.

Oltre al risultato dei comandi, a seguito della loro invocazione verrà anche visualizzato il testo richiamato attraverso la funzione ‘sfMessage’.

2. Il problema affrontato

Il problema affrontato può essere suddiviso in vari task elementari, organizzati in passi di apprendimento ed approfondimento, che permettono di muoversi con cognizione di causa nel campo applicativo, e passi risolutivi i cui prodotti danno luogo alla soluzione adottata. Esponiamo qui di seguito uno schema che mostra i vari task, in cui i passi di apprendimento sono raffigurati con cerchi mentre quelli risolutivi con rettangoli bordati:

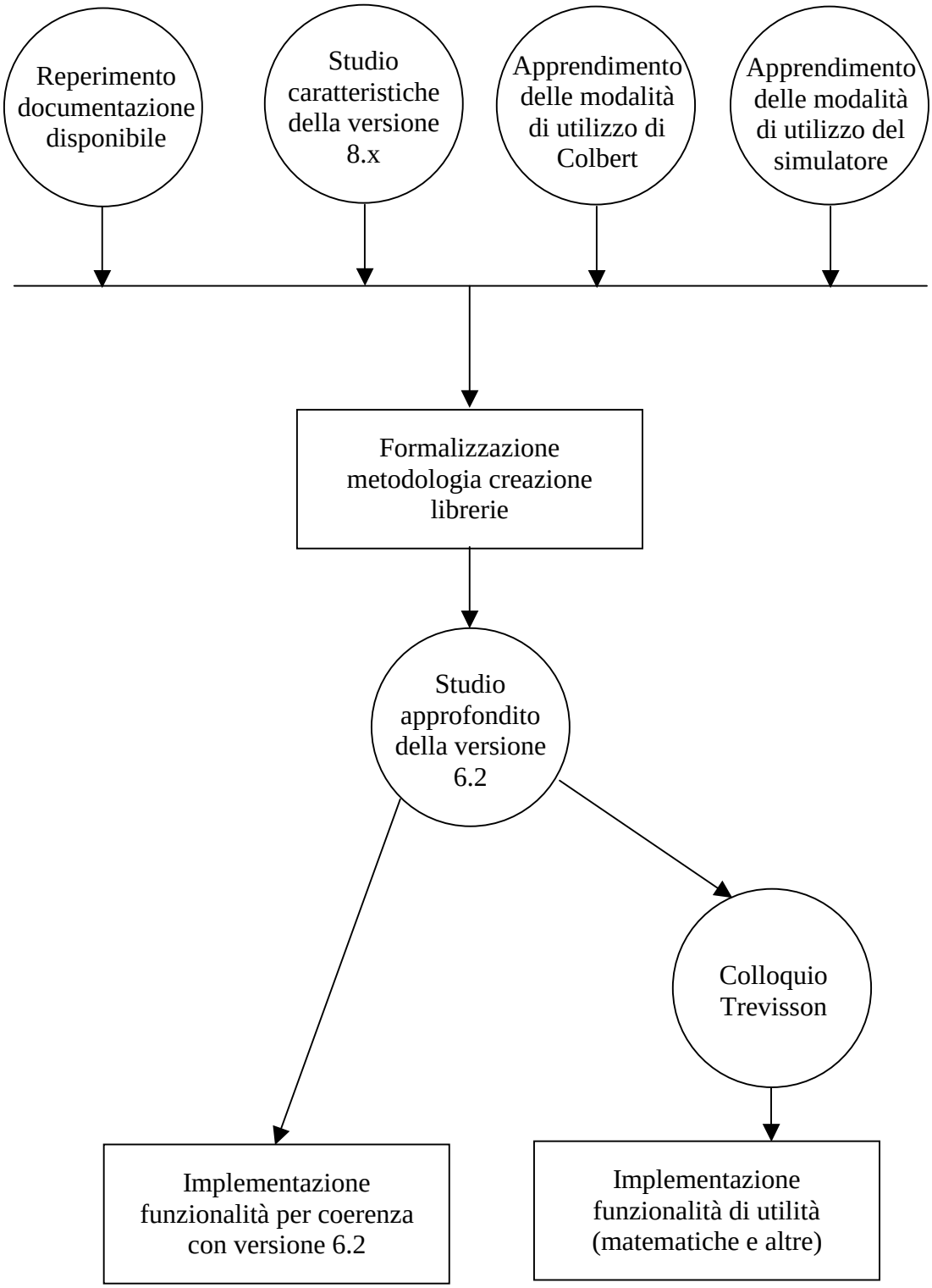

Figura 2.1 – Task elementari del problema affrontato

Il problema sottopostoci concerne in primo luogo la comprensione delle modalità di creazione di nuove librerie personalizzate per la versione di Saphira 8.x. Quindi dopo un primo approccio di apprendimento relativo al campo di studio, esposto a grandi linee nella fase introduttiva, si è passati ad un coinvolgimento maggiormente pratico

attraverso l'utilizzo di manualistica dedicata al 'training' (nello specifico 'Saphira Tutorial' a cui è presente riferimento bibliografico) e prove su esempi ivi illustrati.

I concetti immagazzinati nella fase precedentemente illustrata sono stati quindi impiegati per la soluzione della seconda parte del problema che possiamo suddividere nei successivi paragrafi.

2.1. Coerenza versione 6.2

Innanzitutto ci è stato affidato il compito di rendere la nuova versione di Saphira coerente, dal punto di vista delle funzionalità di gestione dei sonar, con la versione precedente, nello specifico la 6.2.

In quest'ultima la lettura corrente dei sensori sonar faceva riferimento ad una struttura di nome 'sdata', una per ognuno di essi. Veniva quindi automaticamente reso disponibile un vettore chiamato 'sbucket' a cui si poteva fare riferimento per richiamare la struttura relativa al sonar di interesse (per esempio 'sbucket[2]' restituiva la struttura 'sdata' che faceva riferimento al sonar numero 2). La struttura 'sdata' è mostrata qui di seguito così da poter rendere maggiormente agevole la comprensione da parte del lettore:

```
typedef struct /* sonar data collection buffer */
{
float fx, fy, fth; /* robot position when sonar read */
float afx, afy, afth; /* absolute position when sonar read */
float x, y; /* sonar reading in flakey RW coords */
int range; /* sonar range reading in mm */
int snew; /* whether it's a new reading */
} sdata;
IMPORT extern sdata sbucket[]; /* holds one sdata per sonar, indexed by sonar
number */
```

I vari campi indicano rispettivamente la posizione del robot al momento della lettura da parte del sonar in coordinate cartesiane (fx, fy, fth), la posizione assoluta del robot (afx, afy, afth), la lettura del sonar in coordinate relative rispetto al robot (x, y), la distanza assoluta dall'ostacolo individuato (range) e l'eventuale presenza di una nuova lettura (snew). Un eventuale valore di 5000 nel campo 'range' indica che non è stato ricevuto alcun eco dopo l'invio di un segnale da parte del sonar.

La struttura 'sdata' non era però in alcun modo accessibile all'utente attraverso l'interfaccia Colbert per via diretta, ma alcuni dei campi erano raggiungibili attraverso le seguenti funzioni:

- SfSonarRange: restituisce il campo 'range'
- SfSonarXCoord: restituisce il campo 'x'
- SfSonarYCoord: restituisce il campo 'y'
- SfSonarNew: restituisce il campo 'snew'

Questo task di lavoro riguarda quindi l'implementazione di una libreria che rende queste funzioni disponibili anche per la nuova versione di Saphira.

2.2. Funzionalità aggiuntive di utilità

Dopo un colloquio con il tesista Trevisson il sistema attuale è risultato carente dal punto di vista delle funzionalità messe a disposizione dell'utente, in particolare in campo matematico le funzioni trigonometriche risultano assenti così come la radice quadrata. Inoltre nel campo applicativo dei sonar abbiamo ritenuto opportuno fornire utility che permettano di moltiplicare le possibilità di controllo della posizione del robot e del suo stato. Esponiamo qui di seguito un elenco completo con relative note esplicative:

- sfClosestSonarNumber: restituisce il sonar avente la misura della distanza minore da un ostacolo.
- sfClosestSonarRange: restituisce la distanza minore da un ostacolo.
- sfSonarNum: restituisce il numero totale dei sonar presenti sul robot collegato.
- sfFarthestSonarNumber: restituisce il sonar avente la distanza massima da un ostacolo.
- sfFarthestSonarRange: restituisce la distanza massima da un ostacolo.
- sfCos: restituisce il coseno dell'angolo.
- sfSin: restituisce il seno dell'angolo.
- sfSqrt: restituisce la radice quadrata dell'argomento.
- sfTan: restituisce la tangente dell'angolo.

3. La soluzione adottata

3.1. Metodologia di creazione nuove librerie

In questo paragrafo andiamo a mostrare in dettaglio le modalità di implementazione di nuove librerie per l'interfaccia Colbert, che permettono all'utente di definire una serie di nuove funzionalità personalizzate a proprio piacimento. Illustreremo inoltre alcuni concetti fondamentali riguardanti il funzionamento di file oggetto sotto UNIX.

3.1.1. File oggetto condivisibili in UNIX

La maggior parte dei sistemi UNIX permette ai file oggetto (object file) di essere caricati dinamicamente in un programma in esecuzione utilizzando le funzioni 'dl' disponibili (dlopen, dlsym, dlclose).

I file oggetto possono essere compilati a partire da qualsiasi linguaggio sorgente, ma solitamente sono utilizzati i linguaggi C o C++. Una speciale operazione di collegamento (linking) eseguita su questi file crea dei file oggetto condivisibili (sharable object file), i quali contengono informazioni che gli permettono di essere

caricati in un programma in esecuzione. Tipicamente questi file hanno un suffisso '.so'.

Saphira si avvantaggia dell'opzione di caricamento dinamico in UNIX, infatti gli utenti possono creare programmi C/C++ che sono in seguito compilati creando file oggetto condivisibili, questi possono essere poi caricati in un client Saphira in esecuzione e le funzioni al loro interno rese disponibili attraverso l'interfaccia Colbert. In questo modo l'utente può estendere Saphira con nuove funzionalità.

In Saphiara 8.x le librerie sono impostate come una serie di classi C++ e sono costruite appoggiandosi sul sistema Aria creato da ActiveMedia Robotics. I programmi utente accedono ai servizi basilari di Aria e Saphira attraverso queste classi e possono essere scritti con un mix di C e C++, utilizzando le interfacce standard tra i due linguaggi.

A causa delle limitazioni nella sintassi e nella semantica di Colbert non è possibile esprimere interfacce ad oggetti C++ o hai loro metodi direttamente, invece gli utenti, devono creare funzioni globali che accedono ad essi e rendere tali funzioni chiamabili da Colbert.

3.1.2. Compilazione e collegamento di file sorgente C/C++

Per compilare un file oggetto condivisibile caricabile da Colbert è necessario installare le release di Saphira e Aria seguendo le indicazione del file readme, in particolare le variabili d'ambiente di Saphira e Aria devono essere settate ai valori più alti delle loro rispettive release. Inoltre Saphira e Aria dovrebbero essere impostate come sottodirectory di una comune directory.

Dopo l'installazione è necessario seguire i seguenti passi per creare un client o un file oggetto condivisibile:

- Passo 1: scrivere un programma C/C++ contenente il proprio codice, includendo le chiamate alle funzioni di libreria di Saphira.
- Passo 2: compilare il programma e produrre un file oggetto.
- Passo 3: fare un'operazione di collegamento (linking) tra il file oggetto e le librerie di Saphira utilizzate e creare quindi un file oggetto condivisibile o eseguibile.

In Saphira 8.x tutte le funzioni di libreria di Saphira sono contenute in una libreria condivisa, nei sistemi UNIX questa è la libreria 'libsf.so'; mentre per quanto riguarda Aria vi è la libreria 'libAria.so'.

3.1.3. Scrivere programmi client in C/C++

Per caricare nuove funzioni di libreria in Colbert è necessario scrivere uno o più programmi C/C++ che contengono le proprie funzioni e eseguire delle chiamate alle routine di libreria di Saphira. Da notare come siano rimaste attualmente poche funzioni di Saphira in linguaggio C, mentre tutte le funzioni di Aria sono scritte in C++.

Di seguito riportiamo l'esempio con il Saphira Tutorial per poi spiegare nel dettaglio i passi successivi riguardanti l'implementazione di nuove funzionalità.

```
#### testload.cpp ####
#include "Saphira.h"
#include "export.h"                    // need this for MSW, for def of SFEXPORT
#include <math.h>

// global integer variables
// on some systems, these will not be initialized on load...
int nopen = 0;
int globalvar = 1;

// string variable
char buf[80];
char *mystring;                        // we need to set aside space explicitly for a string ptr


//
// C++ global functions
// Set up functions to create and manipulate a point artifact
// This point will draw on the screen
//

SfPoint *mypoint;

int                                    // sets up a point artifact
myfn(int a)
{
  sfMessage("Argument to myfn is: %d",a);
  mypoint = new SfPoint(a,0,0);
  return a+2;
}

float
myErf(float a)
{
#ifndef WIN32
  return (float)((erf(((double)a) / sqrt(2.0)) + 1) / 2.0);
#else
  return 0.0;
#endif
}

void                                   // prints values of a point artifact
showpoint(void *p)
{
  SfPoint *pt = (SfPoint *)p;
  sfMessage("Point x: %d  y: %d  th: %d",
                (int)pt->p.getX(), (int)pt->p.getY(), (int)pt->p.getTh());
}

void                                   // sets the X and Y values of a point
setpoint(void *p, int x, int y)
{
  SfPoint *pt = (SfPoint *)p;
  pt->p.setX((double)x);
  pt->p.setY((double)y);
  sfMessage("Point x: %d  y: %d  th: %d",
                (int)pt->p.getX(), (int)pt->p.getY(), (int)pt->p.getTh());
}
```

```
//
// C++ global functions
// Set up an interface to the sonar buffers
//

int
get_sonar_dist(int begang, int endang, int *retang)
{
  SfSonarDevice *sd = Sf::sonar(); // get the device
  if (!sd)
    return 100000;              // large value, no return
  double ret;
  ArPose rpose = SfROBOT->getPose();
  // Aria occupancy function on the sonar current reading buffer
  double fdist = sd->getCurrent()->getClosestPolar((double)begang, (double)endang,
                                                   rpose, 100000, &ret);
  if (retang)
    *retang = (int)ret;
  return (int)fdist;
}


//
// Motor stalls from SfROBOT
//

int
sfStalledMotor(int which)
{
  if (which == sfLEFT)
    return (int)SfROBOT->isLeftMotorStalled();
  else
    return (int)SfROBOT->isRightMotorStalled();
}

//
// Command to stop the robot and then let behavioral actions go
//

void
sfResetMotion()
{
  SfROBOT->stop();                    // stops robot immediately
  SfROBOT->clearDirectMotion();  // lets behavioral actions through
}


//
// sfLoadInit and sfLoadExit must be declared as exported fns
// under MS Windows
//


SFEXPORT void
sfLoadInit(void)              // this function is evaluated when the object file is loaded
{
  sfMessage("Opened: %d", nopen); // on open, we should get a message
```

```
  printf("Opened!\n");                    // let the console know too...

  // define constants and variables
  //  sfAddEvalConst("sfLEFT",  sfINT, 0);
  sfAddEvalVar("gv", sfINT, (fvalue *)&globalvar); // note indirection in the last argument!
  sfAddEvalVar("mypoint", sfPTR, (fvalue *)&mypoint); // note indirection in the last argument!
  mystring = buf;                 // set up string ptr
  mystring[0] = 'A';
  mystring[1] = 0;
  sfAddEvalVar("mystring", sfSTRING, (fvalue *)&mystring); // note the indirection in the last
argument!


  // define functions
  // any C++ global functions will give an error here if overloaded
  sfAddEvalFn("myfn", (void *)myfn, sfINT, 1, sfINT);
  sfAddEvalFn("myErf", (void *)myErf, sfFLOAT, 1, sfFLOAT);
  sfAddEvalFn("showpoint", (void *)showpoint, sfVOID, 1, sfPTR);
  sfAddEvalFn("setpoint", (void *)setpoint, sfVOID, 3, sfPTR, sfINT, sfINT);
  sfAddEvalFn("get_sonar_dist", (void *)get_sonar_dist, sfINT, 3, sfINT,
                  sfINT, sfTypeRef(sfINT));        // last arg is pointer to int
  sfAddEvalFn("sfResetMotion", (void *)sfResetMotion, sfVOID, 0);

  // set up motor stall interface
  sfAddEvalFn("sfStalledMotor", (void *)sfStalledMotor, sfINT, 1, sfINT);
}


SFEXPORT void
sfLoadExit(void)              /* this should be evaluated on unload */
{
  sfMessage("Unloading: %d", nopen);
}
```

Il file header ‘Saphira.h’ deve essere incluso all’inizio del file sorgente per permettere un successivo collegamento con esso, quindi l’utente può in seguito eseguire chiamate alle librerie di proprio interesse anche al di fuori di quelle di Saphira (nell’esempio math.h). Il programma si sviluppa come un normale listato C/C++, per definire una funzione è sufficiente dichiarare il tipo di dato ritornato (ad esempio integer, char, …) seguito dal nome della funzione e dai parametri necessari alla sua esecuzione e quindi svilupparne il codice.

Dopo tale operazione è quindi necessario creare la funzione ‘sfLoadInit’ che è richiamata dopo il caricamento del file e che rende disponibili le funzioni implementate per l’interfaccia Colbert. Questo avviene attraverso le funzioni ‘sfAddEvalXXX’ che esponiamo qui di seguito:

- sfAddEvalExction(“action_name”, void * function_name, number_of_argoments, type_arg_1, type_arg_2, …): aggiunge un’azione alla lista delle azioni disponibili in Saphira.
  - Ø action_name: nome con cui verrà richiamata l’azione nell’interfaccia Colbert, se per esempio action_name è ‘ExampleAction’ in Colbert sarà sufficiente digitare tale nome seguito dai parametri necessari per richiamare l’azione creata

- Ø function_name: nome della funzione associata all'azione come compare all'interno del programma C/C++
- Ø number_of_argoments: numero degli argomenti che è necessario passare all'azione (contenuti tra parentesi, esempio: ExampleAction(arg_1, arg_2, …))
- Ø type_arg_N: tipo di dato associato all'argomento N-esimo, è necessario che sia uno dei tipi predefiniti da Saphira che sono elencati nel file interprete di libreria 'SfInterp.h' e che andiamo ad illustrare di seguito:
  - sfUNDEFINED (valore associato 0): tipo di dato indefinito che può essere associato ad argomenti dei tipi sfINT, sfFLOAT e sfPTR
  - #define sfINT (1): corrisponde al tipo 'int' di C/C++
  - #define sfFLOAT (2): corrisponde al tipo 'float' di C/C++
  - #define sfVOID (3): corrisponde al tipo 'void' di C/C++
  - #define sfVAR (4):
  - #define sfFUNCTION (5):
  - #define sfEND (6):
  - #define sfACTIVITY (7):
  - #define sfSTRING (8): corrisponde al tipo 'string' di C/C++
  - #define sfACTION (9):
  - #define sfREF (10):
  - #define sfDEREF (11):
  - #define sfDOT (12):
  - #define sfSIZEOF (13):
  - #define sfARROW (14):
  - #define sfFUNCPTR (15): puntatore a funzione
  - #define sfUTASK (16):
  - #define sfPTR (103): puntatore ad un argomento di tipo void

- sfAddEvalConst("const_name", type_const, const_value): aggiunge una constante alla lista delle constanti disponibili in Saphira
  - Ø const_name: nome con cui verrà richiamata la constante nell'interfaccia Colbert, se per esempio const_name è 'ExampleConst' in Colbert sarà sufficiente digitare tale nome per richiamare la costante creata
  - Ø type_const: tipo di dato associato alla costante, è necessario che sia uno dei tipi predefiniti da Saphira
  - Ø const_value: valore associato alla costante consistente con il tipo type_const

- sfAddEvalFn(“function_call_name”, void * function_name, type_returned, number_of_argoments, typa_arg_1, type_arg_2, …):
  - Ø function_call_name: nome con cui verrà richiamata la funzione nell’interfaccia Colbert, se per esempio function_call_name è ‘ExampleFunction’ in Colbert sarà sufficiente digitare tale nome seguito dai parametri necessari per richiamare la funzione creata
  - Ø function_name: nome della funzione come compare all’interno del programma C/C++
  - Ø number_of_argoments: numero degli argomenti che è necessario passare alla funzione (contenuti tra parentesi, esempio: ExampleFunction(arg_1, arg_2, …))
  - Ø type_arg_N: tipo di dato associato all’argomento N-esimo, è necessario che sia uno dei tipi predefiniti da Saphira
- sfAddEvalVar(“var_name”, type_var, var_pointer): aggiunge una variabile alla lista delle constanti disponibili in Saphira
  - Ø var_name: nome con cui verrà richiamata la variabile nell’interfaccia Colbert, se per esempio var_name è ‘examplevar’ in Colbert sarà sufficiente digitare tale nome per richiamare la variabile creata
  - Ø type_var: tipo di dato associato alla variabile, è necessario che sia uno dei tipi predefiniti da Saphira
  - Ø var_pointer: puntatore che permette un collegamento tra la variabile in Colbert e quella all’interno del programma, se per esempio all’interno del programma abbiamo una variabile ‘ex1’ di tipo intero che vogliamo richiamare con ‘examplevar’ in Colbert, inseriremo la seguente riga di codice nel programma:

    sfAddEvalVar(“examplevar”, sfINT, (fvalue *)&ex1)

    da notare il passaggio per indirizzo di ex1 attraverso carattere ‘&’
- sfAddHelp(”help_name”, “help_string”): aggiunge una stringa di help predefinita
  - Ø help_name: nome che l’utente deve digitare per ottenere la stringa di help predefinita nell’interfaccia Colbert
  - Ø help_string: stringa che viene visualizzata quando l’utente digita help_name, se per esempio inseriremo la seguente riga di codice nel programma:

    sfAddlHelp(“ExampleHelp”, “Gives an example to user how to get help ”)

    quando, in Colbert l’utente digiterà “help ExampleHelp” l’interfaccia restituirà la stringa “Gives an example to user how to get help ”
- sfGetHelp(“help_name”): restituisce la stringa di help associata a “help_name”

In ‘testload.cpp’ vengono definite globalvar, mypoint, mystring come variabili e myfn, myErf, showpoint, setpoint, get_sonar_dist, sfResetMotion, sfSatlledMotion come funzioni. Il listato per inizializzare le variabili e le funzioni è il seguente:

```
sfAddEvalVar("gv", sfINT, (fvalue *)&globalvar);
 sfAddEvalVar("mypoint", sfPTR, (fvalue *)&mypoint);
sfAddEvalVar("mystring", sfSTRING, (fvalue *)&mystring);
```

```
sfAddEvalFn("myfn", (void *)myfn, sfINT, 1, sfINT);
sfAddEvalFn("myErf", (void *)myErf, sfFLOAT, 1, sfFLOAT);
sfAddEvalFn("showpoint", (void *)showpoint, sfVOID, 1, sfPTR);
sfAddEvalFn("setpoint", (void *)setpoint, sfVOID, 3, sfPTR, sfINT, sfINT);
sfAddEvalFn("get_sonar_dist", (void *)get_sonar_dist, sfINT, 3, sfINT, sfINT, sfTypeRef(sfINT));
sfAddEvalFn("sfResetMotion", (void *)sfResetMotion, sfVOID, 0);
sfAddEvalFn("sfStalledMotor", (void *)sfStalledMotor, sfINT, 1, sfINT);
```

Dopo aver terminato la stesura della funzione ‘sfLoadInit’ è necessario introdurre la funzione ‘sfLoadExit’ che verrà richiamata al momento dell’unload del file dall’interfaccia.

3.1.4. Compilazione e collegamento di programmi client in UNIX

Dopo aver scritto i programmi client, devono essere compilati utilizzando un compilatore C/C++. Viene raccomandato il compilatore ‘gcc’ per sistemi UNIX; tutti i programmi esempio sono stati compilati con questo compilatore, altri compilatori C forniti con UNIX dovrebbero comunque funzionare correttamente.

Il compilatore ed il linker sono tipicamente chiamati utilizzando l’opzione ‘make’, il file ‘makefile’ presente nella directoy ‘/Saphira/tutor/loadable/’ è stato creato per il file ‘testload.cpp’ ed è esposto qui di seguito:

```
##
## Makefile for the loadable tutorial
##

###############################################################

SRCD = ./
OBJD = ./
INCD = ../../ohandler/include/
BIND = ../../bin/
LIBD = ../../lib/
ARIAD = ../../../Aria/

# check which OS we have
include $(INCD)os.h

SHELL = /bin/sh

INCLUDE = -I$(INCD) -I$(ARIAD)include

###############################################################
all: $(LIBD)testload.so
	touch all

$(OBJD)testload.o: $(SRCD)testload.cpp
	$(CC) $(CFLAGS) -c $(SRCD)testload.cpp $(INCLUDE) -o $(OBJD)testload.o

$(LIBD)testload.so: $(OBJD)testload.o
	$(LD) $(SHARED) -o $(LIBD)testload.so $(OBJD)testload.o
```

La prima parte del ‘makefile’ definisce variabili che sono usate in compilazione e nella fase di linking. Da notare è che le directory Saphira ed Aria devono risiedere in

una directory comune. La directory ‘ohandler/include’ contiene i file header (ossia file con estensione ‘.h’), la directory ‘/lib’ contiene le librerie di Saphira (ossia i file con estensione ‘.so’) ed è in tale directory che vengono messi i file oggetto condivisibili.

Il file ‘ohandler/include/os.h’ permette di determinare il tipo di sistema operativo in uso e setta in modo appropriato alcune delle variabili delle librerie di sistema ed inoltre setta la variabile ‘CONFIG’ rispetto al particolare sistema operativo della macchina, il che permette di eseguire alcuni importanti routine di sistema correttamente.

Il ‘makefile’ produce il file ‘testload.so’ nella directory ‘/lib’ di Saphira; ‘testload.so’ è un file oggetto condivisibile, che è richiamabile attraverso l’interfaccia Colbert. Il codice sorgente in C++ è compilato in modo standard; si consiglia di prestare particolare attenzione alla presenza dell’opzione ‘-g’ poiché questa permette di aggiungere i simboli di debug al file oggetto.

In alcuni sistemi UNIX, il codice sorgente deve essere compilato in modo indipendente dalla locazione in cui si trova (PIC, Position-Indipendent Compilation) questo dettaglio è manipolato dalla variabile ‘PICFLAG’ presente in ‘os.h’.

Saphira richiede obbligatoriamente per un file oggetto condivisibile che possa essere utilizzato l’estensione ‘.so’, in caso contrario non sarà possibile procedere al suo caricamento.

3.1.5. Caricamento di file oggetto condivisibili

I file oggetto condivisibili, con estensione finale ‘.so’, possono essere caricati da un client Saphira usando la finestra di interazione Colbert attraverso il comando ‘loadlib’ che eseguirà il caricamento andando ha ricercare il file prima nella directory corrente e quindi nelle directory di default (di cui fa sempre parte la directory ‘/lib’).

Ad esempio per caricare ‘testload.so’, usiamo il seguente comando

loadlib testload.so

Da notare che l’estensione ‘.so’ non è necessaria per il comando ‘loadlib’.

Se il caricamento è avvenuto con successo viene comunicato il nome del file caricato e quindi viene dichiarata l’eventuale presenza della funzione ‘sfLoadInit’, attraverso il messaggio “Opened: 0” generato attraverso la chiamata a funzione ‘sfMessage’.

L’operazione di ‘unloading’ del file oggetto condivisibile può venire eseguita con il comando ‘unload’. Generalmente nei sistemi UNIX non è necessario eseguire in modo esplicito l’operazione di ‘unloading’ del file poiché la reiterazione del caricamento dello stesso file causa un ‘unload’ automatico prima di essere eseguito. Il comando ‘unload’ è solamente un funzione di convenienza per sistemi MS Windows, dove i file oggetto condivisibili non possono essere ricompilati se sono in uso.

Nel caso in cui venga riscontrato un errore nel codice oggetto condivisibile, è necessario fare opportuni cambiamenti, eseguire una ricompilazione seguita da un’operazione di ‘relink’ sul codice; all’esecuzione dell’operazione di caricamento attraverso l’istruzione ‘load’ le nuove funzioni che sono state definite rimpiazzeranno quelle vecchie.

Una delle conseguenze dell'operazione di 'reloading' è la perdita dei valori precedenti di tutte le variabili globali che vengono resettate al loro valore di inizializzazione, nell'esempio illustrato 'nopen' è portata al valore 0 al caricamento di 'testload.so', ed anche se è incrementata alla chiamata della funzione 'sfLoadInit', viene resettata al valore di inizializzazione ad ogni chiamata 'load' in modo che non possa mai superare il valore 1.

3.1.6. Inizializzazioni al caricamento

Esiste una peculiarità nei file oggetto condivisibili caricati dinamicamente, infatti, a differenza degli altri file oggetto, le variabili globali inizializzate potrebbero non essere caricate con tali valori all'operazione di caricamento. L'unico modo sicuro per eseguire l'inizializzazione correttamente è inserire le operazioni di settaggio all'interno della funzione 'sfLoadInit', altrimenti non si ha alcuna sicurezza del risultato ottenuto.

3.2. Libreria 'betman.so'

Seguendo la modalità di creazione di nuove librerie sopra esposta abbiamo realizzato le funzioni richieste. La nostra libreria realizza varie funzioni tra cui quelle per gestire il sonar, alcune funzioni matematiche (ad esempio seno, coseno, tangente, …) ed alcune funzioni di aiuto. Di seguito riportiamo il listato della libreria 'betman.so':

```
/* ########################################
 *
 * betman.h
 * Library
 *
 * This library povides a set of functions
 * for Colbert, most of them was included in
 * the 6.2c Saphira's Version, but actually 7
 * missing in 8.* versions.
 *
 *########################################
 */



#include "Saphira.h"
#include "math.h"

#define M_PI  3.14159265358979323846

int nopen=0;

// ########## SONAR FUNCTIONS ##########

/// Returns the number of the sonar that has the closest current reading in the given range
/// startAngle and endAngle are in degrees

int sfClosestSonarNumber(float startAngle, float endAngle)
{
  return(SfROBOT->getClosestSonarNumber((double)startAngle,(double)endAngle));
}
```

```
/// Returns the closest of the current sonar reading in the given range
/// startAngle and endAngle are in degrees

int sfClosestSonarRange(float startAngle,float endAngle)
{
  return (SfROBOT->getClosestSonarRange((double)startAngle,(double)endAngle));
}


/// Find the number of sonar there are

int sfSonarNum()
{
  return (SfROBOT->getNumSonar());
}


/// Returns the number of the sonar that has the farthest current reading

int sfFarthestSonarNumber()
{int max_distance=0;
 int num_sonar=0;
 for(int i=0;i<sfSonarNum();i++)
{
 int lettura=SfROBOT->getSonarRange(i);
 if((i==(int)ceil((sfSonarNum()-1)/2))&&(lettura==max_distance))
   num_sonar=(int)ceil((sfSonarNum()-1)/2);

 if(lettura>max_distance)
{
   max_distance=lettura;
   num_sonar=i;
}

}
return num_sonar;
}


/// Returns farthest current sonar  reading

int sfFarthestSonarRange()
{ int max_distance=0;
  for(int i=0;i<sfSonarNum();i++)
{
  if(SfROBOT->getSonarRange(i)>max_distance);
   max_distance=SfROBOT->getSonarRange(i);
}
   return max_distance;
}


/// Find out if the given sonar(num_sonar) has a new reading

int sfSonarNew(int num_sonar)
{
  return (int)(SfROBOT->isSonarNew(num_sonar));
```

```
}


/// Gets the range of the last sonar reading for the given sonar(num_sonar)

int sfSonarRange(int num_sonar)

{
  return SfROBOT->getSonarRange(num_sonar);
}


/// Gets the X location of the given sonar(num_sonar) reading

float sfSonarXCoord(int num_sonar)
{
  //Gets reading from sonar 'num' using class ArSensorReading
  ArSensorReading *vect1=(SfROBOT->getSonarReading(num_sonar));

  //Component in the X direction of the reading
  float x;

  //Get x with ArSensorReading method getX()
  x=(float) (vect1->getX());

  return x;
}


/// Gets the Y location of the given sonar(num_sonar) reading

float sfSonarYCoord(int num_sonar)
{
  //Gets reading from sonar 'num' using class ArSensorReading
  ArSensorReading *vect1=(SfROBOT->getSonarReading(num_sonar));

  //Component in the Y direction of the reading
  float y;

  //Get y with ArSensorReading method getX()
  y=(float) (vect1->getY());

  return y;
}




// ########## MATEMATICAL FUNCTIONS ##########


///Gets the cosine of 'angle'(in degrees)

float sfCos(float angle)
{
  //Converts 'angle' to radiant system
  double radiant=(angle/180)*M_PI;
```

```
  float result=((float)cos(radiant));
  return result;
}


///Gets the sine of 'angle'(in degrees)

float sfSin(float angle)
{
  //Converts 'angle' to radiant system
  double radiant=(angle/180)*M_PI;

  float result=((float)sin(radiant));
  return result;
}


///Gets the square root of 'argoment'

float sfSqrt(float argoment)
{
  float result=((float)sqrt(argoment));
  return result;
}


///Gets the tangent of 'angle'(in degrees)

float sfTan(float angle)
{
  //Converts 'angle' to radiant system
  double radiant=(angle/180)*M_PI;

  float result=((float)tan(radiant));
  return result;
}




// ########## SYSTEM FUNCTIONS ##########


//This function is evaluated when the objectfile is loaded

SFEXPORT void
sfLoadInit(void)
{
  sfMessage("Opened: %d", nopen);
  printf("Opened!\n");                    // let the console know too...


  //Define functions

  /*GENERAL METHOD:
  **
  ** sfAddEvalFn("name_of_colbert_function",(void *)name_of_program_function,
  **             type_returned_by_function,number_of_parameters,
  **             type_parameter_1,type_of_parameter_2,....);
```

```
  **
  */

  //Any C++ global functions will give an error here if overloaded

  //SONAR FUNCTIONS

  sfAddEvalFn("sfSonarRange", (void *)sfSonarRange, sfINT, 1, sfINT);
  sfAddEvalFn("sfSonarXCoord", (void *)sfSonarXCoord, sfFLOAT, 1, sfINT);
  sfAddEvalFn("sfSonarYCoord", (void *)sfSonarYCoord, sfFLOAT, 1, sfINT);
  sfAddEvalFn("sfSonarNew", (void *)sfSonarNew, sfINT, 1, sfINT);
  sfAddEvalFn("sfSonarNum", (void *)sfSonarNum, sfINT, 0);
  sfAddEvalFn("sfClosestSonarRange", (void *)sfClosestSonarRange, sfINT, 2,
            sfFLOAT,sfFLOAT);
  sfAddEvalFn("sfClosestSonarNumber", (void *)sfClosestSonarNumber, sfINT, 2,
            sfFLOAT,sfFLOAT);
  sfAddEvalFn("sfFarthestSonarRange", (void *)sfFarthestSonarRange, sfINT, 0);
  sfAddEvalFn("sfFarthestSonarNumber", (void *)sfFarthestSonarNumber, sfINT,
0);


  //MATH FUNCTIONS

  sfAddEvalFn("sfSin", (void *)sfSin, sfFLOAT, 1, sfFLOAT);
  sfAddEvalFn("sfCos", (void *)sfCos, sfFLOAT, 1, sfFLOAT);
  sfAddEvalFn("sfTan", (void *)sfTan, sfFLOAT, 1, sfFLOAT);
  sfAddEvalFn("sfSqrt", (void *)sfSqrt, sfFLOAT, 1, sfFLOAT);


 sfAddHelp("betman","
sfSonarRange    sfSonarXCoord    sfSonarYCoord
sfSonarNew      sfSonarNum
sfClosestSonarRange    sfClosestSonarNumber
sfFarthestSonarRange    sfFarthestSonarNumber
sfSin    sfCos    sfTan    sfSqrt");



sfAddHelp("sfClosestSonarNumber",
"
int sfSonarNumber(float startAngle,float endAngle)
Returns the number of the sonar that has the closest current reading in the given range
----startAngle and endAngle are in degrees-----
");

sfAddHelp("sfClosestSonarRange",
"
int sfClosestSonarRange(float startAngle,float endAngle)
Returns the closest of the current sonar readings in the given range
-----startAngle and endAngle are in degrees-----
");

sfAddHelp("sfSonarNum",
"
int sfSonarNum()
Returns the number of sonar there are
");
```

```
sfAddHelp("sfFarthestSonarNumber",
"
int sfFarthestSonarNumber()
Returns the number of the sonar that has the farthest current reading
");

sfAddHelp("sfFarthestSonarRange",
"
int sfFarthestSonarRange()
Returns farthest current sonar reading
");

sfAddHelp("sfSonarNew",
"
int sfSonarNew(int num_sonar)
Find out if the given sonar(num_sonar) has a new reading
");

sfAddHelp("sfSonarXCoord",
"
float sfSonarXCoord(int num_sonar)
Gets the X location of the given sonar(num_sonar) reading
");

sfAddHelp("sfSonarYCoord",
"
float sfSonarYCoord(int num_sonar)
Gets the Y location of the given sonar(num_sonar) reading
");

sfAddHelp("sfCos",
"
float sfCos(float angle)
Gets the cosine of 'angle'
-----angle in degrees-----
");

sfAddHelp("sfSin",
"
float sfSin(float angle)
Gets the sine of 'angle'
-----angle in degrees-----
");

sfAddHelp("sfSqrt",
"
sfSqrt(float argoment)
Gets the square root of 'argoment'
");

sfAddHelp("sfTan",
"
float sfTan(float angle)
Gets the tangent of 'angle'
-----angle in degrees-----
");

}
```

```
SFEXPORT void
sfLoadExit(void)              /* this should be evaluated on unload */
{
  sfMessage("Unloading: %d", nopen);
}
```

Le funzionalità ‘sfSonarRange, sfSonarXCoord, sfSonarYCoord, sfSonarNew’ sono state realizzate appositamente per fornire agli utenti le stesse possibilità operative nella gestione dei sonar della versione 6.2c di Saphira. A queste si sono aggiunte delle funzioni personalizzate che abbiamo ritenuto potenzialmente utili agli utenti mantenendoci nel campo dei sonar: ‘sfSonarNum, sfClosestSonarRange, sfClosestSonarNumber, sfFarthestSonarRange, sfFarthestSonarNumber’.

Tutte queste funzioni fanno riferimento al loro interno ad una variabile sfROBOT di tipo ArRobot, questa è una variabile di sistema, messa a disposizione dall’interfaccia Colbert, che fa riferimento al robot attualmente collegato al client Saphira.

Esponiamo qui di seguito la classe ArRobot in modo che siano visibili i metodi messi a disposizione e da noi utilizzati:

```
#ifndef ARROBOT_H
#define ARROBOT_H


/// THE important class
/**
   This is the most important class, the only classes most people will ever
   have to use are this one, and the ArSerialConnection and ArTCPConnection.
   NOTE:  In Windows you cannot make an ArRobot a global, it will crash
   because the windows compiler initializes the constructors in the wrong
   order... you can make a pointer to an ArRobot and then new one however.
   @see ArSerialConnection
   @see ArTcpConnection
*/
class ArRobot
{
public:

  typedef enum {
    WAIT_CONNECTED, ///< The robot has connected
    WAIT_FAILED_CONN, ///< The robot failed to connect
    WAIT_RUN_EXIT, ///< The run loop has exited
    WAIT_TIMEDOUT, ///< The wait reached the timeout specified
    WAIT_INTR, ///< The wait was interupted by a signal
    WAIT_FAIL ///< The wait failed due to an error
  } WaitState;

  /// Constructor
  AREXPORT ArRobot(const char * name = NULL, bool doStateReflection = true,
                   bool doSigHandle=true,
                   bool normalInit = true);

  /// Destructor
  AREXPORT ~ArRobot(void);
```

```
/// Starts the instance to do processing
AREXPORT void run(bool stopRunIfNotConnected);
/// Starts the instance to do processing in its own new thread
AREXPORT void runAsync(bool stopRunIfNotConnected);

/// Returns whether the robot is currently running or not
AREXPORT bool isRunning(void);

/// Stops the robot from doing any more processing
AREXPORT void stopRunning(bool doDisconnect=true);

/// Sets the connection this instance uses
AREXPORT void setDeviceConnection(ArDeviceConnection *connection);
/// Gets the connection this instance uses
AREXPORT ArDeviceConnection *getDeviceConnection(void);

/// Questions whether the robot is connected or not
/**
   @return true if connected to a robot, false if not
*/
AREXPORT bool isConnected(void) { return myIsConnected; }
/// Connects to a robot, not returning until connection made or failed
AREXPORT bool blockingConnect(void);
/// Connects to a robot, from the robots own thread
AREXPORT bool asyncConnect(void);
/// Disconnects from a robot
AREXPORT bool disconnect(void);

/// Clears what direct motion commands have been given, so actions work
AREXPORT void clearDirectMotion(void);
/// Returns true if direct motion commands are blocking actions
AREXPORT bool isDirectMotion(void);

/// Enables the motors on the robot
AREXPORT void enableMotors();
/// Disables the motors on the robot
AREXPORT void disableMotors();

/// Stops the robot
/// @see clearDirectMotion
AREXPORT void stop(void);
/// Sets the velocity
/// @see clearDirectMotion
AREXPORT void setVel(double velocity);
/// Sets the velocity of the wheels independently
/// @see clearDirectMotion
AREXPORT void setVel2(double leftVelocity, double rightVelocity);
/// Move the given distance forward/backwards
/// @see clearDirectMotion
AREXPORT void move(double distance);
/// Sees if the robot is done moving the previously given move
AREXPORT bool isMoveDone(double delta = 0.0);
/// Sets the difference required for being done with a move
AREXPORT void setMoveDoneDist(double dist) { myMoveDoneDist = dist; }
/// Gets the difference required for being done with a move
AREXPORT double getMoveDoneDist(void) { return myMoveDoneDist; }
/// Sets the heading
/// @see clearDirectMotion
```

```
AREXPORT void setHeading(double heading);
/// Sets the rotational velocity
/// @see clearDirectMotion
AREXPORT void setRotVel(double velocity);
/// Sets the delta heading
/// @see clearDirectMotion
AREXPORT void setDeltaHeading(double deltaHeading);
/// Sees if the robot is done changing to the previously given setHeading
AREXPORT bool isHeadingDone(double delta = 0.0);
/// sets the difference required for being done with a heading change
AREXPORT void setHeadingDoneDiff(double degrees)
  { myHeadingDoneDiff = degrees; }
/// Gets the difference required for being done with a heading change
AREXPORT double get(void) { return myHeadingDoneDiff; }


/// Sets the length of time a direct motion command will take precedence
/// over actions, in milliseconds
AREXPORT void setDirectMotionPrecedenceTime(int mSec);

/// Gets the length of time a direct motion command will take precedence
/// over actions, in milliseconds
AREXPORT unsigned int getDirectMotionPrecedenceTime(void);

/// Sends a command to the robot with no arguments
AREXPORT bool com(unsigned char command);
/// Sends a command to the robot with an int for argument
AREXPORT bool comInt(unsigned char command, short int argument);
/// Sends a command to the robot with two bytes for argument
AREXPORT bool com2Bytes(unsigned char command, char high, char low);
/// Sends a command to the robot with a string for argument
AREXPORT bool comStr(unsigned char command, const char *argument);
/// Sends a command to the robot with a size bytes of str as argument
AREXPORT bool comStrN(unsigned char command, const char *str, int size);

/// Returns the Robot's name that is set in its onboard configuration
AREXPORT std::string getRobotName(void) { return myRobotName; }
/// Returns the type of the robot connected to
AREXPORT std::string getRobotType(void) { return myRobotType; }
/// Returns the subtype of the robot connected to
AREXPORT std::string getRobotSubType(void) { return myRobotSubType; }

/// Gets the robots maximum translational velocity
AREXPORT double getMaxTransVel(void) { return myMaxTransVel; }
/// Sets the robots maximum translational velocity
AREXPORT bool setMaxTransVel(double maxVel);

/// Gets the robots maximum rotational velocity
AREXPORT double getMaxRotVel(void) { return myMaxRotVel; }
/// Sets the robots maximum rotational velocity
AREXPORT bool setMaxRotVel(double myMaxVel);

// Accessors

/// Gets the global position of the robot
ArPose getPose(void) { return myGlobalPose; }
/// Gets the global X location of the robot
double getX(void) { return myGlobalPose.getX(); }
/// Gets the global Y location of the robot
```

```
double getY(void) { return myGlobalPose.getY(); }
/// Gets the global Th location of the robot
double getTh(void) { return myGlobalPose.getTh(); }
/// Gets the translational velocity of the robot
double getVel(void) { return myVel; }
/// Gets the rotational velocity of the robot
double getRotVel(void) { return myRotVel; }
/// Gets the robot radius (in mm)
double getRobotRadius(void) { return myParams.getRobotRadius(); }
/// Gets the robot diagonal (half-height to diagonal of octagon) (in mm)
double getRobotDiagonal(void) { return myParams.getRobotDiagonal(); }
/// Gets the battery voltage of the robot
double getBatteryVoltage(void) { return myBatteryVoltage; }
/// Gets the velocity of the left wheel
double getLeftVel(void) { return myLeftVel; }
/// Gets the velocity of the right wheel
double getRightVel(void) { return myRightVel; }
/// Gets the 2 bytes of stall return from the robot
int getStallValue(void) { return myStallValue; }
/// Returns true if the left motor is stalled
bool isLeftMotorStalled(void)
  { return (myStallValue & 0xff) & ArUtil::BIT0; }
/// Returns true if the left motor is stalled
bool isRightMotorStalled(void)
  { return ((myStallValue & 0xff00) >> 8) & ArUtil::BIT0; }
/// Gets the control heading
/**
  Gets the control heading as an offset from the current heading.
  @see getTh
*/
double getControl(void) { return myControl; }
/// Gets the flags values
int getFlags(void) { return myFlags; }
/// returns true if the motors are enabled
bool areMotorsEnabled(void) { return (myFlags & ArUtil::BIT0); }
/// returns true if the motors are enabled
bool areSonarsEnabled(void) { return (myFlags & ArUtil::BIT1); }
/// Gets the compass heading from the robot
double getCompass(void) { return myCompass; }
/// Gets which analog port is selected
int getAnalogPortSelected(void) { return myAnalogPortSelected; }
/// Gets the analog value
unsigned char getAnalog(void) { return myAnalog; }
/// Gets the byte representing digital input status
unsigned char getDigIn(void) { return myDigIn; }
/// Gets the byte representing digital output status
unsigned char getDigOut(void) { return myDigOut; }

/// Gets whether the robot has table sensing IR or not (see params in docs)
bool hasTableSensingIR(void) { return myParams.haveTableSensingIR(); }
/// Gets whether the robot has front bumpers (see params in docs)
bool hasFrontBumpers(void) { return myParams.haveFrontBumpers(); }
/// Gets whether the robot has rear bumpers (see params in docs)
bool hasRearBumpers(void) { return myParams.haveRearBumpers(); }

/// Gets the position of the robot according to the encoders
ArPose getEncoderPose(void) { return myEncoderPose; }

/// Gets the number of motor packets received in the last second
```

```
AREXPORT int getMotorPacCount(void);
/// Gets the number of sonar returns received in the last second
AREXPORT int getSonarPacCount(void);

/// Gets the range of the last sonar reading for the given sonar
AREXPORT int getSonarRange(int num);
/// Find out if the given sonar has a new reading
AREXPORT bool isSonarNew(int num);
/// Find the number of sonar there are
AREXPORT int getNumSonar(void) { return myNumSonar; }
/// Returns the sonar reading for the given sonar
AREXPORT ArSensorReading *getSonarReading(int num);
/// Returns the closest of the current sonar reading in the given range
AREXPORT int getClosestSonarRange(double startAngle, double endAngle);
/// Returns the number of the sonar that has the closest current reading in the given range
AREXPORT int getClosestSonarNumber(double startAngle, double endAngle);

/// Gets the robots name in ARIAs list
AREXPORT std::string getName(void);
/// Sets the robots name in ARIAs list
AREXPORT void setName(const char *name);

/// Moves the robot's idea of its position to this position
AREXPORT void moveTo(ArPose pose, bool doCumulative = true);
/// Moves the robot's RW position to reflect pose From => pose To
AREXPORT void moveTo(ArPose to, ArPose from, bool doCumulative = true);

/// Changes the transform
AREXPORT void setEncoderTransform(ArPose deadReconPos,
                                  ArPose globalPos);

/// Changes the transform directly
AREXPORT void setEncoderTransform(ArPose transformPos);

/// This gets the transform from local coords to global coords
AREXPORT ArTransform getToGlobalTransform(void);

/// This gets the transform for going from global coords to local coords
AREXPORT ArTransform getToLocalTransform(void);

/// This applies a transform to all the robot range devices and to the sonar
AREXPORT void applyTransform(ArTransform trans, bool doCumulative = true);

/// Sets the dead recon position of the robot
AREXPORT void setDeadReconPose(ArPose pose);

/// Adds a rangeDevice to the robot's list of them, and set the device's robot pointer
AREXPORT void addRangeDevice(ArRangeDevice *device);
/// Remove a range device from the robot's list, by name
AREXPORT void remRangeDevice(const char *name);
/// Remove a range device from the robot's list, by instance
AREXPORT void remRangeDevice(ArRangeDevice *device);

/// Finds a rangeDevice in the robot's list
AREXPORT ArRangeDevice *findRangeDevice(const char *name);

/// Gets the range device list
AREXPORT std::list<ArRangeDevice *> *getRangeDeviceList(void);
```

```
/// Finds whether a particular range device is attached to this robot or not
AREXPORT bool hasRangeDevice(ArRangeDevice *device);

/// Goes through all the range devices and checks them
AREXPORT double checkRangeDevicesCurrentPolar(double startAngle,
                                              double endAngle,
                                              double *angle = NULL);

/// Goes through all the range devices and checks them
AREXPORT double checkRangeDevicesCumulativePolar(double startAngle,
                                              double endAngle,
                                              double *angle = NULL);


// Goes through all the range devices and checks them
AREXPORT double checkRangeDevicesCurrentBox(double x1, double y1,
                                            double x2, double y2,
                                            ArPose *readingPos = NULL);

// Goes through all the range devices and checks them
AREXPORT double checkRangeDevicesCumulativeBox(double x1, double y1,
                                               double x2, double y2,
                                             ArPose *readingPos = NULL);

/// Sets the number of milliseconds between state reflection refreshes
/// if the state has not changed
AREXPORT void setStateReflectionRefreshTime(int msec);

/// Sets the number of milliseconds between state reflection refreshes
/// if the state has not changed
AREXPORT int getStateReflectionRefreshTime(void);

/// Adds a packet handler to the list of packet handlers
AREXPORT void addPacketHandler(
        ArRetFunctor1<bool, ArRobotPacket *> *functor,
        ArListPos::Pos position);

/// Removes a packet handler from the list of packet handlers
AREXPORT void remPacketHandler(
        ArRetFunctor1<bool, ArRobotPacket *> *functor);

/// Adds a connect callback
AREXPORT void addConnectCB(ArFunctor *functor, ArListPos::Pos position);
/// Adds a disconnect callback
AREXPORT void remConnectCB(ArFunctor *functor);

/// Adds a callback for when a connection to the robot is failed
AREXPORT void addFailedConnectCB(ArFunctor *functor,
                                 ArListPos::Pos position);
/// Removes a callback for when a connection to the robot is failed
AREXPORT void remFailedConnectCB(ArFunctor *functor);

/// Adds a callback for when disconnect is called while connected
AREXPORT void addDisconnectNormallyCB(ArFunctor *functor,
                                      ArListPos::Pos position);
/// Removes a callback for when disconnect is called while connected
AREXPORT void remDisconnectNormallyCB(ArFunctor *functor);

/// Adds a callback for when disconnection happens because of an error
```

```
AREXPORT void addDisconnectOnErrorCB(ArFunctor *functor,
                                     ArListPos::Pos position);
/// Removes a callback for when disconnection happens because of an error
AREXPORT void remDisconnectOnErrorCB(ArFunctor *functor);

/// Adds a callback for when the run loop exits for what ever reason
AREXPORT void addRunExitCB(ArFunctor *functor, ArListPos::Pos position);
/// Removes a callback for when the run loop exits for what ever reason
AREXPORT void remRunExitCB(ArFunctor *functor);

/// Suspend calling thread until the ArRobot is connected
AREXPORT WaitState waitForConnect(unsigned int msecs=0);
/// Suspend calling thread until the ArRobot is connected or fails to connect
AREXPORT WaitState waitForConnectOrConnFail(unsigned int msecs=0);
/// Suspend calling thread until the ArRobot run loop has exited
AREXPORT WaitState waitForRunExit(unsigned int msecs=0);

/// Wake up all threads waiting on this robot
AREXPORT void wakeAllWaitingThreads();
/// Wake up all threads waiting for connection
AREXPORT void wakeAllConnWaitingThreads();
/// Wake up all threads waiting for connection or connection failure
AREXPORT void wakeAllConnOrFailWaitingThreads();
/// Wake up all threads waiting for the run loop to exit
AREXPORT void wakeAllRunExitWaitingThreads();

/// Adds a user task to the list of synchronous taskes
AREXPORT bool addUserTask(const char *name, int position,
                          ArFunctor *functor,
                          ArTaskState::State *state = NULL);
/// Removes a user task from the list of synchronous taskes by name
AREXPORT void remUserTask(const char *name);
/// Removes a user task from the list of synchronous taskes by functor
AREXPORT void remUserTask(ArFunctor *functor);

/// Finds a user task by name
AREXPORT ArSyncTask *findUserTask(const char *name);
/// Finds a user task by functor
AREXPORT ArSyncTask *findUserTask(ArFunctor *functor);

/// Logs the list of user tasks, strictly for your viewing pleasure
AREXPORT void printUserTasks(void);
/// Logs the list of all tasks, strictly for your viewing pleasure
AREXPORT void printAllTasks(void);

/// Adds a task under the sensor interp part of the syncronous tasks
AREXPORT bool addSensorInterpTask(const char *name, int position,
                                  ArFunctor *functor,
                                  ArTaskState::State *state = NULL);
/// Removes a sensor interp tasks by name
AREXPORT void remSensorInterpTask(const char *name);
/// Removes a sensor interp tasks by functor
AREXPORT void remSensorInterpTask(ArFunctor *functor);

/// Finds a task by name
AREXPORT ArSyncTask *findTask(const char *name);
/// Finds a task by functor
AREXPORT ArSyncTask *findTask(ArFunctor *functor);
```

```
/// Adds an action to the list with the given priority
AREXPORT void addAction(ArAction *action, int priority);
/// Removes an action from the list, by pointer
AREXPORT bool remAction(ArAction *action);
/// Removes an action from the list, by name
AREXPORT bool remAction(const char *actionName);
/// Returns the first (highest priority) action with the given name (or NULL)
AREXPORT ArAction *findAction(const char *actionName);
/// Returns the map of actions... don't do this unless you really
/// know what you're doing
AREXPORT ArResolver::ActionMap *getActionMap(void);

/// Prints out the actions and their priorities
AREXPORT void printActions(void);

/// Gets the resolver the robot is using
AREXPORT ArResolver *getResolver(void);

/// Sets the resolver the robot is using
AREXPORT void setResolver(ArResolver *resolver);

/// Sets the encoderCorrectionCallback
AREXPORT void setEncoderCorrectionCallback(
        ArRetFunctor1<double, ArPoseWithTime> *functor);
/// Gets the encoderCorrectionCallback
AREXPORT ArRetFunctor1<double, ArPoseWithTime> *
    getEncoderCorrectionCallback(void);

// set up some of the internals of how the ArRobot class works
/// Sets the number of ms between cycles
AREXPORT void setCycleTime(unsigned int ms);
/// Gets the number of ms between cycles
AREXPORT unsigned int getCycleTime(void);
/// Sets whether to chain the robot cycle to when we get in SIP packets
void setCycleChained(bool cycleChained) { myCycleChained = cycleChained; }
/// Gets whether we chain the robot cycle to when we get in SIP packets
bool isCycleChained(void) { return myCycleChained; }
/// Sets the time without a response until connection assumed lost
AREXPORT void setConnectionTimeoutTime(int mSecs);
/// Gets the time without a response until connection assumed lost
AREXPORT int getConnectionTimeoutTime(void);
/// Gets the time the last packet was received
AREXPORT ArTime getLastPacketTime(void);

/// Sets the number of packets back in time the ArInterpolation goes
AREXPORT void setPoseInterpNumReadings(size_t numReadings)
  { myInterpolation.setNumberOfReadings(numReadings); }

/// Sets the number of packets back in time the position interpol goes
AREXPORT size_t getPoseInterpNumReadings(void)
  { return myInterpolation.getNumberOfReadings(); }

/// Gets the position the robot was at at the given timestamp
/** @see ArInterpolation::getPose
 */
AREXPORT int getPoseInterpPosition(ArTime timeStamp, ArPose *position)
  { return myInterpolation.getPose(timeStamp, position); }

/// Gets the Counter for the time through the loop
```

```
AREXPORT unsigned int getCounter(void) { return myCounter; }

/// Gets the parameters the robot is using
AREXPORT ArRobotParams *getRobotParams(void);

/// Loads a parameter file (replacing all other params)
AREXPORT bool loadParamFile(const char *file);

/// Attachs a key handler
AREXPORT void attachKeyHandler(ArKeyHandler *keyHandler,
                               bool exitOnEscape = true);
/// Gets the key handler attached to this robot
AREXPORT ArKeyHandler *getKeyHandler(void);

/// Lock the robot instance
AREXPORT int lock() {return(myMutex.lock());}
/// Try to lock the robot instance without blocking
AREXPORT int tryLock() {return(myMutex.tryLock());}
/// Unlock the robot instance
AREXPORT int unlock() {return(myMutex.unlock());}

/// This gets the root of the syncronous task tree, only serious
/// developers should use it
AREXPORT ArSyncTask *getSyncTaskRoot(void);

/// This function loops once...  only serious developers should use it
AREXPORT void loopOnce(void);

/// This is only for use by syncLoop
AREXPORT void incCounter(void) { myCounter++; }

/// Packet Handler, internal
AREXPORT void packetHandler(void);
/// Action Handler, internal
AREXPORT void actionHandler(void);
/// State Reflector, internal
AREXPORT void stateReflector(void);
/// Robot locker, internal
AREXPORT void robotLocker(void);
/// Robot unlocker, internal
AREXPORT void robotUnlocker(void);

/// For the key handler, escape calls this to exit, internal
AREXPORT void keyHandlerExit(void);

/// Processes a motor packet, internal
AREXPORT bool processMotorPacket(ArRobotPacket *packet);
/// Processes a new sonar reading, internal
AREXPORT void processNewSonar(char number, int range, ArTime timeReceived);
/// Processes a new encoder packet, internal
AREXPORT bool processEncoderPacket(ArRobotPacket *packet);

/// Internal function, shouldn't be used
AREXPORT void init(void);

/// Internal function, shouldn't be used, sets up the default sync list
AREXPORT void setUpSyncList(void);
/// Internal function, shouldn't be used, sets up the default packet handlers
AREXPORT void setUpPacketHandlers(void);
```

```
  ArRetFunctor1C<bool, ArRobot, ArRobotPacket *> myMotorPacketCB;
  ArRetFunctor1C<bool, ArRobot, ArRobotPacket *> myEncoderPacketCB;
  ArFunctorC<ArRobot> myPacketHandlerCB;
  ArFunctorC<ArRobot> myActionHandlerCB;
  ArFunctorC<ArRobot> myStateReflectorCB;
  ArFunctorC<ArRobot> myRobotLockerCB;
  ArFunctorC<ArRobot> myRobotUnlockerCB;
  ArFunctorC<ArRobot> myKeyHandlerExitCB;
  ArFunctorC<ArKeyHandler> *myKeyHandlerCB;

  // These four are internal... only people monkeying deeply should mess
  // with them, so they aren't documented... these process the cblists
  // and such
  /// Internal function, shouldn't be used, does a single run of connecting
  AREXPORT int asyncConnectHandler(bool tryHarderToConnect);

  /// Internal function, shouldn't be used, drops the conn because of error
  AREXPORT void dropConnection(void);
  /// Internal function, shouldn't be used, denotes the conn failed
  AREXPORT void failedConnect(void);
  /// Internal function, shouldn't be used, does the after conn stuff
  AREXPORT void madeConnection(void);
  /// Internal function, takes a packet and passes it to the packet handlers,
  /// returns true if handled, false otherwise
  AREXPORT bool handlePacket(ArRobotPacket *packet);
  /// Internal function, shouldn't be used, does what its name says
  AREXPORT std::list<ArFunctor *> * getRunExitListCopy();
  /// Internal function, processes a parameter file
  AREXPORT void processParamFile(ArRobotParamFile *paramFile);
protected:
  enum RotDesired {
    ROT_NONE,
    ROT_IGNORE,
    ROT_HEADING,
    ROT_VEL
  };
  enum TransDesired {
    TRANS_NONE,
    TRANS_IGNORE,
    TRANS_VEL,
    TRANS_VEL2,
    TRANS_DIST,
    TRANS_DIST_NEW
  };
  void reset(void);

  bool myFirstEncoderPose;
  ArTransform myEncoderTransform;

  ArMutex myMutex;
  ArSyncTask *mySyncTaskRoot;
  std::list<ArRetFunctor1<bool, ArRobotPacket *> *> myPacketHandlerList;

  ArSyncLoop mySyncLoop;

  std::list<ArFunctor *> myConnectCBList;
  std::list<ArFunctor *> myFailedConnectCBList;
  std::list<ArFunctor *> myDisconnectNormallyCBList;
```

```
std::list<ArFunctor *> myDisconnectOnErrorCBList;
std::list<ArFunctor *> myRunExitCBList;

ArRetFunctor1<double, ArPoseWithTime> *myEncoderCorrectionCB;
std::list<ArRangeDevice *> myRangeDeviceList;

ArCondition myConnectCond;
ArCondition myConnOrFailCond;
ArCondition myRunExitCond;

ArResolver::ActionMap myActions;
bool myOwnTheResolver;
ArResolver *myResolver;

std::map<int, ArSensorReading *> mySonars;
int myNumSonar;

unsigned int myCounter;
bool myIsConnected;

bool myBlockingConnectRun;
bool myAsyncConnectFlag;
int myAsyncConnectState;
int myAsyncConnectNoPacketCount;
int myAsyncConnectTimesTried;

bool myStateReflecting;
bool mySentPulse;

double myTransVal;
double myTransVal2;
int myLastTransVal;
int myLastTransVal2;
TransDesired myTransType;
TransDesired myLastTransType;
ArTime myTransSetTime;
ArTime myLastTransSent;
int myLastActionTransVal;
bool myActionTransSet;
ArPose myTransDistStart;
double myMoveDoneDist;

double myRotVal;
int myLastRotVal;
RotDesired myRotType;
RotDesired myLastRotType;
ArTime myRotSetTime;
ArTime myLastRotSent;
int myLastActionRotVal;
bool myLastActionRotStopped;
bool myActionRotSet;
double myHeadingDoneDiff;

ArTime myLastPulseSent;

int myDirectPrecedenceTime;

int myStateReflectionRefreshTime;
```

```
  ArActionDesired myActionDesired;

  std::string myName;
  std::string myRobotName;
  std::string myRobotType;
  std::string myRobotSubType;

  double myMaxTransVel;
  double myMaxRotVel;

  double myActionMaxTransVel;
  double myActionMaxNegTransVel;
  double myActionMaxRotVel;

  double myLastSentMaxTransVel;
  double myLastSentMaxRotVel;

  ArDeviceConnection *myConn;

  ArRobotPacketSender mySender;
  ArRobotPacketReceiver myReceiver;

  ArRobotParams myParams;

  ArInterpolation myInterpolation;

  ArKeyHandler *myKeyHandler;

  // variables for tracking the data stream
  time_t myTimeLastMotorPacket;
  int myMotorPacCurrentCount;
  int myMotorPacCount;
  time_t myTimeLastSonarPacket;
  int mySonarPacCurrentCount;
  int mySonarPacCount;
  unsigned int myCycleTime;
  bool myCycleChained;
  ArTime myLastPacketReceivedTime;
  int myTimeoutTime;

  // all the state reflecing variables
  ArPoseWithTime myRawEncoderPose;
  ArPoseWithTime myEncoderPose;
  ArTime myEncoderPoseTaken;
  ArPose myGlobalPose;
  ArTransform myEncoderGlobalTrans;
  double myLeftVel;
  double myRightVel;
  double myBatteryVoltage;
  int myStallValue;
  double myControl;
  int myFlags;
  double myCompass;
  int myAnalogPortSelected;
  unsigned char myAnalog;
  unsigned char myDigIn;
  unsigned char myDigOut;
  double myVel;
  double myRotVel;
```

```
  int myLastX;
  int myLastY;
  int myLastTh;

};

#endif // ARROBOT_H
```

Andiamo a spiegare approfonditamente le funzioni da noi sviluppate:

- int sfClosestSonarNumber(float startAngle, float endAngle): questa funzione restituisce un numero intero che indica il sonar avente la misura della distanza minore da un ostacolo nel range indicato.
  - Ø startAngle: variabile di tipo float che indica l'angolo iniziale, in gradi, del range di misura
  - Ø endAngle: variabile di tipo float che indica l'angolo finale, in gradi, del range di misura

  Ad esempio eseguendo la chiamata sfClosestSonarNumber(0, 359) otteniamo il numero del sonar più vicino ad un ostacolo tra tutti quelli presenti.
- int sfClosestSonarRange(float startAngle,float endAngle): questa funzione restituisce un numero intero che indica la distanza minore da un ostacolo nel range indicato.
  - Ø startAngle: variabile di tipo float che indica l'angolo iniziale, in gradi, del range di misura
  - Ø endAngle: variabile di tipo float che indica l'angolo finale, in gradi, del range di misura

  Ad esempio eseguendo la chiamata sfClosestSonarRange(0, 359) otteniamo il valore della distanza minore da un ostacolo.
- int sfSonarNum(): questa funzione restituisce un numero intero che indica il numero totale dei sonar presenti sul robot collegato.

  Questo numero è solitamente, per quanto riguarda la nostra esperienza, sette (7).
- int sfFarthestSonarNumber(): questa funzione restituisce un numero intero che indica il sonar avente la misura della distanza massima da un ostacolo.
- int sfFarthestSonarRange(): questa funzione restituisce un numero intero che indica la distanza massima da un ostacolo.
- int sfSonarNew(int num_sonar): questa funzione restituisce un numero intero che indica se il sonar 'num_sonar' ha eseguito una nuova lettura (restituendo 1) o meno (restituendo 0).
  - Ø num_sonar: variabile di tipo integer che indica il sonar interessato
- int sfSonarRange(int num_sonar): questa funzione restituisce un numero intero che indica la distanza da un ostacolo del sonar indicato da 'num_sonar'.
  - Ø num_sonar: variabile di tipo integer che indica il sonar interessato

- float sfSonarXCoord(int num_sonar): questa funzione restituisce un numero di tipo floating point che indica la distanza da un ostacolo, lungo l'asse X, del sonar indicato da 'num_sonar'.
  - Ø num_sonar: variabile di tipo integer che indica il sonar interessato
- float sfSonarYCoord(int num_sonar): questa funzione restituisce un numero di tipo floating point che indica la distanza da un ostacolo, lungo l'asse Y, del sonar indicato da 'num_sonar'.
  - Ø num_sonar: variabile di tipo integer che indica il sonar interessato
- float sfCos(float angle): questa funzione restituisce un numero di tipo floating point che indica il coseno dell'angolo indicato della variabile 'angle'.
  - Ø angle: variabile di tipo floating point che indica, in gradi, l'angolo su cui si vuole eseguire l'operazione trigonometrica
- float sfSin(float angle): questa funzione restituisce un numero di tipo floating point che indica il seno dell'angolo indicato della variabile 'angle'.
  - Ø angle: variabile di tipo floating point che indica, in gradi, l'angolo su cui si vuole eseguire l'operazione trigonometrica
- float sfSqrt(float argoment): questa funzione restituisce un numero di tipo floating point che indica la radice quadrata della variabile 'argoment'.
- argoment: variabile di tipo floating point che indica il numero su cui si vuole eseguire l'operazione
- float sfTan(float angle): questa funzione restituisce un numero di tipo floating point che indica la tangente dell'angolo indicato della variabile 'angle'.
  - Ø angle: variabile di tipo floating point che indica, in gradi, l'angolo su cui si vuole eseguire l'operazione trigonometrica

4. Verifica lavoro svolto

Parte fondamentale dell'elaborato è stata una attenta verifica del lavoro svolto sia dal punto di vista della correttezza che da quello del rispetto dei requisiti iniziali. Per quanto riguarda quest'ultima parte, dopo una fase di discussione, ci appare chiaro che l'elaborato ha raggiunto gli scopi prefissati e più volte esposti, anche se un'ultima valutazione è lasciata al lettore.

Passando alla correttezza del lavoro, è stata eseguita una approfondita fase di verifica di tutto ciò che è stato implementato, sia per quanto riguarda le modalità di creazione di nuove librerie che per la nuova 'betman.so'.

Il controllo sulle modalità è stata una parte implicita del compito poiché non era possibile passare all'implementazione pratica di una libreria senza trovare i passi necessari all'operazione.

Per quanto riguarda le nuove funzionalità, esse sono state testate soprattutto attraverso il programma di simulazione a disposizione, sia da noi che dal tesista Trevisson.

Abbiamo ritenuto opportuno eseguire operazioni di raffronto tra i risultati ottenuti e quelli attesi seguendo lo stato del robot simulato ed inoltre sono state implementate activity ad hoc per la verifica.

In particolare, l'activity 'perimetro' ha lo scopo di testare le funzioni 'sfFarthestSonarNumber' e 'sfClosestSonarRange', riportiamo qui di seguito il suo listato:

```
loadlib betman; // load betman.so library
act perimetro()
{
 // data declaration and initialization
 int greater; // number of sonar with the greater range
 int a;int i;
 a=1; i=0;
 speed(100);
 while(a)
 {
  // if lowest Range is less than 500 mm stop robot
   if(sfClosestSonarRange(0,359)<500)
   {
     stop;
     a=0;
   }
   else
   {
     greater=sfFarthestSonarNumber();
     if (greater==0)
       turn(90); // if greater=0 turn 90 degrees
     else
     if (greater==1)
       turn(30); // if greater=0 turn 30 degrees
     else
       if (greater==2)
         turn(15); // if greater=0 turn 15 degrees
       else
         if (greater==4)
           turn(-15); // if greater=0 turn -15 degrees
         else
           if (greater==5)
             turn(-30); // if greater=5 turn -30 degrees
           else
             if (greater==6)
               turn(-90); // if greater=6 turn -90 degrees
   }
 }
}
```

In pratica questa activity ha l'obbiettivo di mantenere il robot il più distante possibile da ogni ostacolo mentre si muove a velocità costante. Infatti il robot, attraverso 'sfFarthestSonarNumber', acquisisce il numero del sonar avente la distanza maggiore da una barriera e si dirige nella sua direzione facendo attenzione che non venga violata la distanza di sicurezza di 500 mm, nel qual caso si ferma e l'activity termina.

Possiamo quindi ritenere esaurienti le prove eseguite su ciò che è stato sviluppato che nel complesso appare corretto e confacente agli obbiettivi posti inizialmente.

5. Modalità operative

5.1. Componenti necessari

Per quanto riguarda l'installazione del client Saphira sotto sistemi Linux, si rendono necessari due file denominati 'Saphira-8.1-0.i386.rpm' e 'ARIA-1.1-0.i386.rpm' reperibili sul sito della Pioneer (per scaricare questi due file d'installazione è necessario essere in possesso di un opportuno nome utente e relativa password).

5.2. Modalità di installazione

Una volta scaricati sul proprio calcolatore possono essere lanciati attraverso i seguenti comandi:

rpm –i –nodeps Saphira-8.1-0.i386.rpm

rpm –i ARIA-1.1-0.i386.rpm

Eseguiti questi semplici passi d'installazione bisogna andare a modificare il file 'saphira.sh' presente nella directory '/etc/profile.d', esso contiene informazioni riguardanti le variabili d'ambiente di Saphira. Riportiamo di seguito il contenuto del file con breve spiegazione per ogni termine usato:

- SAPHIRA="/usr/local/Saphira": *indica la directory principale di Saphira*
- SAPHIRA_LOAD="$SAPHIRA/colbert": *indica dove risiede il file 'startup.act', necessario al Colbert quando si avvia Saphira*
- #SAPHIRA_LOAD="$HOME":
- SAPHIRA_COMPIPE="$HOME/robot": *indica dove si trova il socket tcp*
- ARIA="/urs/local/Aria": *indica la directory principale di Aria*

Definizione variabili d'ambiente:

- #LD_LIBRARY_PATH= "$SAPHIRA/handler/obj":$LD_LIBRARY_PATH
- LD_LIBRARY_PATH="$SAPHIRA/lib":"$ARIA/lib":$LD_LIBRARY_PATH
- PATH="$SAPHIRA/bin":$PATH
- ExportSAPHIRALD_LIBRARY_PATHSAPHIRA_LOADSAPHIRA_COMPIPE ARIAPATH
- aliaspioneer="pioneer-rpion1m": *settaggio particolare del robot*
- aliasssaphira="pioneer & saphira &": *per lanciare Saphira ed il simulatore*

Eseguite queste brevi e semplici istruzioni, il client Saphira è pronto per essere utilizzato. Il software Saphira 8.x viene installato nella directory '/usr/local/Saphira', mentre le librerie Aria vengono installate nella directory '/usr/local/Aria'. Una breve avvertenza che anticipiamo riguarda le librerie Aria, le quali vengono inserite anche nella directory '/usr/local/Saphira/ohandler/Aria' per agevolare la programmazione.

5.3. Avvertenze

Una volta eseguite tutte le procedure d'installazione abbiamo riscontrato qualche problema di carattere pratico, in quanto alcuni pezzi di codice contenevano degli errori e altri delle imprecisioni; abbiamo dovuto intervenire quindi modificando il codice errato. A questo proposito sono stati di grande aiuto anche il tesista Trevisson e Carlo Mor (esperto di programmazione e di sistemi UNIX).

I file che modificati sono 'Enumerations.h' e 'SfSystem.h', questi due file sostanzialmente risultavano errati in quanto contenevano il carattere '\' come fine riga non riconosciuto dal compilatore da noi utilizzato; riportiamo di seguito il listato corretto:

'Enumerations.h'

```
// "$Id: Enumerations.H,v 1.6 2002/02/22 20:16:58 konolige Exp $"
//
// Enumerations for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2001 by Bill Spitzak and others.
//
// This library is free software; you can redistribute it and/or
// modify it under the terms of the GNU Library General Public
// License as published by the Free Software Foundation; either
// version 2 of the License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Library General Public License for more details.
//
// You should have received a copy of the GNU Library General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA.
//
// Please report all bugs and problems to "fltk-bugs@fltk.org".
//

#ifndef Fl_Enumerations_H
#define Fl_Enumerations_H

#include "Fl_Export.H"


//
// The FLTK version number; this is changed slightly from the beta versions
// because the old "const double" definition would not allow for conditional
// compilation...
//
// FL_VERSION is a double that describes the major and minor version numbers.
// Version 1.1 is actually stored as 1.01 to allow for more than 9 minor
// releases.
//
// The FL_MAJOR_VERSION, FL_MINOR_VERSION, and FL_PATCH_VERSION constants
// give the integral values for the major, minor, and patch releases
// respectively.
```

```
//

#define FL_MAJOR_VERSION    1
#define FL_MINOR_VERSION    1
#define FL_PATCH_VERSION    0
#define FL_VERSION              ((double)FL_MAJOR_VERSION +
(double)FL_MINOR_VERSION * 0.01 + (double)FL_PATCH_VERSION * 0.0001)
//                               (double)FL_MINOR_VERSION * 0.01 + \
//                               (double)FL_PATCH_VERSION * 0.0001)

typedef unsigned char uchar;
typedef unsigned long ulong;

enum Fl_Event {  // events
  FL_NO_EVENT                   = 0,
  FL_PUSH               = 1,
  FL_RELEASE            = 2,
  FL_ENTER              = 3,
  FL_LEAVE              = 4,
  FL_DRAG               = 5,
  FL_FOCUS              = 6,
  FL_UNFOCUS            = 7,
  FL_KEYDOWN                    = 8,
  FL_KEYUP              = 9,
  FL_CLOSE              = 10,
  FL_MOVE               = 11,
  FL_SHORTCUT                   = 12,
  FL_DEACTIVATE                 = 13,
  FL_ACTIVATE           = 14,
  FL_HIDE               = 15,
  FL_SHOW               = 16,
  FL_PASTE              = 17,
  FL_SELECTIONCLEAR             = 18,
  FL_MOUSEWHEEL                 = 19
};
#define FL_KEYBOARD FL_KEYDOWN

enum Fl_When { // Fl_Widget::when():
  FL_WHEN_NEVER                 = 0,
  FL_WHEN_CHANGED = 1,
  FL_WHEN_RELEASE   = 4,
  FL_WHEN_RELEASE_ALWAYS= 6,
  FL_WHEN_ENTER_KEY         = 8,
  FL_WHEN_ENTER_KEY_ALWAYS=10,
  FL_WHEN_ENTER_KEY_CHANGED=11,
  FL_WHEN_NOT_CHANGED     = 2 // modifier bit to disable changed() test
};

// Fl::event_key() and Fl::get_key(n) (use ascii letters for all other keys):
#define FL_Button           0xfee8 // use Fl_Button+FL_*_MOUSE
#define FL_BackSpace    0xff08
#define FL_Tab              0xff09
#define FL_Enter 0xff0d
#define FL_Pause            0xff13
#define FL_Scroll_Lock  0xff14
#define FL_Escape           0xff1b
#define FL_Home                     0xff50
#define FL_Left             0xff51
#define FL_Up               0xff52
```

```
#define FL_Right 0xff53
#define FL_Down 0xff54
#define FL_Page_Up 0xff55
#define FL_Page_Down 0xff56
#define FL_End 0xff57
#define FL_Print 0xff61
#define FL_Insert 0xff63
#define FL_Menu 0xff67 // the "menu/apps" key on XFree86
#define FL_Num_Lock 0xff7f
#define FL_KP 0xff80 // use FL_KP+'x' for 'x' on numeric keypad
#define FL_KP_Enter 0xff8d // same as Fl_KP+'\r'
#define FL_KP_Last 0xffbd // use to range-check keypad
#define FL_F 0xffbd // use FL_F+n for function key n
#define FL_F_Last 0xffe0 // use to range-check function keys
#define FL_Shift_L 0xffe1
#define FL_Shift_R 0xffe2
#define FL_Control_L 0xffe3
#define FL_Control_R 0xffe4
#define FL_Caps_Lock 0xffe5
#define FL_Meta_L 0xffe7 // the left MSWindows key on XFree86
#define FL_Meta_R 0xffe8 // the right MSWindows key on XFree86
#define FL_Alt_L 0xffe9
#define FL_Alt_R 0xffea
#define FL_Delete 0xffff

// Fl::event_button():
#define FL_LEFT_MOUSE 1
#define FL_MIDDLE_MOUSE 2
#define FL_RIGHT_MOUSE 3

// Fl::event_state():
#define FL_SHIFT 0x00010000
#define FL_CAPS_LOCK 0x00020000
#define FL_CTRL 0x00040000
#define FL_ALT 0x00080000
#define FL_NUM_LOCK 0x00100000 // most X servers do this?
#define FL_META 0x00400000 // correct for XFree86
#define FL_SCROLL_LOCK 0x00800000 // correct for XFree86
#define FL_BUTTON1 0x01000000
#define FL_BUTTON2 0x02000000
#define FL_BUTTON3 0x04000000

enum Fl_Boxtype { // boxtypes (if you change these you must fix fl_boxtype.C):
  FL_NO_BOX = 0, FL_FLAT_BOX,

  FL_UP_BOX, FL_DOWN_BOX,
  FL_UP_FRAME, FL_DOWN_FRAME,
  FL_THIN_UP_BOX, FL_THIN_DOWN_BOX,
  FL_THIN_UP_FRAME, FL_THIN_DOWN_FRAME,
  FL_ENGRAVED_BOX, FL_EMBOSSED_BOX,
  FL_ENGRAVED_FRAME, FL_EMBOSSED_FRAME,
  FL_BORDER_BOX, _FL_SHADOW_BOX,
  FL_BORDER_FRAME, _FL_SHADOW_FRAME,
  _FL_ROUNDED_BOX, _FL_RSHADOW_BOX,
  _FL_ROUNDED_FRAME, _FL_RFLAT_BOX,
  _FL_ROUND_UP_BOX, _FL_ROUND_DOWN_BOX,
  _FL_DIAMOND_UP_BOX, _FL_DIAMOND_DOWN_BOX,
  _FL_OVAL_BOX, _FL_OSHADOW_BOX,
  _FL_OVAL_FRAME, _FL_OFLAT_BOX,
```

```
  _FL_PLASTIC_UP_BOX,           _FL_PLASTIC_DOWN_BOX,
  _FL_PLASTIC_UP_FRAME,       _FL_PLASTIC_DOWN_FRAME,
  FL_FREE_BOXTYPE
};
extern FL_EXPORT Fl_Boxtype define_FL_ROUND_UP_BOX();
#define FL_ROUND_UP_BOX define_FL_ROUND_UP_BOX()
#define FL_ROUND_DOWN_BOX (Fl_Boxtype)(define_FL_ROUND_UP_BOX()+1)
extern FL_EXPORT Fl_Boxtype define_FL_SHADOW_BOX();
#define FL_SHADOW_BOX define_FL_SHADOW_BOX()
#define FL_SHADOW_FRAME (Fl_Boxtype)(define_FL_SHADOW_BOX()+2)
extern FL_EXPORT Fl_Boxtype define_FL_ROUNDED_BOX();
#define FL_ROUNDED_BOX define_FL_ROUNDED_BOX()
#define FL_ROUNDED_FRAME (Fl_Boxtype)(define_FL_ROUNDED_BOX()+2)
extern FL_EXPORT Fl_Boxtype define_FL_RFLAT_BOX();
#define FL_RFLAT_BOX define_FL_RFLAT_BOX()
extern FL_EXPORT Fl_Boxtype define_FL_RSHADOW_BOX();
#define FL_RSHADOW_BOX define_FL_RSHADOW_BOX()
extern FL_EXPORT Fl_Boxtype define_FL_DIAMOND_BOX();
#define FL_DIAMOND_UP_BOX define_FL_DIAMOND_BOX()
#define FL_DIAMOND_DOWN_BOX (Fl_Boxtype)(define_FL_DIAMOND_BOX()+1)
extern FL_EXPORT Fl_Boxtype define_FL_OVAL_BOX();
#define FL_OVAL_BOX define_FL_OVAL_BOX()
#define FL_OSHADOW_BOX (Fl_Boxtype)(define_FL_OVAL_BOX()+1)
#define FL_OVAL_FRAME (Fl_Boxtype)(define_FL_OVAL_BOX()+2)
#define FL_OFLAT_BOX (Fl_Boxtype)(define_FL_OVAL_BOX()+3)

extern FL_EXPORT Fl_Boxtype define_FL_PLASTIC_UP_BOX();
#define FL_PLASTIC_UP_BOX define_FL_PLASTIC_UP_BOX()
#define FL_PLASTIC_DOWN_BOX (Fl_Boxtype)(define_FL_PLASTIC_UP_BOX()+1)
#define FL_PLASTIC_UP_FRAME (Fl_Boxtype)(define_FL_PLASTIC_UP_BOX()+2)
#define FL_PLASTIC_DOWN_FRAME (Fl_Boxtype)(define_FL_PLASTIC_UP_BOX()+3)

// conversions of box types to other boxtypes:
inline Fl_Boxtype down(Fl_Boxtype b) {return (Fl_Boxtype)(b|1);}
inline Fl_Boxtype frame(Fl_Boxtype b) {return (Fl_Boxtype)(b|2);}

// back-compatability box types:
#define FL_FRAME FL_ENGRAVED_FRAME
#define FL_FRAME_BOX FL_ENGRAVED_BOX
#define FL_CIRCLE_BOX FL_ROUND_DOWN_BOX
#define FL_DIAMOND_BOX FL_DIAMOND_DOWN_BOX

enum Fl_Labeltype {         // labeltypes:
  FL_NORMAL_LABEL   = 0,
  FL_NO_LABEL,
  _FL_SHADOW_LABEL,
  _FL_ENGRAVED_LABEL,
  _FL_EMBOSSED_LABEL,
  _FL_MULTI_LABEL,
  _FL_ICON_LABEL,

  FL_FREE_LABELTYPE
};
#define FL_SYMBOL_LABEL FL_NORMAL_LABEL
extern Fl_Labeltype FL_EXPORT define_FL_SHADOW_LABEL();
#define FL_SHADOW_LABEL define_FL_SHADOW_LABEL()
extern Fl_Labeltype FL_EXPORT define_FL_ENGRAVED_LABEL();
#define FL_ENGRAVED_LABEL define_FL_ENGRAVED_LABEL()
extern Fl_Labeltype FL_EXPORT define_FL_EMBOSSED_LABEL();
```

```
#define FL_EMBOSSED_LABEL define_FL_EMBOSSED_LABEL()

enum Fl_Align {  // align() values
  FL_ALIGN_CENTER               = 0,
  FL_ALIGN_TOP                          = 1,
  FL_ALIGN_BOTTOM               = 2,
  FL_ALIGN_LEFT                         = 4,
  FL_ALIGN_RIGHT                = 8,
  FL_ALIGN_INSIDE               = 16,
  FL_ALIGN_TEXT_OVER_IMAGE      = 32,
  FL_ALIGN_IMAGE_OVER_TEXT      = 0,
  FL_ALIGN_CLIP                         = 64,
  FL_ALIGN_WRAP                         = 128,
  FL_ALIGN_TOP_LEFT             = FL_ALIGN_TOP | FL_ALIGN_LEFT,
  FL_ALIGN_TOP_RIGHT                    = FL_ALIGN_TOP | FL_ALIGN_RIGHT,
  FL_ALIGN_BOTTOM_LEFT                  = FL_ALIGN_BOTTOM | FL_ALIGN_LEFT,
  FL_ALIGN_BOTTOM_RIGHT                 = FL_ALIGN_BOTTOM | FL_ALIGN_RIGHT,
  FL_ALIGN_LEFT_TOP             = FL_ALIGN_TOP_LEFT,
  FL_ALIGN_RIGHT_TOP                    = FL_ALIGN_TOP_RIGHT,
  FL_ALIGN_LEFT_BOTTOM                  = FL_ALIGN_BOTTOM_LEFT,
  FL_ALIGN_RIGHT_BOTTOM                 = FL_ALIGN_BOTTOM_RIGHT,
  FL_ALIGN_NOWRAP               = 0 // for back compatability
};

enum Fl_Font {    // standard fonts
  FL_HELVETICA                  = 0,
  FL_HELVETICA_BOLD,
  FL_HELVETICA_ITALIC,
  FL_HELVETICA_BOLD_ITALIC,
  FL_COURIER,
  FL_COURIER_BOLD,
  FL_COURIER_ITALIC,
  FL_COURIER_BOLD_ITALIC,
  FL_TIMES,
  FL_TIMES_BOLD,
  FL_TIMES_ITALIC,
  FL_TIMES_BOLD_ITALIC,
  FL_SYMBOL,
  FL_SCREEN,
  FL_SCREEN_BOLD,
  FL_ZAPF_DINGBATS,

  FL_FREE_FONT                  = 16,     // first one to allocate
  FL_BOLD               = 1,      // add this to helvetica, courier, or times
  FL_ITALIC             = 2       // add this to helvetica, courier, or times
};

extern FL_EXPORT int FL_NORMAL_SIZE;

enum Fl_Color {  // standard colors
  FL_BLACK              = 0,
  FL_RED                = 1,
  FL_GREEN              = 2,
  FL_YELLOW             = 3,
  FL_BLUE               = 4,
  FL_MAGENTA            = 5,
  FL_CYAN               = 6,
  FL_WHITE              = 7,
  FL_INACTIVE_COLOR= 8,
```

```
  FL_SELECTION_COLOR        = 15,

  FL_FREE_COLOR             = 16,
  FL_NUM_FREE_COLOR         = 16,

  FL_GRAY_RAMP              = 32,

  // boxtypes limit themselves to these colors so whole ramp is not allocated:
  FL_GRAY0          = 32,   // 'A'
  FL_DARK3          = 39,   // 'H'
  FL_DARK2          = 45,   // 'N'
  FL_DARK1          = 47,   // 'P'
  FL_GRAY           = 49,   // 'R' default color
  FL_LIGHT1         = 50,   // 'S'
  FL_LIGHT2         = 52,   // 'U'
  FL_LIGHT3         = 54,   // 'W'

  FL_COLOR_CUBE             = 56
};

FL_EXPORT Fl_Color fl_inactive(Fl_Color c);
FL_EXPORT Fl_Color fl_contrast(Fl_Color fg, Fl_Color bg);
FL_EXPORT Fl_Color fl_color_average(Fl_Color c1, Fl_Color c2, float weight);
inline Fl_Color fl_lighter(Fl_Color c) { return fl_color_average(c, FL_WHITE, .67f); }
inline Fl_Color fl_darker(Fl_Color c) { return fl_color_average(c, FL_BLACK, .67f); }
inline Fl_Color fl_rgb_color(uchar r, uchar g, uchar b) {
  return (Fl_Color)(((((r << 8) | g) << 8) | b) << 8);
}
#define FL_NUM_GRAY 24
inline Fl_Color fl_gray_ramp(int i) {return (Fl_Color)(i+FL_GRAY_RAMP);}
#define FL_NUM_RED    5
#define FL_NUM_GREEN        8
#define FL_NUM_BLUE   5
inline Fl_Color fl_color_cube(int r, int g, int b) {
  return (Fl_Color)((b*FL_NUM_RED + r) * FL_NUM_GREEN + g + FL_COLOR_CUBE);}

enum Fl_Cursor {        // standard cursors
  FL_CURSOR_DEFAULT         = 0,
  FL_CURSOR_ARROW   = 35,
  FL_CURSOR_CROSS   = 66,
  FL_CURSOR_WAIT    = 76,
  FL_CURSOR_INSERT  = 77,
  FL_CURSOR_HAND    = 31,
  FL_CURSOR_HELP    = 47,
  FL_CURSOR_MOVE    = 27,
  // fltk provides bitmaps for these:
  FL_CURSOR_NS              = 78,
  FL_CURSOR_WE              = 79,
  FL_CURSOR_NWSE    = 80,
  FL_CURSOR_NESW    = 81,
  FL_CURSOR_NONE    = 255,
  // for back compatability (non MSWindows ones):
  FL_CURSOR_N               = 70,
  FL_CURSOR_NE              = 69,
  FL_CURSOR_E               = 49,
  FL_CURSOR_SE              = 8,
  FL_CURSOR_S               = 9,
  FL_CURSOR_SW              = 7,
  FL_CURSOR_W               = 36,
```

```
  FL_CURSOR_NW                    = 68
  //FL_CURSOR_NS        = 22,
  //FL_CURSOR_WE        = 55,
};

enum { // values for "when" passed to Fl::add_fd()
  FL_READ = 1,
  FL_WRITE = 4,
  FL_EXCEPT = 8
};

enum Fl_Mode { // visual types and Fl_Gl_Window::mode() (values match Glut)
  FL_RGB          = 0,
  FL_INDEX        = 1,
  FL_SINGLE       = 0,
  FL_DOUBLE       = 2,
  FL_ACCUM        = 4,
  FL_ALPHA        = 8,
  FL_DEPTH        = 16,
  FL_STENCIL      = 32,
  FL_RGB8         = 64,
  FL_MULTISAMPLE= 128
};

// damage masks

enum Fl_Damage {
  FL_DAMAGE_CHILD    = 0x01,
  FL_DAMAGE_EXPOSE   = 0x02,
  FL_DAMAGE_SCROLL   = 0x04,
  FL_DAMAGE_OVERLAY  = 0x08,
  FL_DAMAGE_ALL      = 0x80
};

#endif

//
// End of "$Id: Enumerations.H,v 1.6 2002/02/22 20:16:58 konolige Exp $".
//
```

'SfSystem.h'

```
//
// SfSystem.h
//
// Saphira system definitions

//
// Copyright 2001 by Kurt Konolige
//
// The author hereby grants to SRI permission to use this software.
// The author also grants to SRI permission to distribute this software
// to schools for non-commercial educational use only.
//
// The author hereby grants to other individuals or organizations
// permission to use this software for non-commercial
// educational use only.  This software may not be distributed to others
// except by SRI, under the conditions above.
//
// Other than these cases, no part of this software may be used or
// distributed without written permission of the author.
```

```
//
// Neither the author nor SRI make any representations about the
// suitability of this software for any purpose.  It is provided
// "as is" without express or implied warranty.
//
// Kurt Konolige
// Senior Computer Scientist
// SRI International
// 333 Ravenswood Avenue
// Menlo Park, CA 94025
// E-mail:  konolige@ai.sri.com


#ifndef SFSYSTEM_H
#define SFSYSTEM_H

#undef SFEXPORT
#if defined(MS_WINDOWS) && !defined(SF_STATIC)
#ifdef MAKE_LIBRARY
#define SFEXPORT _declspec( dllexport )
#else
#define SFEXPORT _declspec( dllimport )
#endif
#else
#define SFEXPORT
#endif


/// Saphira system definition
/**
   The Saphira system class is a static class that holds basic information
   about the single robot server for which Saphira is the client.  It has
   useful functions to get the robot object, device buffers, artifact list,
   connections to the robot server, and so on.

   On startup of the standard Saphira client, all the items in the Sf class
   are initialized by calling init().
*/

class Sf
{
  Sf() {}; // no constructor!

public:


  /// initializes Saphira system, including Aria; called by the standard Saphira client
  /// on startup
  static SFEXPORT void init();
  /// Current robot object; Saphira 8.0 has only one
  /// User programs can access this with the SfROBOT macro
  static SfRobot *robot() { return ourRobot; };
  /// sonar device object
  static SfSonarDevice *sonar() { return ourSonar; }
  /// laser device object
  static SfLaserDevice *laser() { return ourLaser; }
  /// artifact list
  static SFEXPORT SfArtifactList *artList;     // artifact list
  /// main Colbert stream for reading/writing Colbert commands
```

```
  static SFEXPORT SfColbertStream *ourColbert; // console Colbert stream

  // values for the current robot

  /// Gets the robot current X value, in mm
  static double getX() { return ourRobot->getX(); }; // in mm
  /// Gets the robot current Y value, in mm
  static double getY() { return ourRobot->getY(); }; // in mm;
  /// Gets the robot current heading, in degrees
  static double getTh() { return ourRobot->getTh(); }; // in deg
  /// Gets the robot current pose object
  static ArPose getRwPose() { return ourRobot->getPose(); }; // robot's pose

  // connections
  static SFEXPORT ArTcpConnection *tcpCon;
  static SFEXPORT ArSerialConnection *serialCon;

  /// display frame; user programs can acces this with the SfFRAME macro
  static SFEXPORT SfFr *frame;
  static SFEXPORT bool isExited;                // true when we have an exit request

  // internal, use the accessor... laser device, currently just a single one
  static SFEXPORT SfLaserDevice *ourLaser;
private:
  // sonar device, most robots have this...
  static SFEXPORT SfSonarDevice *ourSonar;

  // returns current robot instance
  static SFEXPORT SfRobot *ourRobot;

  // parsing stream handler
  static void parser();
};

//
// Global macros
//

/// Standard way to access the current robot
#define SfROBOT Sf::robot()

#define SfSELFTASK  SfActRegister::currentTask


//
// Helper macro for User and Sensor tasks
//


// use these in a constructor or any other member fn for the object
// NAME is a string, double quotes

#define ADDUSERTASK(NAME,PRIORITY,CLASS,PROC) Sf::robot()->addUserTask(NAME,
PRIORITY, new ArFunctorC<CLASS>(this, &CLASS::PROC));
//  Sf::robot()->addUserTask(NAME, PRIORITY,   \
//   new ArFunctorC<CLASS>(this, &CLASS::PROC));

#define ADDSENSORTASK(NAME,PRIORITY,CLASS,PROC) Sf::robot()-
>addSensorInterpTask(NAME, PRIORITY,new ArFunctorC<CLASS>(this, &CLASS::PROC));
```

```
//  Sf::robot()->addSensorInterpTask(NAME, PRIORITY,   \
//   new ArFunctorC<CLASS>(this, &CLASS::PROC));


//
// Various connection globals
//
SFEXPORT void sfSetSerialBaud(int rate);
SFEXPORT void sfSetSerialPort(char *dev);


#endif // SFSYSTEM_H
```

Infine è stato modificato il file ‘Saphira.h’ presente nella directory ‘/usr/local/Saphira/ohandler/include’, poiché contenva un include errato alle librerie Aria che dava luogo a diversi errori (tra cui l’impossibilità di compilare nuove librerie). Alla riga 59 era infatti presente l’istruzione:

#include "../Aria/Aria.h"

che deve essere modificata in:

#include "/usr/local/Aria/include/Aria.h"

6. Conclusioni e sviluppi futuri

Attraverso il lavoro svolto è stata esposta in modo approfondito la modalità di creazione di nuove librerie per Saphira 8.x, quindi ne è stata implementata una, di nome ‘betman.so’, che forrnisce funzioni per il controllo di dispositivi sonar, presenti nella versione 6.2, ed altre utility.

Per quanto riguarda gli sviluppi futuri, attraverso la documentazione, è possibile estendere in quantità notevole le operazioni a disposizione dell’utente con notevole facilità, rendendo più semplice ed intuitiva l’interazione tra la macchina e l’utente stesso.

Bibliografia

[1] “Saphira Tutorial”, Compiling, Loading and Debugging C++. Files: Unix Systems Software version 8.0 (Saphira/Aria) September 2001

[2] Doxygen 1.2.12: “Aria Reference Manual 1.1.x”, Tue Mar 12 17:12:11 2002

[3] “Saphira Software Manual”, Saphira Version 8.0a (Saphira/Aria integration)

[4] Doxygen 1.2.12: “Saphira Reference Manual 8.1.x”, Tue Mar 12 17:12:45 2002

[5] “Colbert User Manual”, Software version 8.0a (Saphira/Aria), September 2001

[6] Kurt Konolige: “COLBERT: A Language for Reactive Control in Saphira”, SRI International, 333 Ravenswood Avenue Menlo Park, CA 94025

Indice