AIBO Sony – Introduzione all'uso

Laura D'Angelo, Igor Colombo, Gilberto De Caro, Alessandro Quattro
Laboratory of Applied Intelligent Systems (AIS-Lab)
http://ais-lab.dsi.unimi.it

Sommario

Il documento si propone di essere una rielaborazione della documentazione reperita su internet, sia proveniente da Sony che non.

1. Introduzione all'AIBO

L'Aibo della Sony è un cane robot costruito per scopi ludici, ma che ha avuto e continua ad avere un elevato utilizzo in campo universitario. Offre funzionalità e feature che lo rendono molto interessante per un suo utilizzo nello studio dell'intelligenza artificiale e della robotica.

1.1 Caratteristiche Hardware/Sofware degli Aibo

Le specifiche Hardware di Aibo modello ERS7 (l'ultimo modello prodotto dalla Sony) sono:

- Processore: Risc 64bit, MIPS R7000 a 576Mhz
- Memoria: SDRAM 64MB
- Storage Media: un slot per Memory Stick (da 8 o 16MB)
- Telecamera: CMOS a 350.000 pixel, 30FPS (con algoritmo di riconoscimento colori implementato in Hardware)
- Scheda di Rete: WiFi IEEE 802.11b
- Giunture: Bocca: 1 grado di libertà / Testa: 3 gradi di libertà / Zampe: 3 gradi di libertà per "zampa" / Orecchie: 1 grado di libertà / Coda: 2 gradi di liberà
- Sensori:
- Temperatura
- Di distanza ad infrarossi: sul corpo: (da 0.1 a 0.9 m); sul muso (near: da 0.05 a 0.5m; far: da 0.2 a 1.5m).
- Accelerazione: su tre assi.
- Elettrorstatico: su testa e corpo (carezza e tocco)
- Pressione: uno per zampa più uno sul mento
- Vibrazione
- Audio: Input: microfono stereo, frequenza di cambionamento 16Khz a 16bit. Output: 2 Speaker, frequenza 8Khz/16Khz a 8/16bit.
- Led: 28 sul muso, 2 sulle orecchie, 2 vicino al sensore elettrostattico sulla Testa, 1 per funzionalità rete WiFi sulla Testa e 16 sul corpo.

L'Aibo utilizza **Aperios** un sistema operativo RealTime per sistemi Embendded sviluppato dalla Sony.

Il supporto di memorizzazione utilizzato dagli Aibo sono le **Sony Aibo Programmable Memory Stick** (di colore **rosa**). Sono sconsigliate Memory Stick di altri produttori e modelli differenti. Tutto il codice prodotto verrà caricato su questi supporti per poter essere letto ed eseguito dagli Aibo.

Le Programmable Memory Stick utilizzano un filesystem FAT16, pertanto i file devono rispettare il **formato 8+3** (8 caratteri per il nome e 3 caratteri per l'estensione).

2. Ambiente Aibo Open-R

Nello spiegare questi concetti faremo riferimento ad un esempio. Si vuole realizzare un'applicazione per fare mettere "a cuccia" l'Aibo (cf. Fig. 1). Una possibile soluzione consiste nel creare un oggetto **('WakeUp') controllore del movimento** che al ricevimento di un comando ("START"), da parte di un altro oggetto **('Commander'), attivatore del movimento,** attiverà i motori dell'Aibo e inizierà una comunicazione con l'oggetto di sistema '**OVirtualRobotCom', che attiva i motori associati alle giunture,** per far muovere l'Aibo fino al raggiungimento della posizione "a cuccia".

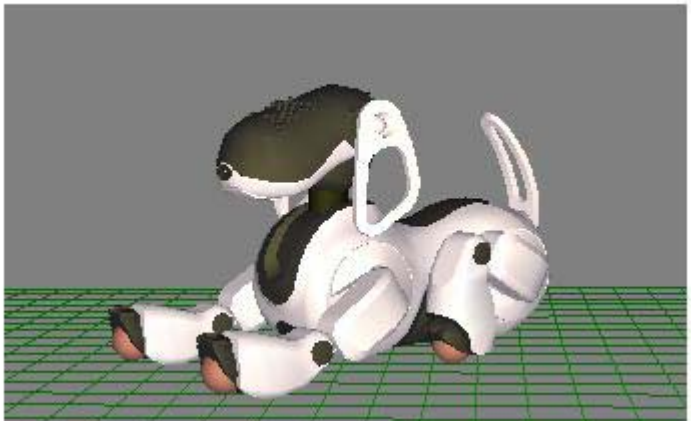

Fig. 1- Posizione "a cuccia".

2.1 Architettura Open-R

L'architettura OPEN-R è organizzata a strati. In particolare sono presenti 2 livelli principali:il livello di sistema e il livello delle applicazioni.

In mezzo ai due strati è presente un'interfaccia che definisce i servizi offerti dal livello di sistema al livello delle applicazioni. Tra questi l'input e l'output di audio, l'input di immagini, l'output di controllo dei joint e l'input di dati provenienti dai diversi sensori.....

Questi servizi permettono agli oggetti del livello applicativo di utilizzare le funzionalità del robot senza conoscere i dettagli dei device hardware (Fig. 2).

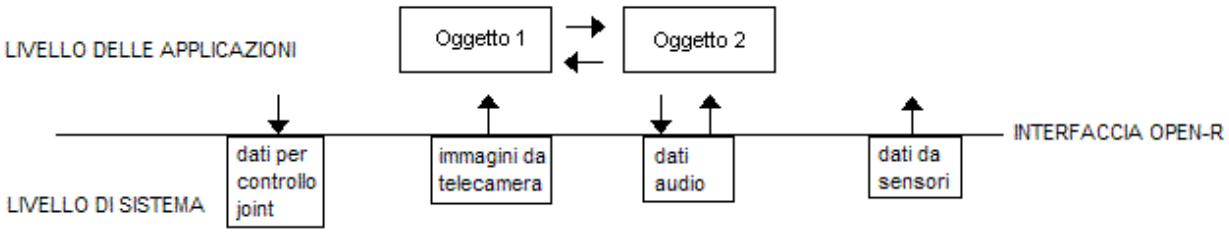

Fig. 2 - Architettura OPEN-R.

2.2. La programmazione modulare e gli open-r object

L' ambiente di esecuzione messo a disposizione dall'AIBO è concepito come un insieme di oggetti **concorrenti,** simili a processi UNIX, chiamati OPEN-R object, questi oggetti sono fisicamente rappresentati da file binari. Gli OPEN-R object vengono caricati in memoria all'avvio dell'AIBO leggendo i files binari dalla memory stick. La metotologia di sviluppo è orientata agli oggetti, sia in quanto basata su linguaggio C++, sia intesa come strutturazione di un applicativo in ottica di oggetti o moduli concorrenti come indicato sopra.

Gli oggetti comunicano tra loro attraverso scambio di **messaggi** o attraverso una o piu' zone di **memoria condivisa** (da un punto di vista tecnico, lo scambio di messaggi avviene sfruttando un buffer di memoria condivisa interamente gestito da Aperios in modo da essere trasparente per gli oggetti).

All'avvio, Aperios carica piu' di cinquanta oggetti OPEN-R, questi rappresentano il livello sistema dell'Aibo e gestiscono i motori, i sensori, i led, la telecamera.... (cf. Appendice A). Successivamente vengono caricati gli oggetti creati dall'utente che comunicheranno sia tra loro che con gli oggetti di sistema per poter svolgere i loro compiti.

Un OPEN-R Object contiene un certo numero di entry point; ciascun entry point corrisponde a un particolare metodo dell'oggetto, e ogni metodo corrisponde a una particolare funzione membro della classe.

Un oggetto OPEN-R puo' essere visto come un processo singolo che lavora in parallelo con gli altri oggetti OPEN-R. E' importante capire che nella programmazione degli Aibo l'unico modo per creare un processo è quello di creare un oggetto OPEN-R, **non sono state implementate funzioni fork**.

Gli OPEN-R object ereditano dalla classe base Oobject. Da Oobject ereditano quattro metodi, che sono quindi comuni a tutte le classi. Questi sono:

- `DoInit()` Questo metodo viene chiamato quando l'oggetto caricato dal sistema. Inizializza tutti gli entrty point e registra subject e observer dell'oggetto.
- `DoStart()` Questo metodo viene chiamato dopo che DoInit() stato eseguito in tutti gli oggetti e dopo aver abilitato tutti i subject spedisce un segnale di Assert Ready (vedi dopo) a tutti gli objects.
- `DoStop()` Questo metodo chiamato all arresto del sistema, disabilita tutti i subject e spedisce una Dessert Ready a tutti gli observers; in questo modo loggetto non pu pi ricevere messaggi.
- `DoDestroy()` Questo metodo viene chiamato all arresto del sistema, dopo che DoStop() è stato chiamato in tutti gli oggetti.

Altri metodi (con eventualmente altri entry-point), possono essere definiti dal programmatore.

Non è possibile definire sugli open-r object il copy-costructor e l'operatore "=", questo perche' un oggetto OPEN-R non va inteso come un normale oggetto C++. Un oggetto OPEN-R, infatti, è un processo che lavora in modo concorrente con gli altri oggetti e rappresenta un'entità a sè stante compilato ed eseguito indipendentemente dagli altri oggetti.

2.3 Come caricare un oggetto proprietario

Una volta creato **l'eseguibile associato ad un oggetto (*.bin)**, questo viene messo assieme agli altri *.bin proprietari, nella directory MS/OPEN-R/MW/OBJS dell'AIBO Programming Memory Stick, che viene poi inserita nel robot.

Occorre poi elencare nel file MS/OPEN-R/MW/CONF/OBJECT.CFG tutti gli oggetti che si vogliono caricare.

Quando l'AIBO viene acceso il sistema software carica gli oggetti elencati nel OBJECT.CFG nell'ordine in cui sono scritti, e li istanzia; subito dopo avere caricato tutti gli altri oggetti di sistema.

Nel nostro esempio OBJECT.CFG conterrà i nomi dei 2 oggetti che costituiscono l'applicazione.

```
/MS/OPEN-R/MW/OBJS/COMMNDER.BIN
/MS/OPEN-R/MW/OBJS/WAKE_UP.BIN
```

2.4 Tipico ciclo di vita di un oggetto

1) caricato dal sistema (all'accensione).

2) esecuzione di DoInit (all'accensione)
3) esecuzinone di DoStart (all'accensione)
4) in attesa di un messaggio
5) quando riceve un messaggio esegue un metodo dell'oggetto. Eventualmente può inviare alcuni messaggi ad altri oggetti.
6) quando finisce l'esecuzione del metodo torna in attesa di un messaggio (2)
7) allo spegnimento esecuzione di DoStop quindi DoDestory (spegnimento).

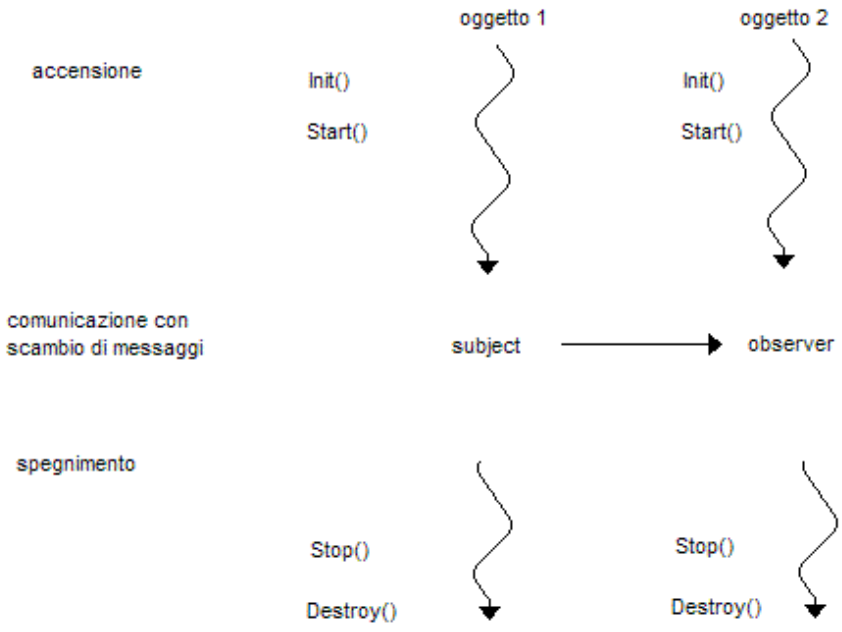

Fig. 3 - Ciclo di vita degli oggetti. Per la definizione di subject ed observer vedi dopo.

Quando il robot viene acceso, su tutti gli oggetti, viene chiamato il metodi Init() e subito dopo il metodo Start(). Questi due metodi corrispondono rispettivamente alle funzioni DoInit() e DoStart().

Quando il robot viene spento vengono chiamati, su tutti gli oggetti, i metodi Stop() e Destroy() che corrispondono rispettivamente alle funzioni DoStop() e DoDestroy().

Un oggetto non può terminare da solo. Rimane in esecuzione finchè il sistema è attivo.

3. Comunicazione tra oggetti

Gli OPEN-R object in esecuzione comunicano tramite lo **scambio di messaggi**, "comunicazione inter-oject". L'uso di questa comunicazione permette di creare gli oggetti separatamente e di connetterli in un secondo momento e quindi mantenere una modularità nella scrittura del codice.

La comunicazione è attivata e sincronizzata da eventi. Il sistema nel complesso è **event driven.** Ad esempio: la lettura di un sensore scatena l'invio di un messaggio all'oggetto preposto ad interpretare ed eseguire un azione. Ciascun OPEN-R object è **single thread,** ovvero in ricezione può gestire un solo messaggio alla volta, **se durante l'elaborazione di un messaggio ne giungono altri, il sistema li pone in una coda**. Verranno reinviati uno per uno all' OPEN-R object solo al termine dell'elaborazione corrente.

3.1 Protocollo di comunicazione e configurazione

La comunicazione è impostata dichiarando per ciascun oggetto dei **servizi di ingresso detti observers** (classe OObserver) e dei **servizi di uscita detti subjects** (classe OSubject). Questi servizi possono essere considerati alla stregua di porte di comunicazione. Un servizio subject permetterà di *inviare messaggi*, un servizio observer di *riceverli*.

Un modulo può avere più servizi, ovvero diversi subject e diversi observer.

3.2 Ruolo del file stub.cfg

I servizi offerti da ogni oggetto vengono dichiarati nel file stub.cfg. Il programmatore compila **per ciascun oggetto un file stub.cfg** dove indicare **tutti i subject e observer** di cui l'oggetto dispone e dichiara, per ciascuno, se necessario, gli entry points da eseguire alla ricezione o invio di messaggi.

Ad **esempio** il file Stub.cfg associato al nostro esempio è il seguente:

```
ObjectName : WakeUp
NumOfOSubject   : 1
NumOfOObserver  : 1
Service : "WakeUp.ReceiveCommand.char.O", null, ReceiveCommand()
Service : "WakeUp.Move.OCommandVectorData.S", null, Ready()
```

Nella prima riga viene indicato il nome dell'oggetto (Wake up). Le successive due righe dichiarano il numero di servizi subject e di servizi observer che dell'oggetto. Nel nostro caso **'WakeUp' ha un solo servizio observer** per ricevere messaggi dall'oggetto 'Commander' e un **solo servizio subject** per inviare messaggi all'oggetto 'OVirtualRobotCommon'. Quindi NumOfOSubject e NumOfOObserver sono stati messi entrambi uguali a 1.

È comunque sempre necessario specificare almeno un subject e un observer, eventualmente fittizi.

Successivamente **ogni servizio è descritto su una riga**. La descrizione del servizio inizia con una stringa che corrisponde al nome della connessione. Questa stringa è costituita da 4 parti:

- nome dell'oggetto. (WakeUp).
- nome del servizio. (ReceiveCommand per il primo servizio e Move per il secondo)
- tipo di dato contenuto nel messaggio.
- tipo di servizio uguale a S per subject O per observer.

La seconda parte della descrizione corrisponde al nome della funzione chiamata quando viene ricevuto il risultato di una connessione (in genere questa funzione non serve e, come nel nostro caso, viene specificato null per non far chiamare alcuna funzione).

L'ultima parte della descrizione corrisponde a:

- al nome della funzione chiamata (Receive Command) viene ricevuto un messaggio (char) da un oggetto subject (Commander) nel caso il tipo di servizio sia observer,
- al nome della funzione chiamata (Ready) quando viene ricevuto un messaggio di tipo ASSERT-READY o DEASSERT-READY da un oggetto observer (OVirtualRobotCom) nel caso il tipo di servizio sia subject. Se questa funzione non è necessaria viene specificato null.

3.3 Ruolo del file connect.cfg

Fino a questa fase, i moduli sono considerati a sè stante. Occorre ora definire le loro connessioni, ovverosia i **canali di comunicazione**. Questo viene fatto nel file MS/OPEN-R/MW/CONF/CONNECT.CFG all'interno della memory stick, nel quale viene definita la **connessione tra subjects e observers di moduli diversi**. Ogni subject o obsever è visto dal sistema come un servizio, preposto all'invio o alla ricezione di messaggi; il file connect.cfg indica le connessioni tra tali servizi **impostando quindi i canali di comunicazione**.

Un canale di comunicazione è unidirezionale (unisce quindi un solo subject con un solo observer) ed è possibile mandare solo un tipo di messaggio; **per instaurare una comunicazione bidirezionale tra due oggetti e' necessario definire due canali di comunicazione**.

Il file CONNECT.CFG associato al nostro esempio sarà il seguente:

```
Commander.SendWakeUp.char.S WakeUp.ReceiveCommand.char.O
WakeUp.Move.OCommandVectorData.S OVirtualRobotComm.Effector.OCommandVectorData.O
```

La prima riga descrive la connessione tra il servizio subject **SendWakeUp** dell'oggetto '**Commander**', descritto nella parte sinistra, e il servizio observer **ReceiveCommand** dell'oggetto '**WakeUp**', descritto nella parte destra.

Ovviamente il tipo di messaggio scambiato deve essere lo stesso da entrambe le parti, in questo caso di tipo char.

3.4 Sincronizzazione della comunicazione

Ricordiamo che il funzionamento del Sistema Operativo è "event-driven". Di norma la sincronizzazione tra un subject e un observer avviene tramite la **comunicazione da parte dell'observer** al subject di un messaggio di **ASSERT-READY** (AR). Questo messaggio avvisa il subject che l'observer è ponto a ricevere i dati. **Una volta ricevuto l'AR message il subject invia i dati e attendende il successivo AR**. Questo verrà inviato quando l'observer sara di nuovo pronto alla ricezione.

L'observer è in grado di inviare un DEASSERT-READY message nel caso voglia interrompere la comunicazione con il subject fino a prossimo avviso.

La comunicazione può essere asincrona, ad esempio nel caso in cui l'invio di un messaggio non dipenda dalla ricezione di un AR. Questo può avvenire quando non viene specificato alcun entry point in ricezione o spedizione**. In questo caso le attività del subject (tra cui il controllo sulla disponibilità dell'oberver x) e l'invio del messaggio potranno essere richiamate dove ritiene opportuno il programmatore.**

Solitamente i moduli sono impostati per inviare un segnale di ASSERT_READY a tutti i subject all'inizio (all'interno di DoStart). Quando il mittente riceve un AR può inviare il messaggio contenente i dati.

Nel nostro esempio:

- L'**oggetto 'Commander'** avrà solo un servizio subject (SendWakeUp) che permetterà di inviare messaggi contenenti char.
- L'**oggetto 'WakeUp'** avrà un servizio observer (ReceiveCommand) per ricevere messaggi contenenti char e un servizio subject (Move) per inviare messaggi contenenti il tipo predefinito OCommandVectorData.

L’**oggetto di sistema ‘OVirtualRobotCommon’** ha sempre un servizio observer chiamato Effector per ricevere messaggi di tipo OCommandVectorData.

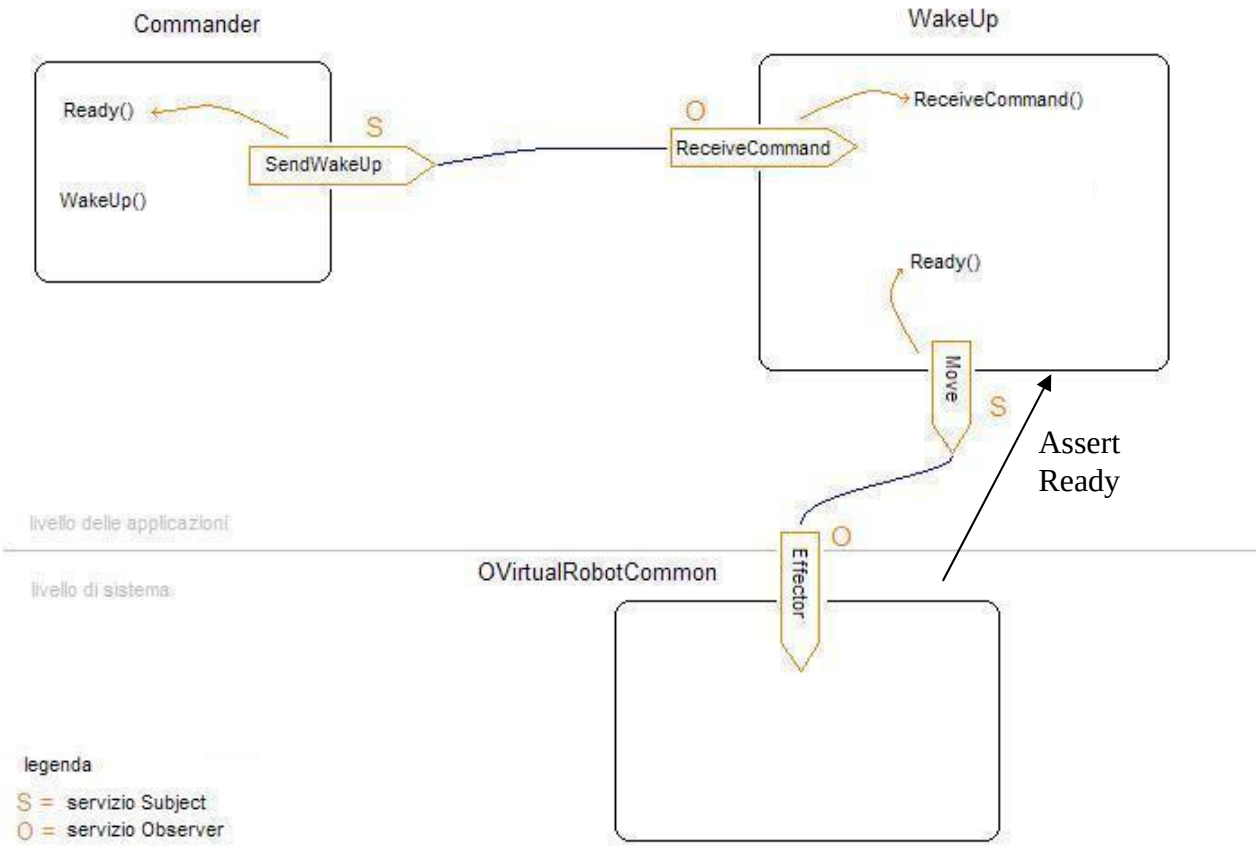

Fig. 4- Schema della sicronizzazione tra moduli nell’esempio di riferimento.

Nel nostro esempio:

- essendo ReceiveCommand un servizio observer le funzione ReceiveCommand() verrà chiamata ogni volta che il servizio ReceiveCommand riceverà un messaggio dall’oggetto subject corrispondente.
- essendo Move un servizio subject, la funzione Ready() verrà chiamata ogni volta che il servizio Move riceverà un AssertReady dall’oggetto observer corrispondente (‘OVirtualRobotCommon’)

3.5 Contenuto della comunicazione

I **messaggi** che vengono scambiati sono oggetti mutuati dagli oggetto di tipo **ONotifyEvent** in ingresso e **OReadyEvent** in uscita.

Il **contenuto** di un messaggio può essere costituito da un tipo predefinito (intero, float...) una struttura o un puntatore ad un oggetto, in quest’ultimo caso solo nel caso in cui l'oggetto non utilizzi il dynamic binding o il polimorfismo.

Inoltre gli oggetti Subject (OSubject) mettono a disposizione alcuni **metodi per l’invio dei dati.** Tra questi:

- setBufferSize(): imposta la dimensione del buffer per la ricezione dei dati dall observer.
- setData(void* buf, size_t size): generalmente permette di **copiare in un segmento di shared memory** (buffer di trasmissione) i **dati del riferimento** all’oggetto passato come paramentro per un dato Observer o per tutti gli Observers legati al subject. Gli oggetti observer vengono dotati di un buffer di ricezione adeguata. Il comando setData() di OSubject si occupa anche di **impostare il segmento di shared memory sul buffer di ricezione dell'observer o degli**

observers. Viene chiamato per ogni unità data da impostare (numero di unità impostato col comando setNotifyUnitSize()), con il relativo pacchetto di dati.

- NotifyObserver(ObserverID& id) / NotifyObservers(): **Spedisce il messaggio presente nel buffer di trasmissione** all'observer di indice ID, nel primo caso, o a tutti gli observer collegati con il subject, nel secondo caso.
- setNotifyUnitSize(): Imposta il numero **di chiamate da effettuare a SetData() per raggiungere la unità di trasmissione minima**; per esempio in alcuni casi un messaggio può essere composto da una parte header e da una parte di dati, in questo caso è possibile impostare con SetNotifyUnitSize, l'unità di trasmissione uguale a 2 quindi chiamare SetData(<header>) e poi SetData(<dati>), seguiti da NotifyObserver(). L'invio all'observer sara' effettuato solo dopo che il comando setData sul subject sarà chiamato il numero di volte impostato da questo comando. Di default è impostato ad 1.

In alternativa alla scrittura su shared memory dei dati da inviare da parte di OSubject, è possibile chiamare il metodo setData() in modo da passare direttamente un riferimento alla classe RCRegion che permette di gestire i dati direttamente su shared memory (setData(RCRegion* rgn)).

La classe **RCRegion** permette di instanziare un **puntatore alla shared memory** con il costruttore chiamato passando la dimensione di memoria desiderata. Oppure di instanziare un puntatore alla shared memory di un segmento di memoria gia' utilizzato da altre classi di oggetto con contenuto dati.

Il sistema riserva per il buffer di ingresso di un dato observer un area di shared memory dove il subject scrive i dati, questa scrittura di puo' evitare gestendo direttamente i dati con RCRegion. E' quindi sufficiente passare il riferimento di questa classe come parametro del metodo setData per impostare il puntatore al buffer di ricezione dell'observer.

3.6 Metodi a disposizione

Gli oggetti **observer** (OObserver) **mettono a disposizione i seguenti metodi**:

- AssertReady() invia ilmessaggio di AR a tutti i subject collegati o solo al subject il cui id e' passato come parametro
- DeassertReady() come sopra ma invia un deassertReady.
- NumberOfSubjects() ritorna il numero di subject collegati.

Sia per gli OSubject che per gli OObserver ci sono dei **metodi di inizializzazione** che vengono chiamati nell'entry point doInit(). L'entry point legato ad un observer per un dato oggetto sviluppato, riceve a parametro un oggetto OnotifyEvent sul quale è possibile chiamare i metodi:

- NumOfData() Il numero di pacchetti data ricevuti nel messaggio (lo stesso impostato dal subject con setNotifyUnitSize())
- SenderID() restituisce l'id del subject che ha effettuato l'invio.
- Data(<int n>) restituisce il dato in posizione n-esima dove 0<= n <= NumOfData().

OreadyEvent è l'oggetto passato a paramentro dell'entry point dichiarato per il **subject, possiede i metodi:**

- SenderID() restituisce l'id dell'observer che ne ha effettuato l'invio
- IsAssert() restituisce un bool che informa se l'OreadyEvent e' un AR.
- IsDeassert() come sopra per un DeassertReady.

Per concludere: l'ogetto OSubject e' predisposto con dei metodi che concorrono alla composizione dell'ONotifyEvent ricevuto dall'OObserver, mentre l'OObserver ha dei metodi che concorrono alla composizione dell'OReadyEvent ricevuto dal subject. OSubject e OObserver implementano i metodi per l'invio dei messaggi/eventi rispettivamente ONotifyEvent e OredyEvent.

3.7 Input/Output audio-video e sensori

Seguendo l'ottica dell'inter object communication messa a disposizione dall AIBO, possiamo identificare delle “API” di sistema che lavorano con la stessa logica precedentemente presentata.

Per leggere le informazioni dai sensori, dalla camera e dal dispositivo sonoro è possibile utilizzare dei **servizi preposti per queste attività**. Come servizi intendiamo esattamente gli stessi intesi come i sopra citati subject e observer. I fornitori di questi servizi sono gli open-r object di sistema: **OVirtualRobotComm** per i sensori e la camera, **OVirtualRobotAudioComm** per l’audio.

Tramite **la connessione ai subject di questi oggetti** (dichiarata nel file connect.cfg) ad **observer dichiarati nei propri oggetti** sviluppati, è possibile acquisire i formati “data” messi a disposizione dal sistema: posizioni angolari, segnale audio, immagini.... Viceversa per inviare i dati ai dispositivi, occorre connettere agli observer degli stessi oggetti subject dichiarati nei propri oggetti.

Gli **oggetti di tipo “data”** hanno una componente comune ODataVectorInfo contentente informazioni generiche come ad esempio:

```
MemoryRegionID memRegionID;
void* physAddr;
size t o set;
size t totalSize;
ODataType type;
size t numData;
```

Nello specifico il formato **OCommandVectorData** contiene i campi per i comandi verso le giunture e i led, **OSensorVectorFrame** contiene le informazioni inerenti tutti i sensori, e **OFbkImageVectorData** contiene le informazioni relative alla camera (immagini).

3.7 La temporizzazione del sistema

L'hardware di Aibo lavora in un ambiente temporale suddiviso in frames. **Ogni frame dura 8ms** ed è identificato da un numero legato a un contatore che va da 0 a 0x0ffffff0 (oframeMAX_NUMBER) che viene resettato una volta raggiunto oframeMAX_NUMBER.

Gli oggetti di interfaccia mandano le informazioni lette dai sensori in blocchi di 16 (osensorMAX_FRAMES) frames. Viceversa anche la comunicazione verso i sensori viene temporizzata in blocchi di 16 frames (ocommandMAX FRAMES), ovvero viene inviato un set di comandi uno per ciascuno degli 0 < n <= ocommandMAX_FRAMES frames.

La temporizzazione dei sensori in frames, non è necessariamente sincronizzata con quella del sistema nel suo insieme. Quindi se il sistema e' sul frame 21 la camera può essere ad esempio al frame 15.

La temporizzazione dorebbe essere presa in considerazione in fase di valutazione dello stato dell'AIBO in un dato istante; **dato che i dispositivi hanno refresh rate differenti e quindi per valutare le informazioni di tutti i sensori al momento e' necessario fare opportune valuazioni**: i sensori possono aver mandato 3 volte i loro dati mentre la camera nello stesso tempo solo una.

Nota le informazioni sulla temporizzazione potrebbero essere revisionate in quanto la documentazione dalla quale sono state prese si riferisce ad un modello più vecchio di aibo. Verificando le inforamzioni, purtroppo scarse, della reference guide di Sony viene confermata la presenza di frames di 8 ms per la comunicazione da e verso i sensori ma nessun'altra informazione di dettaglio.

4. Tipi di Tool di sviluppo

Sul sito openr.aibo.com sono disponibili vari tool per la gestione e programmazione degli Aibo.

L'***R-CODE*** è un linguaggio di script che permette di gestire parte delle funzionalità dell'Aibo, pertanto è semplice da utilizzare, ma offre un controllo ridotto e minor libertà di progettazione.

Il ***Remote FrameWork*** è un tool di sviluppo che lavora solo sotto Windows e permette di scrivere applicazioni in Visual C++ che controllano l'Aibo attraverso la rete Wireless; tutto il codice di controllo viene quindi eseguito sul PC che manderà all'Aibo solo i comandi da eseguire.

OPEN-R è il tool più potente messo a disposizione dalla Sony, permette di programmare e gestire tutte le parti dell'Aibo. E' basato sul C++ ed utilizza i tool standard offerti dal Linux per la programmazione.

4.1 Requisiti software di OPENR

OPENR è un **SDE** in C++ basato su compilatore gcc e può essere utilizzato sia sotto Linux sia sotto Windows (2000 e XP). La procedura di installazione è documentata nel file ***InstallationGuide_E.pdf*** nella documentazione di OPENR. Sebbene i passi da seguire per l'installazione siano, ovviamente, diversi i programmi da installare sono gli stessi sotto Linux e Windows:

- **gcc**: il compilatore C e C++
- **binutils**: una serie di utility per la programmazione tra cui, as (l'assembler), ld (il linker GNU), ar (per la creazione di archivi di file, utilizzato per la creazione di librerie) e runlib (crea un indice dei file presenti negli archivi creati con ar; velocizza il linking di liberie), strip (per rimuovere simboli di debugging da file oggetto).
- **newlib**: una libreria scritta dalla RedHat per sistemi Embedded che fornisce le librerie standard di C.

Il funzionamento del SW istallato su AIBO si è rivelato, fino ad ora, identico.

Utilizzo sotto Windows. Per potere utilizzare OPEN-R su piattaforma Windows e' necessario scaricare (da https://openr.aibo.com/openr/eng/perm/openrsdk/download/download.php4, è necessario registrarsi gratuitamente) ***Cygwin***, che e' un ambiente di sviluppo Unix che lavora sotto Windows, e MIPS cross-development tools for Cygwin, che contiene il cross-compiler gcc per Cygwin , installarli e successivamente installare OPENR, a questo punto lo SDE è pronto per essere utilizzato.

Utilizzo sotto Linux: Per potere utilizzare OPEN-R su piattaforma Linux e' necessario scaricare, dal sito openr.aibo.com, i seguenti pacchetti di sorgenti:

- gcc versione 3.3.2
- binutils versione 2.14
- newlib versione 1.10.10

Scaricare quindi (da https://openr.aibo.com/openr/eng/perm/openrsdk/download/download.php4) lo script bash build-devtools-3.2-r1.sh che ha il compito di costruire a partire dai pacchetti precedentemente indicati un cross-compiler per i processori MIPS necessario per poter compilare il codice C++ da far girare sugli Aibo.

Installato l'SDE Cygwin o il cross-compiler Linux, OPENR offre tutti i comandi per la compilazione disponibili in Linux sotto gcc, con una differenza però nel modo in cui viene compilato e generato un eseguibile.

I sorgenti vengono compilati sia sotto cygwin che sotto Linux con il comando **gcc –c**. Questo comando genera in output dei file oggetto che, a differenza che nel gcc standard, vengono poi linkati assieme utilizzando il comando **mkbin** presente nel pacchetto OPENR. mkbin genera in

output un file in formato binario con esensione **.bin** il quale viene a sua volta **compresso con gzip** e quindi **copiato nella Programmable Memory Stick** dell Aibo per la sua esecuzione.

Il comando mkbin richiede anche un **file ASCII *.ocf** che contiene alcune informazioni necessarie per il processo di link. Di seguito è riportato un esempio di file .ocf, ***commander.ocf***:

```
object commander 3072 16384 128 cache tlb user
```

Le voci indicano rispettivamente:

- **object**
- il **nome** dell'oggetto (commander).
- le dimensioni dello **stack** (in byte, 3072).
- la dimensione di cui verrà esteso lo **heap** in caso di necessità; l'heap è un area di memoria utilizzata da malloc e dall'operatore new (16384).
- **la priorità di schedulazione** (128).
- **abilitazione della cache**: *cache* abilita la cache; *nocache* disabilita la cache del processore.
- **gesionte del TLB**: *tlb* l'area di memoria dell'oggetto è allocata in uno spazio di indirizzi virtuali; *notlb*: l'area di memoria è allocata nello spazio di indirizzi fisici (il valore di TLB viene ignorato se viene utilizzata una configurazione nomemprot, guarda "Configurazioni di OPENR").
- **modo di esecuzione**: *kernel* l'oggetto viene eseguito in kernel mode; *user* l'oggetto viene eseguito in user mode.

I valori numerici riportati sono valori tipici.

La catena completa di compilazione e link, è quindi ad esempio:

```
stubgen2 stub.cfg
gcc -o Commander.o -c Commander.cc
gcc -o CommanderStub .o -c CommanderStub.cc
mkbin CommanderStub.o Commander.o commander.ocf
gzip -c commander.bin > COMMNDR.BIN
```

Il file Stub.cfg viene utilizzato durante la fase di compilazione dal **comando stubgen2** per creare alcuni file (nomeFileStub.h, nomeFileStub.cc, entry.h, def.h) necessari per generare l'oggetto.

5. Utilizzo della memory stick (rosa)

Per utilizzare un robot AIBO, occorre prima inserire la memory stick rosa nell'apposito alloggiamento e poi accendere il robot.

Nella memory stick verranno inseriti sia i **file dell'applicativo** sviluppato **che file di sistem**a e di supporto. In particolare esistono 3 possibili configurazioni del sistema:

- BASIC = Aperios base, offre tutti i servizi tranne l'accesso alla rete WiFi
- WLAN = BASIC con l'accesso alla rete WiFI
- WCONSOLE = WLAN con la possibilità di accedere via telnet al robot.

Per ogni diversa configurazione sono disponibili due ulteriori modalità:

- nomemprot = senza alcun tipo di protezione delle zone di memoria
- memprot = con zone di memoria protette, ogni oggetto lavora nella sua zona di memoria riservata, il tentativo di accedere alla zona di memoria di un altro oggetto genererà un'eccezione.

Come prima cosa occorre copiare nella memory stick vuota la directory OPEN-R corrispondente alla configurazione scelta. Le directory OPEN-R da copiare possono essere trovate nella directory usr/local/OPEN_R_SDK/OPEN_R/MS_ERS7/...

Successivamente è possibile copiare nella directory OPEN-R/MW/OBJS i file degli oggetti che costituiscono l'applicazione.

Alla fine la **struttura della directory della memory stick sarà la seguente**:

```
+-- OPEN-R \
      |
      +-- MW \
      |    |
      |    +-- CONF \
      |    +-- DATA \
      |    +-- OBJS \
      |
      +-- SYSTEM \
           |
           +-- CONF \
           +-- DATA \
           +-- OBJS \
```

Nelle **directory [CONF]** sono presenti i file di configurazione; in [DATA] è consuetudine mettere i file contenenti dati necessari alla gesionte dell'Aibo come per esempio file contenenti i suoni; in [OBJS] sono presenti gli oggetti da caricare .BIN.

Nella **directory [SYSTEM]** sono presenti tutti i file, oggetti e di configurazione, necessari al sistema operativo Aperios. Più precisamente:

- [CONF]: sono presenti i file di configurazione del sistema. I file che, normalmente, vengono modificati sono:
  - WLANCONF.TXT: file di configurazione della scheda di rete Wireless
  - EMON.CFG: file di configurazione che specifica cosa salvare e che suoni emettere in caso di eccezione.
- [DATA]: alcuni file di dati necessari ad Aperios
- [OBJS]: gli oggetti di sistema..

Nella directory [MW] sono presenti i file creati dall'utente. È questa la directory in cui vengono copiati i file creati dall'utente per poi essere eseguiti; più precisamente è composta da:

- [CONF]: sono presenti alcuni file di configurazione tra cui:
  - CONNECT.CFG: connette i servizi dei vari oggetti OPEN-R
  - OBJECT.CFG: la lista degli oggetti da caricare all'avvio dell'Aibo
  - DESIGNDB.CFG: specifica alcuni file, spesso di dati o di configurazione, che possono essere caricati in base al modello del robot utilizzato, per permettere una maggiore portabilità del codice.
- [DATA]: in questa directory sono presenti alcuni file di dati, esempio audio.
- [OBJS]: qui sono presenti i vari oggetti, .BIN, creati dall'utente e specificati nel file OBJECT.CFG.

Appendice A – I moduli caricati dal Sistema Operativo all'accensione dell'AIBO:

Numero	Nome Oggetto	ContextID	OID (ID dell'Oggetto)	Descrizione
1	systemCore	0x80282c20	0xffffffff	
2	(Handler)	0x80283a20	----------	
3	mCOOPReflector	0x80287200	0x8000000b	
4	uniMailer	0x80287360	0x8000000c	
5	mCOOPFaultHandle	0x802874c0	0x8000000d	
6	mDriveFaultHandl	0x80287620	0x8000000e	Gestore dei Fault
7	registryManager	0x80287780	0x8000000f	
8	addressManager	0x802878e0	0x80000010	
9	kernelModeLib	0x80287a40	0x80000011	
10	mCoreReflector	0x80287ba0	0x80000012	
11	idle	0x80287d00	0x80000013	
12	exceptionHandler	0x80287e60	0x80000014	Gestore delle Eccezioni
13	analyzer	0x80287fc0	0x80000015	
14	mDriveReflector	0x8028cfc0	0x80000016	
15	obletManager	0x8028ce60	0x80000017	
16	installer	0x8028cd00	0x80000018	
17	driverManager	0x8028cba0	0x80000019	
18	objectManager	0x8028ca40	0x8000001a	
19	mClassReflector	0x8028c8e0	0x8000001b	
20	memoryRegionMana	0x8028c780	0x8000001c	Gestore della memoria
21	sharedMemoryMana	0x8028c620	0x8000001d	Gestore della memoria condivisa
22	mSystemReflector	0x8028c4c0	0x8000001e	
23	mAVFaultHandler	0x8028c360	0x8000001f	
24	mAVReflector	0x8028c200	0x80000020	
25	mAVInit	0x802907c0	0x80000021	
26	avManager	0x80290660	0x80000022	
27	eventManager	0x80290500	0x80000023	Gestore degli Eventi
28	downloader	0x8028ff80	0x8000002a	
29	nonBlockingFileS	0x8028fe20	0x8000002b	
30	openrBusManager	0x8028fcc0	0x8000002c	
31	(Handler)	0x80295660	----------	
32	bmnDriver	0x8028fb60	0x8000002d	
33	(Handler)	0x80291620	----------	
34	fbkDriver	0x8028fa00	0x8000002e	Gestore della Telecamera
35	cardManager	0x80291fc0	0x8000002f	Gestore del lettore memory stick
36	(Handler)	0x80294a00	----------	
37	memoryStickDrive	0x80291e60	0x80000030	Driver del lettore memory stick
38	(Handler)	0x80291780	----------	
39	fatFileSystem	0x80291d00	0x80000031	File System Fat16 della memory stick
40	memoryStickWatch	0x80291ba0	0x80000032	
41	oobjectManager	0x80291a40	0x80000033	
42	aperiosClass	0x802918e0	0x80000034	
43	opowerManager	0x802903a0	0x80000035	Gestore della batteria
44	oserviceManager	0x80290240	0x80000036	

45	ovirtualRobot	0x802900e0	0x80000037	
46	odesignedRobot	0x802914c0	0x80000038	
47	osystemLogger	0x80291360	0x80000039	
48	ovirtualRobotCom	0x80291200	0X8000003a	Oggetto che funge da interfaccia con i joint e i sensori dell'Aibo.
49	ovirtualRobotAud	0x802957c0	0x8000003b	Oggetto che funge da interfaccia con l'apparato audio dell'Aibo
50	IPStack	0x80295500	0x8000003c	Implementa il protocollo Ipv4
51	OrinocoDriver	0x802953a0	0x8000003d	Driver scheda di rete Wireless
52	(Handler)	0x802967c0	----------	
53	OrinocoEnabler	0x80295240	0x8000003e	Gestisce la scheda di rete Wirelss
54	hookConsoleIO	0x802950e0	0x8000003f	
55	emergencyMonitor	0x80294f80	0x80000040	
56	netconf	0x80294e20	0x80000041	
57	anttcpio	0x80294cc0	0x80000042	
58	hookConsoleIOAct	0x80294b60	0x80000043	
59	powerMonitor	0x80296660	0x80000044	Controlla lo stato di carica delle batterie
60	commander	0x80296500	0x80000045	Oggetto Commander (vedi esempio)
61	wakeUp	0x802963a0	0x80000046	Oggetto WakeUp (vedi esempio)

Appendice B – Link utili.

http://www.tzi.de/4legged/bin/view/Website/WebHome: Sony Four Legged Robot League; raccoglie informazioni sulla Robot CuP (torneo di calcio tra Aibo). Si torvano link sui vari gruppi che hanno partecipato alle varie edizioni, spesso rendono disponibili i sorgenti.

http://openr.aibo.com/: Sito ufficiale di OPEN-R (in Inglese e Giapponese). E' necessaria la registrazione gratuita, è possibile scaricare i vari tool OPEN-R, RCODE, Remote Framework ed sono disponibili Tutorial e BBS.

http://www.tekkotsu.org/: Un progetto Open Source per la creazione di un Framework di sviluppo per Aibo.

http://www.ensta.fr/~baillie/openr_tutorial.html: un ottimo tutorial su Open-R scritto dal laboratorio francese ENSTA.

Librerie che potrebbero essere utili:
http://www-2.cs.cmu.edu/~coral/download/index.html
http://www-2.cs.cmu.edu/~tekkotsu/dox/classes.html

Per quanto riguarda il C++ e le sue librerie standard, trovate un tutorial veloce e ben fatto qui:
http://www.bo.cnr.it/corsi-di-informatica/corsoCstandard/Lezioni/01Indice.html

Altri siti di interesse:
http://www.cc.gatech.edu/~tucker/courses/amrs/aibo/AIBOProgrammingTutorial.pdf
http://citeseer.ist.psu.edu/689250.html : concurrencia04.pdf – ancora da visionare nel dettaglio
http://www.ai.rug.nl/vakinformatie/pas/index2.php?content=openr_camera
http://www.cs.uu.nl/docs/vakken/aibop/1-AIBOP-intro.ppt