AIBO Programming Tutorial

REVISON 1

TABLE OF CONTENTS

1. Introduction ………………………………………………………......... 2
2. Basic OPENR Program Constructs ………………………..………… 3
3. Building ……………………………………………………................... 10
4. Running ……………………………………………………................... 12
5. Debugging …………………………………………………................. 12
6. Flow of Development ………………………………………….…... 13
7. Sample Program 1 – HelloWorld ..……………………………....... 14
8. Sample Program 2 – Walking Dog ……………………………….…….. 15
9. Sample Program 3 – Inter-dog communication ………………........ 15
10. Help Information ………………………………………………….……16
11. Bibliography …………………………………………………….……16

© 2002, Borg Lab, Georgia Institute Of Technology

Introduction

The Environment

The Sony AIBO robots run a Real time object oriented OS called Aperios. Programs for this environment can be written using the OPENR-SDK available from Sony's website at www.jp.aibo.com/openr/ (there is a / at the end).

The environment is installed under */usr/OPENR_SDK* (as opposed to */usr/local/OPENR_SDK which is the default environment for installation).*

Documentation

The AIBO Documentation can be found at */usr/OPENR_SDK/documentation*. The Documentation consists of the following files in PDF format.

a. The AIBO Programming manual

The AIBO programming manual can be downloaded from Sony's website. This manual will probably be the single most important documentation you will ever need.

b. Model information for ERS-210

We have four ERS-210 robots and two ERS-210A robots. Design-wise the two models are the same with the exception that the ERS-210A runs at 400 MHz as opposed to the ERS-210 which run at 200 MHz. Please refer chapters 2 to 5 in the model information manual, for specific information on joints etc.

c. *API Specifications*

There are two documents – The level II Reference Guide, and Internet Protocol. These have more information on some of the networking stuff, and other stuff that you might want to reference. Take a look at them, but you don't need to read them yet.

Sample Programs

The best way to learn any language is to take a look at the sample programs. The sample programs are available in /usr/OPENR_SDK/sample. Copy the folder into a directory where you have write permissions. (Be sure to *chmod* it appropriately.) You can run any demo by going to the respective directory and executing:
make PREFIX=/usr/OPENR_SDK

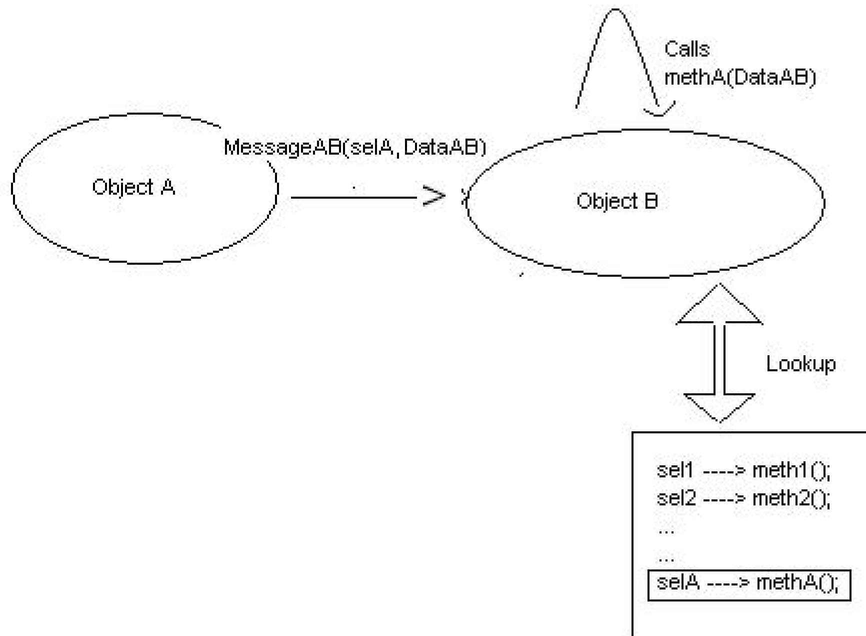
© 2002, Borg Lab, Georgia Institute Of Technology

Basic OPENR Program Constructs

OPENR Objects

An OPENR object is more like a UNIX process. Each executable file in the final program (with a .bin extension) corresponds to one object. Each object communicates with other objects by passing messages. Each message consists of a selector (which is an integer) and some data The flow of execution is event-driven.

1. Object waits for messages to arrive.
2. Once a message arrives, it calls the corresponding method based on the selector. The method – selector combinations are specified in the stub.cfg file. The arguments for the method are the data included in the message. The method might pass messages to other objects.
3. Wait for more messages.



© 2002, Borg Lab, Georgia Institute Of Technology

Inter-object communication

The object that sends a message is called the "subject", while the object that receives the message is called the "observer".

To send a message, the subject first sends a 'NotifyEvent' to an observer or set of observers. The subject then waits for a 'ReadyEvent' from the observer. The 'ReadyEvent' from the observer back to the subject can either be:

a) ASSERT_READY – if the observer is ready to receive a message from the subject.
b) DEASSERT-READY – if the observer is not ready to receive a message

Objects are single-threaded. They cannot process more than one message at a time. If the observer receives a new message when it is processing some other message, then the new message is put in the message-queue. Each object has one message-queue.

Core classes – entry points, member functions, doXXX() functions,

Each object has a corresponding core class. A core classes is a C++ class that defines the object's data and methods.

Here is an example of a simple Class[1].

```
#include <OPENR/OObject.h>
#include <OPENR/OSubject.h>
#include <OPENR/OObserver.h>
#include "def.h"

class SampleClass : public OObject {

public:
SampleClass();
virtual ~SampleClass() {}

OSubject* subject[numOfSubject];
OObserver* observer[numOfObserver];

virtual OStatus DoInit(const OSystemEvent& event);
virtual OStatus DoStart(const OSystemEvent& event);
virtual OStatus DoStop(const OSystemEvent& event);
virtual OStatus DoDestroy(const OSystemEvent& event);

//Describe the member functions corresponding to Notify,
//Control, Ready, Connect method.
};
```

The different parts of the sample program are explained below:

© 2002, Borg Lab, Georgia Institute Of Technology

```
#include <OPENR/OObject.h>
#include <OPENR/OSubject.h>
#include <OPENR/OObserver.h>
#include "def.h"
```

`OSubject.h` has the necessary method declarations for the Subjects while `OObserver.h` has the corresponding declarations for the Observers. The `def.h` file is generated by the *stubgen* program (which will be discussed later).

```
class SampleClass : public OObject {
```

All core classes in OPENR are and must be inherited from OObject. This is very similar to Object class in Java.

```
OSubject* subject[numOfSubject];
OObserver* observer[numOfObserver];
```

This indicates the number of subjects and the observers that the class will have. Every program must have these. The `def.h` file defines the `numOfSubject` and `numOfObserver`.

```
virtual OStatus DoInit(const OSystemEvent& event);
```

The DoInit() method is called at startup to initialize all the instances and variables.

```
virtual OStatus DoStart(const OSystemEvent& event);
```

The DoStart() Method executes right after DoInit() finishes execution.

```
virtual OStatus DoStop(const OSystemEvent& event);
virtual OStatus DoDestroy(const OSystemEvent& event);
```

The DoStop() Method is called at shutdown. This is followed by DoDestroy() Method which destroys all the subject and observer instances, and calls the destructor function.

In addition to these four methods, at least four other methods have to be defined to enable inter object communication:

Control Method

This method is used by the subject to receive the connection results with its observers.

Connect Method

This method is used by the observer to receive the connection results.

Notify Method

This method is used by the observer to receive a message from the subject.

Ready Method

This method is used by the subject to receive the ASSERT-READY and the DEASSERT-READY notifications.

The DoXXX( ) Method MACROs

In most cases, you will have to include all these macros in every core class you make.

DoInit( )

NEW_ALL_SUBJECT_AND_OBSERVER

This registers the necessary number of subjects and observers.

REGISTER_ALL_ENTRY

This registers the connection to services offered by other objects.

SET_ALL_READY_AND_NOTIFY_ENTRY

This registers all entry points.

DoStart( )

ENABLE_ALL_SUBJECT

This enables all the subjects

ASSERT_READY_TO_ALL_OBSERVER

This sends ASSERT_READY to all subjects.

© 2002, Borg Lab, Georgia Institute Of Technology

DoStop( )

DISABLE_ALL_SUBJECT

This disables all subjects of core class.

DEASSERT_READY_TO_ALL_OBSERVER

This sends a DEASSERT_READY to all connecting subjects.

DoDestroy( )

DELETE_ALL_SUBJECT_AND_OBSERVER

This deletes all observer and subjects.

Message Passing

When the observer is ready to receive data from the subject, the observer sends the subject an ASSERT-READY message.

```
void SampleObserver::SendAssertReady()
{
   observer[obsfunc1]->AssertReady( );
}
```

When the subject receives the ASSERT-READY signal, it sends the observer the data.

```
void SampleSubject::Ready(const OReadyEvent& event)
{
    char str[32];
    strcpy(str, “Some Text Message”);
    subject[sbjFunc2]->SetData(str,sizeof(str));
    subject[sbjFunc2]->NotifyObservers( );
}
```

When the observer receives the data, it sends an ASSERT_READY again.

```
void SampleObserver::Notify(const ONotifyEvent& event)
{
    const char* text = (const char*)event.Data(0);
    observer[sbjFunc1]->AssertReady();
}
```

© 2002, Borg Lab, Georgia Institute Of Technology

Stub file

The stub file is used to specify the control, notify, ready and connect methods specified in the previous sections. In addition to these 4 methods, the *stub.cfg* also specifies member functions that receive messages.

A Sample *stub.cfg* file has been included below:

```
ObjectName : SampleClass
NumOfOSubject : 1
NumOfOObserver : 2
Service : "SampleClass.Func1.Data1.S",Control(),Ready()
Service : "SampleClass.Func2.Data2.O",Connect(),Notify1()
Service : "SampleClass.Func3.Data2.O",Control(),Notify2()

Extra : WakeUpOnLan()
Extra : UpdateBatteryStatus()
```

A Breakdown of the format gives:

ObjectName
Name of the core class

NumOfOSubject
Number of Subjects. Minimum of 1 subject is necessary. If there are no subjects needed you have to register a dummy subject.

NumOfOObserver
Number of Observers. Minimum of 1 observer is necessary. If there are no observers needed you have to register a dummy observer.

Service:
This describes the various types if messages that is passed between the subject and its observers. The general format of a service is given here:

"(ObjectName).(Subname).(Data Name).(Service Type)",memb_func1(), memb_func2( )

Subname
This is a unique name for every service.

Data Name
The data-type used in the communication

Service Type
S (for subject) or O (for Observer)

© 2002, Borg Lab, Georgia Institute Of Technology

Memb_func1()

This function is called when a connection result is received. This function is implemented in the core class. If not needed, this entry can be "null".

Memb_func2()

If the service is a service for observers, then this method is invoked whenever a message is received from the subject. If this is for the subject, this methods is invoked each time the subject receives an ASSERT-READY or DEASSERT-READY from the observer. This function is implemented in the core class. If not needed, this entry can be "null".

Extra

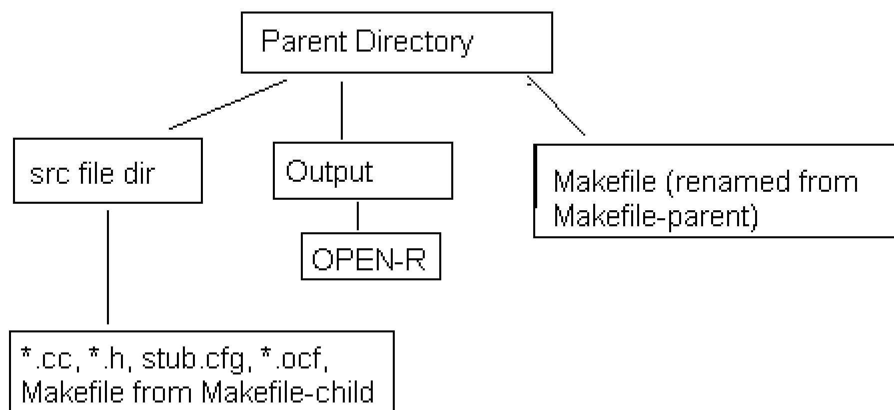
This gives additional points of entry.

© 2002, Borg Lab, Georgia Institute Of Technology

Building

1. Make a Directory (let's call it *Output*) in the parent directory of the directory where you have the source files. We will put the final output executables in this directory.
2. Copy the directory OPEN-R from /usr/OPENR_SDK/MS into *Output.*
3. Copy Makefile-parent from /usr/OPENR_SDK/MS to <Parent dir>/Makefile
4. Copy Makefile-child from /usr/OPENR_SDK/MS to <src file dir>/Makefile
5. Edit both makefiles.
6. type make;make install in the parent directory.

   So the final directory structure should look something like this –



Making .ocf file

This file has to be made prior to compiling and linking your files.

The file contains a single line of the format

object OBJ_NAME STACK_SIZE HEAP_SIZE SCHED_PRIORITY cache tlb user

OBJ_NAME – This is the name of your object

STACK_SIZE – This is the size of your stack in bytes. Be sure to allocate enough number of bytes

HEAP_SIZE – This specifies the size of the heap in bytes.

SCHED_PRIORITY – This denotes the scheduling priority. Set it to 128.

Edit the stub.cfg file as described in the previous section.

© 2002, Borg Lab, Georgia Institute Of Technology

Editing the CONNECT.CFG file

Edit the connect.cfg file in <parent>/MS/OPEN-R/MW/CONF.

e.g.

object1.sendStream.datatype1.S object2.recvStream.dataType1.O

This connects the appropriate services between the subject and observer. The left and right hand sides are registered as services in the two objects' STUB.CFG files.

Editing the OBJECT.CFG file

Edit the object.cfg file in <parent>/MS/OPEN-R/MW/CONF.

Add the names of all the executables that you have made (.bin files) to the existing list (tinyftpd.bin and powermon.bin). Do not remove the tinyftpd.bin and powermon.bin entries.

© 2002, Borg Lab, Georgia Institute Of Technology

Running

IF this is the first time, you are running your program, then follow method A. If you are adding stuff to your program, then refer to method B.

Method A

1. Insert the memory stick in the laptop.
2. At the prompt. Type `mount /sony`
3. Once the memory stick has been mounted, then copy the files from /MS/OPEN_R to /sony
   ```
   cp –rf <parent dir>/MS/OPEN_R /sony
   ```
4. un-mount the memory stick.
   ```
   umount /sony
   ```
5. Put the memory stick in the dog and press the power button on the dog.

Method B

1. Copy the perl script ftpdog.pl from /usr/OPENR_SDK/ and run the following in your parent directory
   ```
   ftpdog 192.168.1.[49 – 54] MS/OPEN-R,
   ```
   The IP of the dogs will be pasted on the dog.

Debugging

For more information on debugging, please refer to chapter 5 in the programmer's manual. The explanation in the chapter is very clear and extensive.

© 2002, Borg Lab, Georgia Institute Of Technology

Flow of development

1. Design your objects
2. Decide on the data type for the inter-object communication
3. Write the stub.cfg file
4. Write the core class with the necessary member functions
5. Write connect.cfg file
6. Write your .ocf file
7. Build
8. Write object.cfg and designdb.cfg files
9. Execute on AIBO
10. Debug (perhaps)

Sample programs are the best way to learn programming a new language. In the next section, I will go over a couple of sample programs. My sample programs are simplified versions of Sony's sample programs[1].

Please read the copyright notice at the end of the document.

Sample Program #1: Helloworld

This program introduces you to the OPENR programming model. Nothing fancy – just prints hello world.

© 2002, Borg Lab, Georgia Institute Of Technology

Sample Program #2: A more advanced program

In this program the following features are illustrated:

1. Message passing
2. Using touch sensors
3. Using the LEDs (fine debug tools)
4. Joint motion control
5. Using the camera
6. Using the microphone and speakers.
7. Inter-robot communication

Two dogs are used in this experiment. When someone taps one of of the dog's back sensor, the LEDs on the dog light up, and the dog looks for an orange ball. When the dog spots the orange ball, he wags his tail and sends an audio file to the other dog, which plays the file.

© 2002, Borg Lab, Georgia Institute Of Technology

Help Information

If you have any problem with any of the steps after the tutorial, you can come to the borg lab during my office hours (which I will let you know after the tutorial.) I am also available via email, gte751m@prism.gatech.edu. Please prefix your subject with [CS 8803]. You can also email amrs@robocup.biglist.com.

Bibliography and copyright information

[1] Some code snippets and the organization of discussion have been borrowed from Sony's AIBO programming manual. All materials taken from the manual are explicit copyrights of Sony. Please do not distribute this tutorial outside the class.