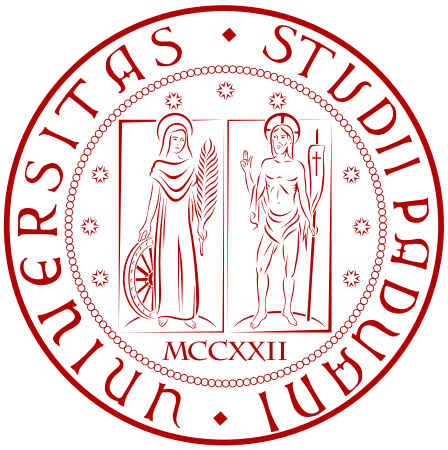
Università degli Studi di Padova
Facoltà di Ingegneria
Dipartimento di Ingegneria dell'Informazione
Tesi di Laurea Magistrale

SILENT: un'interfaccia tra Scratch e LEGO NXT



Laureando: Luca Zenatti
Relatore: Prof. Michele Moro
A.A. 2012/2013

Padova

11.12.2012

“Life is about passions. Thanks for sharing mine.”

Indice

Sommario

L'obiettivo generale dell'elaborato è quello di realizzare un collegamento operativo tra due sistemi realizzati per scopi educativi: il software Scratch e il robot LEGO MINDSTORM NXT

Il contesto in cui si inserisce il lavoro di tesi è quello della robotica educativa, ovvero l'utilizzo di strumenti software e hardware per l'insegnamento dei fondamenti della programmazione, del controllo di dispositivi automatici e delle materie scientifiche più in generale Questi insegnamenti possono essere fine a se stessi o mezzo per sviluppare capacità logiche, di problem solving e creative negli studenti.

Gli strumenti utilizzati sono stati:

- Scratch e l'ambiente Squeak nel quale esso è realizzato;
- JAVA SDK per lo sviluppo del backend;
- Robot LEGO MINDSTORM NXT 2.0;
- Linguaggio NXC.

Il presente elaborato si è concretizzato nella realizzazione del software denominato SILENT, "Scratch Interface for LEGO NXT", che realizza un'interfaccia tra Scratch e i robot LEGO NXT. Permette di programmare il dispositivo robotico didattico componendo blocchetti funzionali che rappresentano le unità fondamentali della programmazione e del controllo.

1 Introduzione

Sono numerosi i software didattici utilizzati a supporto delle più svariate attività educative: software a supporto di un insegnamento (materiale aggiuntivo elettronico o programmi da utilizzare in classe), software per lo svolgimento dei compiti a casa, software per ricerche (enciclopedie, dizionari), software specifico su hardware personalizzato (minilaptop come LeapFrog [2]), software a sostegno della formazione nelle organizzazioni (e-learning), software per necessità educative specifiche (patente, apprendimento lingua, costruzione mappe mentali, etc ...) fino a sistemi operativi pensati esclusivamente per scopi educativi (Sugar, DoudouLinux, Edubuntu, Uberstudent, ...) [1]

I software di questo tipo, che si rivolgono ad un target di bambini in età scolare elementare/media, spesso uniscono una grafica accattivante a semplici funzionalità per far avvicinare gli studenti all'uso del computer o per aiutarli ad acquisire le competenze più svariate. Alcuni arrivano a combinare abilità manuali e uso di software in modo da animare costruzioni effettivamente create, come nel caso del software LEGO MINDSTORM, che si propone in questo modo di portare il ragazzo a compiere un ulteriore passo.

A questo proposito cominceremo il nostro lavoro con un'analisi orientativa del mondo del software didattico Scratch, pensato per introdurre e sviluppare nello studente capacità logiche e di problem solving attraverso la programmazione, dei robot LEGO MINDSTORM NXT, che uniscono la versatilità del mondo LEGO al gioco e alla possibilità di realizzare dispositivi robotici autonomi con il minimo sforzo.

1.1 Software didattici Scratch/BYOB

Scratch, e il suo derivato BYOB (acronimo di Build Your Own Block), rappresentano una delle possibilità per quanto riguarda software didattici pensati per l'insegnamento della programmazione a bambini di età scolare.

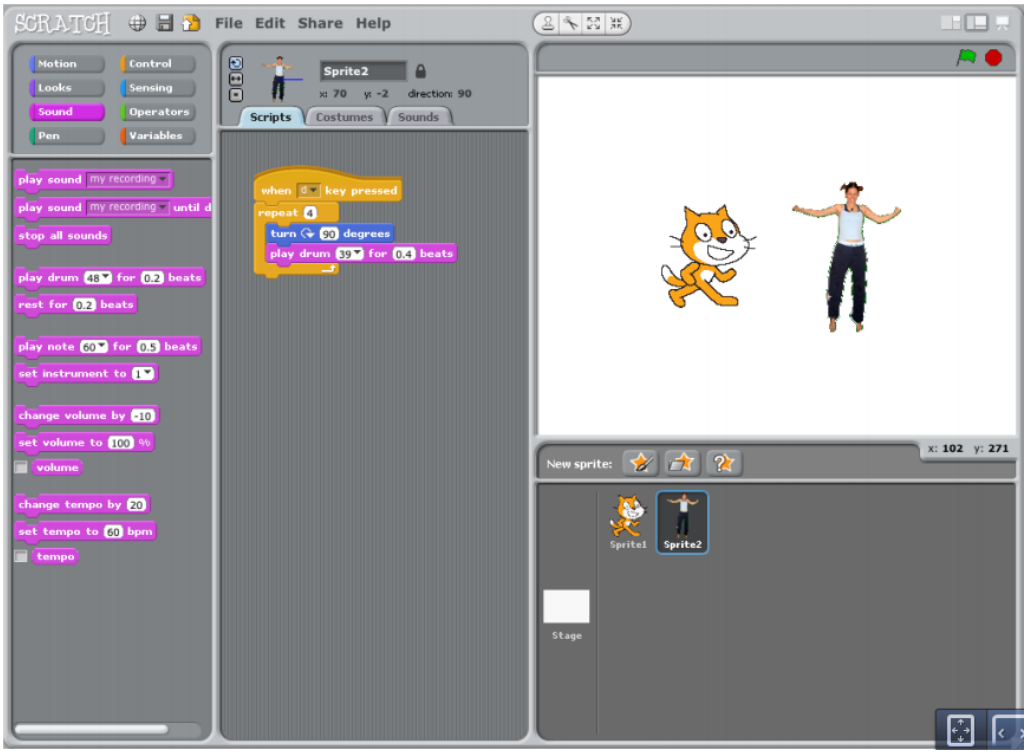


Figura 1.1: Scratch

Esistono diversi software a disposizione degli educatori che vogliono cimentarsi nell'insegnamento della realizzazione di un programma (Alice [3], PythonTurtle [4], etc ...); la particolarità di Scratch, che approfondiremo nel seguito, è quella di permettere un agile passaggio attraverso 3 fasi cardine: immaginare, creare e condividere.

Attraverso un ambiente intuitivo e un'interfaccia semplice da usare il bambino è messo nelle migliori condizioni per poter realizzare ciò che la sua immaginazione ha pensato, vederne immediatamente il risultato e condividere la creazione con la comunità di Scratch. Byob è un mod di Scratch che implementa la ricorsione e altre funzioni leggermente più avanzate.

Scratch è stato progettato tenendo come linee guida l'educazione e l'apprendimento. Nel momento in cui i ragazzini creano e condividono i progetti in Scratch, essi sviluppano importanti capacità di progettazione e di problem-solving, imparando a pensare creativamente, ragionare sistematicamente e a lavorare collaborativamente.

Scratch può essere usato in molti differenti ambiti: scuole, musei, community center e naturalmente anche a casa. E' inteso in particolare per ragazzini di età compresa tra gli 8 e i 16 anni, ma bambini più giovani possano lavorare sui progetti Scratch

con i loro parenti, mentre i più grandi possono usarlo in corsi di introduzione alle scienze informatiche.

Nel capitolo 2 affronteremo nel dettaglio Scratch e la sua maggior variante, delucidandone il funzionamento e i principi cardine.

1.2 Robot didattici LEGO NXT

LEGO Mindstorms NXT è un kit robotico programmabile rilasciato dalla Lego alla fine di luglio 2006 [10] e che rimpiazza il kit Lego Mindstorms di prima generazione,che era chiamato Robotics Invention System.

Lego Mindstorms è una linea di prodotti LEGO che combinano mattoncini programmabili con motori elettrici, sensori, mattoncini LEGO, pezzi di LEGO Technic (come ingranaggi, assi e parti pneumatiche) per costruire robot e altri sistemi automatici e/o interattivi.

LEGO Mindstorms può essere usato per costruire un modello di sistema integrato con parti elettromeccaniche controllate da computer. Praticamente tutti i tipi di sistemi integrati elettromeccanici esistenti nella vita reale (come gli elevatori o i robot industriali) possono essere modellati con Mindstorms. [5]

La particolarità di LEGO NXT è che rende semplice, relativamente economico e didatticamente realizzabile lo studio del mondo robotico giocando con il paradigma che si rifà ai più classici mattoncini LEGO, alle componenti elettromeccaniche e ad interfacce software di diverso livello di difficoltà e controllo, permettendone l'utilizzo praticamente a qualsiasi livello, dal semplice gioco fino alle attività di corsi Universitari avanzati.

Nel capitolo 3 affronteremo nel dettaglio il mondo di LEGO Mindstorm NXT.

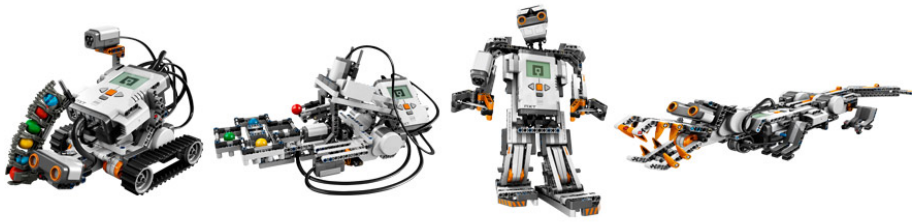


Figura 1.2: Esempi di robot costruiti con LEGO MINDSTORM NXT

1.3 Obiettivo tesi

L'obiettivo della presente tesi è quello di studiare e realizzare un'interfaccia software che permetta di sfruttare le potenzialità dei due supporti didattici: Scratch e LEGO MINDSTORM NXT.

Per quanto infatti esistano già soluzioni che permettono di utilizzare i robot della LEGO, è interessante poter unire le potenzialità di Scratch come strumento di apprendimento della programmazione e il divertimento nel costruire un semplice robot con i mattoncini della LEGO, collegarli al computer e provare a tradurre in realtà l'algoritmo realizzato sullo schermo. Il lavoro di questa tesi potrebbe essere il passo successivo all'apprendimento dei concetti fondamentali della programmazione e quello subito precedente all'accesso a strumenti più avanzati o addirittura alla programmazione testuale.

Per come è strutturato, infatti, il software permette infatti sia di ignorare completamente i dettagli più tecnici della programmazione del robot sia di interessarsene andando a curiosare nei file sorgenti prodotti in uscita, permette di concentrarsi solamente sul come far muovere il sistema autonomo per risolvere il problema che si ha di fronte e non di come, in termini di sintassi e competenze, scrivere e far "girare" il programma che poi lo realizzerà.

La curiosità è proprio la molla che si spera di caricare nel giovane studente alle prime esperienze col mondo della robotica, che poi potrà scaricare nella direzione che più preferisce, il mondo infatti della robotica e della sua programmazione è vasto e ricco di possibilità.

Lo scopo principale del presente elaborato è quello di generare codice eseguibile sul robot propagando in ambiente Scratch la propria applicazione robotica. Si è quindi voluto sviluppare un'applicazione che permettesse di progettare il proprio programma scrivendolo con il paradigma a blocchi di Scratch e di interfacciarsi con il robot LEGO NXT per l'esecuzione delle istruzioni. Nella progettazione, e poi nella realizzazione pratica, si è cercato di separare i diversi aspetti coinvolti in modo da rendere l'architettura flessibile e modificabile senza particolari difficoltà.

1.4 Strumenti utilizzati

1.4.1 Software

Nei seguenti paragrafi verranno illustrati brevemente gli strumenti utilizzati per lo sviluppo del lavoro di tesi.

1.4.1.1 Squeak [6]

E' un'implementazione moderna e open-source del linguaggio Smalltalk e del suo ambiente creato più di 35 anni fa.

Smalltalk [9] definì i termini dell'orientazione ad oggetti dei linguaggi di programmazione è il primo linguaggio in cui qualsiasi cosa è costruita a partire da un oggetto. Smalltalk è stato profondamente ispirato dalle idee di Simula, Sketchpad e Lisp. A tutt'oggi Smalltalk fissa l'asticella per i linguaggi e gli ambienti orientati agli oggetti, interattivi, dinamicamente e fortemente tipizzati.

A differenza dello standard statico, basato su file, degli altri linguaggi come Ruby o Phyton, Squeak offre un ambiente uniforme e riflessivo[1] nel quale "vivono" gli oggetti creati. In questo ambiente, quando avviene un cambiamento su di un oggetto, il suo comportamento cambia immediatamente senza avere la necessità di riavviare il software nel quale è implementato. E' anche possibile modificare o creare oggetti mentre l'applicazione sta girando.

Squeak include librerie e plugin per la macchina virtuale per applicazioni multimediali avanzate come 2D con anti-alias, 3D accelerato, sintesi realtime di suoni e musica, video MPEG2 e molto altro. Inoltre Squeak ha uno degli ambienti riflessivi più avanzati mai creati con oltre 600 pacchetti disponibili per il download e l'installazione.

E' quindi altamente portabile rendendo facile l'operazione di debug, di analisi e di modifica. Squeak è un buon veicolo per un vasto range di progetti dalle applicazioni multimediali, alle piattaforme educative fino allo sviluppo di applicazioni web commerciali.

Le caratteristiche salienti del linguaggio, ereditate direttamente da Smalltalk, sono:

[1] In informatica, la riflessione o reflection è la capacità di un programma di eseguire elaborazioni che hanno per oggetto il programma stesso, e in particolare la struttura del suo codice sorgente.

- dinamicamente tipizzato
- fortemente tipizzato
- interpretato
- Puramente Object-Oriented
- open Source
- riflessivo
- estensibile
- Cross-platform
- Cross-OS
- Cross-hardware

Inoltre peculiari di Squeak sono le seguenti caratteristiche:

- la virtual machine viaggia incorporata col programma, ovvero Squeak è un IDE per se stesso: quindi è possibile sviluppare, eseguire ed effettuare debugging nello stesso ambiente;
- è un ambiente persistente: quando si salva l'immagine, ogni singola cosa viene salvata, dal testo inserito alla posizione della finestra;
- usa la compilazione incrementale: una volta che il codice viene salvato, è pronto per essere eseguito.

Scratch è stato sviluppato in Squeak.

1.4.1.2 NXC

NXC[7] sta per "Not eXactly C" ed è un semplice linguaggio per programmare i bricks della LEGO Mindstorm NXT.

Il brick NXT ha un interprete di bytecode, fornito dalla LEGO, che può essere usato per eseguire i programmi. Il compilatore NXC traduce il codice sorgente in bytecode NXT, che può essere eseguito sul brick.

Anche se il preprocessore e le strutture di controllo del NXC sono molto similari al C, NXC non è un linguaggio di programmazione ad uso generale: ci sono molte restrizioni che derivano dalle limitazioni dell'interprete del bytecode NXT.

Logicamente NXC è definito come due parti separate:

1. Il linguaggio NXC descrive la sintassi da usare nelle scrittura dei programmi.
2. L'API di NXC descrive le funzioni di sistema, le costanti e le macro che possono essere usate dal programma. Questa API è definita in un file speciale conosciuto come "header file" che è, da default, automaticamente incluso quando viene compilato un programma.

NXC è un linguaggio ad alto livello, simile al C, realizzato in cima al 'compilatore' NBC. Next Byte Codes (NBC) è un semplice linguaggio con sintassi simile all'assembly.

Per compilare programmi NXC basta utilizzare il compilatore NBC con i file di codice sorgente con estensione ".nxc".

1.4.1.3 Java

Java è un linguaggio di programmazione orientato agli oggetti, creato da James Gosling e altri ingegneri di Sun Microsystems.

Venne creato a partire da ricerche effettuate alla Stanford University agli inizi degli anni Novanta.

Java venne creato per soddisfare quattro scopi:

- essere orientato agli oggetti;
- essere indipendente dalla piattaforma;
- contenere strumenti e librerie per il networking;
- essere progettato per eseguire codice da sorgenti remote in modo sicuro.

I programmi scritti in linguaggio Java sono destinati all'esecuzione sulla piattaforma Java, ovvero saranno lanciati su una Java Virtual Machine e, a tempo di esecuzione, avranno accesso alle API della libreria standard. Ciò fornisce un livello di astrazione che permette alle applicazioni di essere interamente indipendenti dal sistema su cui esse saranno eseguite.

Nel corso del lavoro di tesi è stato utilizzato il Java Development Kit che Oracle mette a disposizione gratuitamente insieme ad Eclipse, un IDE open-source realizzato da IBM. La versione di Java utilizzata è la 1.6.0_24, mentre il JDK è il IcedTea6 1.11.4.

Java è stato utilizzato per sviluppare tutta la parte di Backend, l'interfaccia, tra Scratch e NXT.

1.4.1.4 Bash scripting

Per realizzare una completa portabilità del software tra sistemi Linux e Windows è stato necessario utilizzare un minimo di programmazione bash, sui due sistemi operativi.

Tipicamente Bash[8] è un processore di comandi, che viene eseguito in una terminale di testo, che permette all'utente di impartire dei comandi che causano delle azioni. Bash può anche leggere i comandi da un file, in questo caso viene chiamato script.

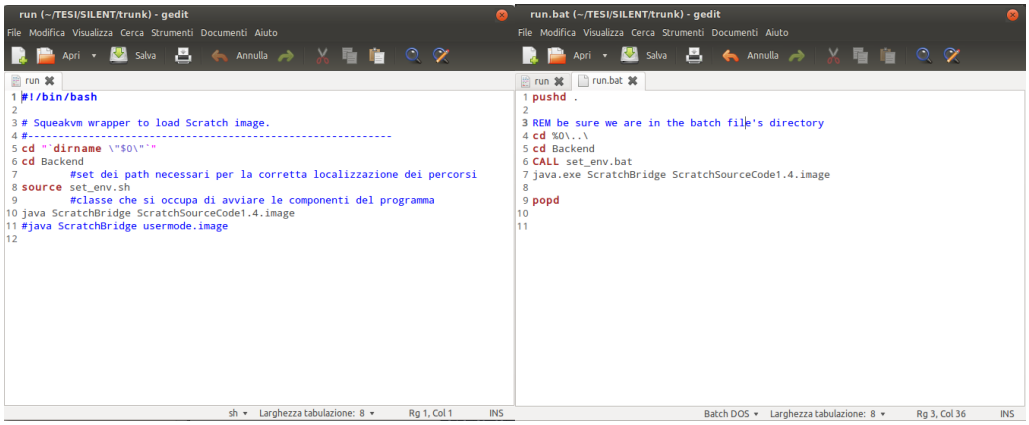


Figura 1.3: Due esempi di script bash, nel primo caso in ambiente Linux, nel secondo in ambiente Windows.

A seconda del SO sono quindi a disposizione comandi e programmi che permettono di realizzare script per il compimento di determinate azioni.

Nel caso di questa tesi sono stati utilizzati i comandi utili a realizzare l'avvio del software prodotto.

1.4.2 Ambienti

Lo sviluppo del software è avvenuto per la maggioranza in ambiente Linux Ubuntu (kernel 3.2.14), e ne è stato collaudato il funzionamento, oltre che su Linux, anche su Windows XP.

Naturalmente a seconda dell'ambiente è necessario munirsi della java virtual machine adatta e dei driver per LEGO MINDSTORM NXT, liberamente scaricabili dal sito del produttore.

1.4.3 Hardware

L'hardware principale utilizzato in questo progetto è il LEGO MINDSTORM NXT.

L'NXT è il cervello di un robot Mindstorm, si tratta di un blocchetto intelligente e controllato dal computer che consente ad un robot Mindostorm di "prendere vita" ed eseguire differenti operazioni. Oltre al blocchetto programmabile il kit comprende vari pezzi (utili a costruire diversi robot), svariati sensori (luminosità, suono, contatto, infrarossi) e a 3 motori collegabili al blocchetto.

2 Scratch

"Un mio collega mi disse come provò a far interessare alla programmazione la figlia di 10 anni, e l'unica cosa che rispose al suo appello fu Scratch".

Moshe Y. Vardi, Editor-in-Chief of Communications

Scratch è nato proprio dalla speranza di poter fornire un valido approccio per avvicinare alla programmazione persone che non si sarebbero mai immaginate prima nella veste di programmatori.

E' stato realizzato in modo che fosse semplice per chiunque, per tutte le età, i background e gli interessi, per programmare storie interattive, giochi, animazioni e simulazioni, e per condividere le creazioni con chiunque altro.

Dal momento del suo lancio, nel Marzo del 2007, il sito web di Scratch è diventato una comunità online vibrante, con persone che condividono, discutono e riarrangiano ciascuno i progetti degli altri.

Scratch è stato chiamato "lo Youtube dei media interattivi". Ogni giorno, Scratchers da tutto il mondo inseriscono più di 1500 nuovi progetti sul sito, con il codice sorgente liberamente disponibile per la condivisione e la modifica. La collezione di progetti sul sito è molto diversificata: include videogames, newsletter interattive, simulazioni scientifiche, tour virtuali, biglietti di auguri, contest di danza animati e tutorial interattivi, tutti programmati in Scratch.

Il core dell'audience del sito è compreso nell'età dagli 8 ai 16 anni, con picco intorno ai 12, anche se un nutrito gruppo di adulti vi partecipa. Con la programmazione e la condivisione di progetti Scratch, i ragazzi apprendono importanti concetti matematici e computazionali, ma imparano anche a pensare creativamente, ragionare sistematicamente e lavorare in collaborazione: tutte skill essenziali per il XXI secolo.

In realtà l'obiettivo primario di Scratch è quello di nutrire una nuova generazione di creativi, pensatori sistematici utilizzando la programmazione per esprimere le loro idee. [15]

2.1 Scratch

Scratch è sviluppato dal Lifelong Kindergarten Group dei Media Lab dell'MIT, ed è supportato da contributi finanziari della National Science Foundation, di Microsoft, di Intel Foundation, di Nokia e del consorzio di ricerca dei Media Lab dell'MIT.

Il nome Scratch deriva dalla tecnica dello scratching usata dai disc jockey di musica hip-hop che trascinano avanti e indietro con le loro mani i vecchi dischi in vinile per fondere insieme brani musicali in modo creativo. Puoi fare qualcosa di simile con Scratch, mescolando in modo creativo diversi tipi di media (grafica, fotografia, musica, suoni).

Al cuore di Scratch c'è un linguaggio di programmazione grafica che ti permette di controllare le azioni e le interazioni tra media diversi. Programmare in Scratch è molto più semplice che con i tradizionali linguaggi di programmazione: per creare uno script è sufficiente collegare insieme dei blocchi grafici, esattamente come si farebbe con i mattoncini LEGO o con i pezzi di un puzzle. [16]

2.1.1 Imparare con Scratch

Cosa imparano gli studenti quando creano storie interattive, animazioni, giochi, musica e arte con Scratch?

Da un lato, imparano idee matematiche e computazionali integrate nell'esperienza fornita da Scratch. Quando gli studenti creano programmi con Scratch apprendono concetti chiave della computazione come l'iterazione e le condizioni; acquisiscono una comprensione di importanti concetti matematici come le coordinate, le variabili, i numeri casuali; ma, ciò che è più importante, gli studenti imparano questi concetti in un contesto significativo e motivante.

Quando gli studenti lavorano su un progetto con Scratch prendono coscienza anche il ciclo del progettazione. Partendo da un'idea potranno creare un primo prototipo funzionante, sperimentare con esso, correggerlo se qualcosa non funziona, ricevere il parere degli altri, rivederlo e riprogettarlo. E' una spirale continua: partire da un'idea, creare un progetto che porterà a nuove idee, che porteranno ad un nuovo progetto e così via.

Il ciclo della progettazione riunisce molte delle capacità di apprendimento del XXI secolo che risulteranno poi importantissime per avere successo: pensare in maniera

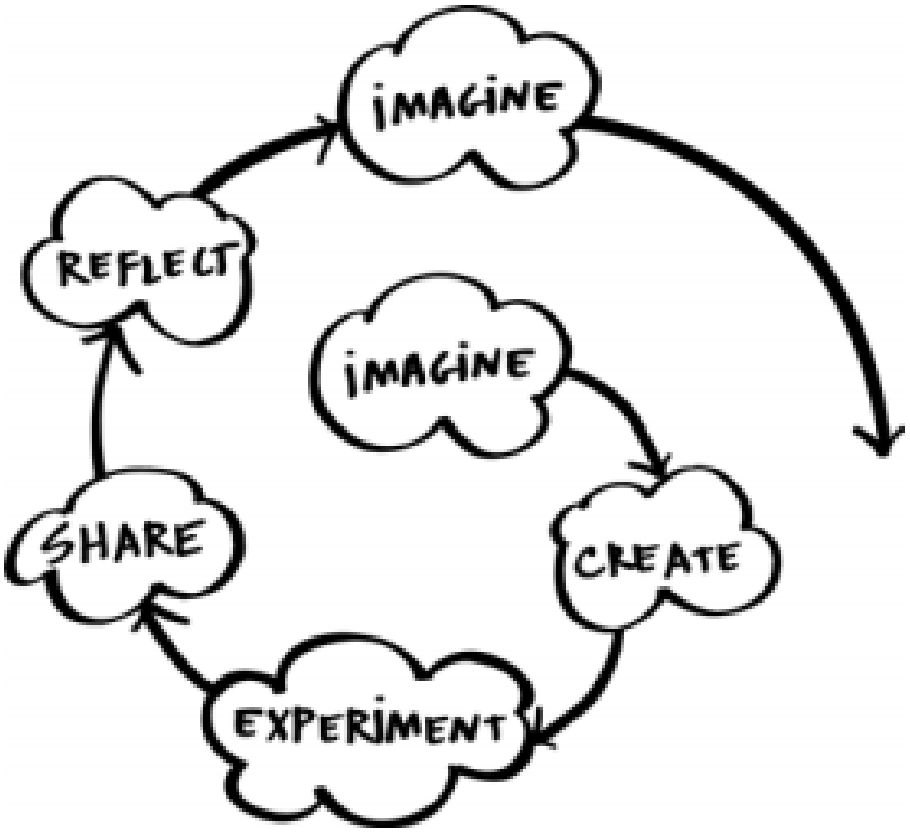


Figura 2.1

creativa, comunicare in modo chiaro, essere in grado di analizzare con sistematicità, saper collaborare con successo, progettare in maniera ciclica, apprendere con continuità.

Creare progetti con Scratch aiuta inoltre gli studenti ad acquisire una più profonda familiarità con le tecnologie digitali. Cosa intendiamo con familiarità? Ad esempio per padroneggiare l'Italiano, l'Inglese o un'altra lingua non basta saper leggere in quella lingua ma occorre anche saper scrivere, occorre cioè sapersi esprimere in quella lingua. Allo stesso modo per essere familiari nelle tecnologie digitali occorre imparare non soltanto ad interagire con il computer ma anche a creare attraverso il computer.

Naturalmente molti studenti non diventeranno dei programmatori professionisti, così come molti di loro non diventeranno degli scrittori professionisti. Ma imparare a programmare offre dei benefici per tutti: rende gli studenti in grado di esprimersi con maggiore completezza e creatività, li aiuta a sviluppare il loro pensiero logico e li aiuta a capire il funzionamento delle nuove tecnologie da cui si trovano circondati nella loro vita quotidiana. [[16]

2.1.2 Programmare con Scratch

Molte persone pensano che programmare i computer sia un'attività noiosa, specialistica, accessibile solo a chi ha avuto un'istruzione tecnica avanzata. E, infatti, i tradizionali linguaggi di programmazione come Java e C++ risultano difficili da imparare per molti.

Al contrario Scratch, nuovo linguaggio di programmazione, ha l'ambizione di cambiare tutto ciò. Scratch: grazie alle maggiori potenze di calcolo dei nostri PC e alle più recenti ricerche nella progettazione di interfacce, infatti, rende la programmazione più coinvolgente e accessibile a bambini, ragazzi e a tutti coloro che stanno imparando a programmare. Le caratteristiche chiave di Scratch includono:

- Programmazione a blocchi. Per creare programmi con Scratch si devono semplicemente assemblare dei blocchi colorati. I blocchi sono disegnati per incastrarsi soltanto se formano sequenze corrette, quindi non esistono gli errori sintattici. Tipi di dati diversi hanno forme diverse, eliminando gli errori di tipo. Si possono modificare i programmi anche mentre sono in esecuzione,

Figura 2.2

cosicché sperimentare nuove idee in maniera incrementale e iterativa risulta semplice.

- Manipolazione dei media. Con Scratch puoi creare programmi che controllano e mescolano grafica, animazioni, musica e suoni. Scratch estende le possibilità di manipolazione dei media così popolari nella nostra cultura ad esempio programmare i filtri di immagini di Photoshop.
- Condividere e collaborare. Il sito di Scratch può fornire ispirazione e visibilità: si possono vedere i progetti creati da altri, riusare e riadattare le loro immagini e i loro script, inviare i propri progetti.
- Scratch non richiede particolari conoscenze per iniziare, offre possibilità di creare progetti complessi e supporto per una grande varietà di progetti. Nello sviluppo di Scratch è stato messo l'accento sulla semplicità, sacrificando talvolta la funzionalità in nome della facilità di comprensione.

Come è già stato sottolineato, quando gli studenti lavorano con Scratch hanno l'opportunità di imparare importanti concetti computazionali come l'iterazione, i condizionali, le variabili, i tipi di dato, gli eventi, i processi. Scratch è stato usato per introdurre a questi concetti studenti di ogni età, dalle elementari fino all'università. Alcuni studenti passano ai linguaggi tradizionali dopo aver imparato la programmazione con Scratch.

Scratch è costruito sul linguaggio Squeak. È stato ispirato da precedenti lavori su Logo e da Squeak Etoys, ma il suo obiettivo è quello di risultare più semplice ed intuitivo.

Scratch è un progetto open-source ma a sviluppo chiuso. Il codice sorgente è liberamente disponibile, ma l'applicazione continua ad essere sviluppata da un piccolo team di ricercatori dei Media Lab del MIT. [[16]]

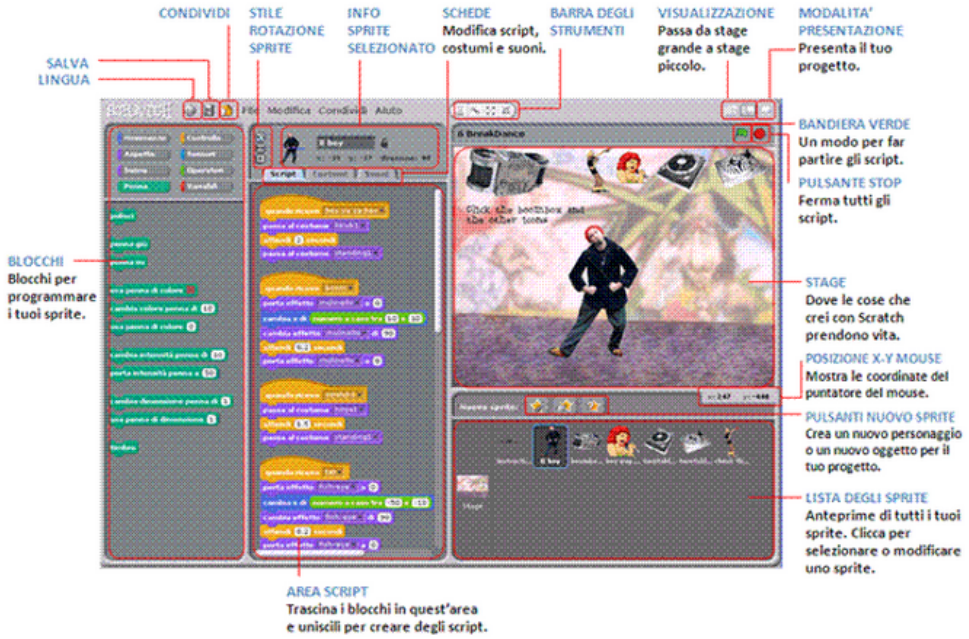


Figura 2.3: L'interfaccia di Scratch

2.1.3 Ingredienti base dei progetti in Scratch

I progetti di Scratch si compongono di oggetti chiamati sprite.

Si può modificare l'aspetto di uno sprite dandogli un diverso costume rappresentato da una qualsiasi immagine e quindi dandogli l'aspetto di una persona o di un treno o di una farfalla o di qualunque altra cosa. Il costume può essere disegnato utilizzando l'Editor di Immagini buil-in, importarne una dal tuo hard disk o trascinarla da un sito web, puoi scattare una foto con la tua webcam (se il tuo PC ne è provvisto).

Per fornire ad uno sprite delle istruzioni, su come muoversi o suonare o reagire agli altri sprite, è necessario collegare insieme dei blocchi grafici in elenchi chiamati script: cliccando su uno di essi Scratch esegue i blocchi dello script dal primo all'ultimo.

2.1.4 Area dei blocchi e area degli script

Per programmare uno sprite è necessario trascinare i blocchi dall'Area dei Blocchi all'Area degli Script. Per eseguire un blocco invece basta cliccarvici sopra.

E' possibile creare degli script (programmi) unendo i blocchi in modo da formare delle sequenze. E' necessario cliccare sulla lista di blocchi per eseguire l'intero script, dal primo all'ultimo blocco.

Per scoprire il funzionamento di un blocco basta cliccare con il tasto destro del mouse (Mac: Ctrl+click) e selezionare "aiuto" dal menu contestuale. Quando si trascina un blocco all'interno dell'Area degli Script, la comparsa di una linea bianca indica i punti in cui è possibile aggiungere il blocco formando un collegamento valido con un altro blocco.

Per spostare uno script, basta "afferrarlo" dal primo blocco in alto. Se si trascina un blocco che si trova al centro di uno script, tutti i blocchi al di sotto di esso si sposteranno insieme a questo. Per copiare uno script da uno sprite ad un altro basta trascinare lo script sull'anteprima dell'altro sprite nella Lista degli Sprite.

Alcuni blocchi hanno delle caselle bianche il cui testo può essere modificato. Per modificare il valore inserito, basta cliccare all'interno della casella e inserire il nuovo valore. E' possibile anche trascinare dei blocchi arrotondati, all'interno di queste caselle.

Alcuni blocchi hanno anche dei menù a discesa, come ad esempio il blocco . E' sufficiente cliccare sulla freccina nera per vedere il menu, quindi cliccare di nuovo per selezionare il nuovo valore.

Per riordinare l'Area degli Script basta cliccarci sopra con il tasto destro del mouse (Mac: Ctrl+click) e selezionare "riordina" dal menu contestuale.

Per esportare un'immagine dell'Area degli Script, bisogna cliccarci sopra con il tasto destro del mouse e seleziona "salva immagine degli script".

Per aggiungere un commento all'Area degli Script, basta cliccare con il tasto destro del mouse (Mac: Ctrl+click) e seleziona "aggiungi commento". Si vedrà comparire l'area gialla del commento e si potrà inserire del testo al suo interno.

Per ridimensionare la larghezza del commento bisogna usare la "maniglia" presente sul suo bordo destro. Basta cliccare il triangolo dell'angolo in alto a destra del commento per collassare o espandere l'area del commento. I commenti possono essere aggiunti in un qualunque punto dell'Area degli Script, e possono essere spostati trascinandoli. Per attaccare un commento ad un blocco (in modo che si sposti insieme al blocco quando questo viene spostato) basta trascinare il commento sopra

Figura 2.4: Pannello di creazione dei blocchi personalizzati.

il blocco. Per staccare il commento dal blocco, basta invece trascinare il commento lontano dal blocco. [17]

2.2 BYOB

BYOB è un'interessante estensione di Scratch che permette di costruire blocchi (Build Your Own Blocks) in forma di procedure/funzioni richiamabili con parametri.BYOB è stato principalmente progettato da Brian Harvey dell'università di Berkley e da Jens Mönig , programmatore tedesco..

Possiede molte caratteristiche che non sono disponibili in Scratch senza apportare larghe modifiche al codice sorgente, tra le quali si annoverano la costruzione di blocchi personalizzati, l'uso di Mesh, liste, procedure e sprite di "prima classe".

Le caratteristiche salienti sono quindi quelle di:

- permettere di costruire dei blocchi personalizzati senza dover mettere mano al codice sorgente di Scratch ma semplicemente utilizzando l'interfaccia grafica messa a disposizione.

- permettere la ricorsione, fino ad un massimo di 6 livelli.

2.2.1 First Class Lists

Un tipo di dato è "first class" in un linguaggio di programmazione se esso può essere:

- il valore di una variabile
- l'input di una procedura
- il valore ritornato da una procedura
- un membro di un dato aggregato
- anonimo (senza nome)

In Scratch 1.4, numeri e stringhe sono first class. È possibile inserire un numero in una variabile, usarne uno come input per un blocco, scrivere un reporter che riporti un numero o inserire un numero in una lista.

Tuttavia le liste di Scratch non sono first class. È possibile crearne una usando il pulsante apposito, che richiede che venga dato un nome alla lista. Non è possibile inserire una lista in una variabile, in un form di input di un blocco o come elemento di una lista (ovvero non si possono avere liste di liste). Nessuno dei reporter di Scratch ritorna valori lista.

Un fondamentale principio di progettazione in BYOB è quello che tutti i dati possano essere first class. Se è nel linguaggio, allora deve essere possibile usarlo liberamente.

Al cuore delle first-class list sta la possibilità di poter creare liste "anonime", ovvero di poter creare liste senza dover fornire contestualmente un nome.

Una lista può essere inserita come elemento in una lista più grande. E possibile cosi creare strutture ad hoc a seconda delle necessità.

Allo stesso modo è possibile costruire qualsiasi struttura dati classica dell'informatica al di fuori delle liste di liste, definendo costruttori, selettori e metodi alla necessità (es binary tree).

In Scratch esistono sostanzialmente tre tipi di input: stringhe, numeri e booleani. In BYOB la tipizzazione è estesa includendo anche le tipologie procedura, lista e oggetto.

Figura 2.5: Rircorsione in BYOB

2.2.2 Ricorsione

Dato che i blocchi personalizzati appaiono nella loro palette appena si inizia ad editarli, è possibile scrivere blocchi ricorsivi trascinando il blocco nella sua stessa definizione. Questa rappresenta l'aspetto forse più interessante di questa estensione, poiché permette di affrontare problemi di natura più complessa e di usufruire quindi di BYOB per studi, o semplici esercizi, più elaborati. Infatti non è raro imbattersi in algoritmi ricorsivi di varia natura.

Ritenendo questa funzionalità interessante e per approfondirne le potenzialità è stato realizzato un semplice applicativo che implementa un algoritmo ricorsivo per disegnare frattali, per ricalcare anche un po la possibilità che dava il LOGO di disegnare forme geometriche in maniera molto semplice, insegnando i concetti della scienza degli elaboratori. Il suddetto applicativo verrà illustrato nei capitoli successivi.

2.3 Scratch e il mondo reale/esterno

Data la sua facilità d'uso e immediatezza, Scratch stimola molto anche per la possibilità di essere connesso ad altri programmi o dispositivi al fine di creare applicazioni più complesse a copertura delle più disparate esigenze didattiche (e non).

2.3.1 RSC/RSP

Il Remote Sensor Connection [18] è una caratteristica di Scratch che permette di collegare a Scratch altri componenti esterni. Questo gli consente di essere esteso per

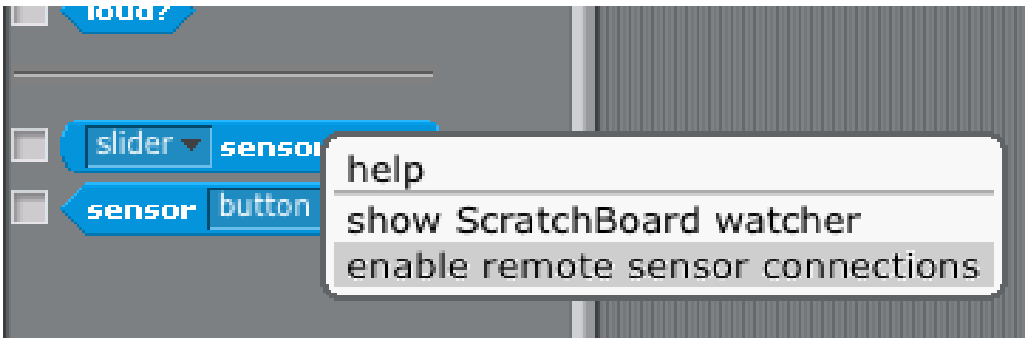


Figura 2.6

connettersi a device, ad internet o per eseguire altre funzioni non possibili all'interno di Scratch.

Per esempio JoyTail [19] permette di utilizzare un joystick con Scratch.

2.3.2 Abilitazione

Per abilitare il RSC è necessario possedere una versione di Scratch uguale o superiore alla 1.3.1, in tal caso basta:

- effettuare click col destro sul blocco Sensor Value, che si trova nella categoria sensing
- selezionare "Enable remote sensor connections", ora scratch è pronto per spedire e ricevere dati..

2.3.3 Come funziona

RSC abilita su Scratch un server locale sulla porta 42001: ogni qualvolta che un broadcast è spedito o una variabile globale è cambiata esso spedisce verso l'esterno un messaggio a tutti i programmi collegati con le informazioni rilevanti sull'evento.

Il protocollo RSP [20], acronimo di Remote Sensor Protocol, è l'estensione sperimentale che abilita l'interazione tra scratch e gli altri programmi.

Nel manuale tecnico si affronterà nel dettaglio il funzionamento e l'implementazione del protocollo.

3 LEGO NXT

Lego Mindstorms NXT è un kit robotico programmabile rilasciato da Lego nel tardo Giugno 2006.[?, ?, ?]

Esso rimpiazza il kit Lego Mindostorms di prima generazione, che veniva chiamato "Robotics Invention System". La base del kit viene venduta in due versioni: la versione Retail[11] e la Education Base Set[12].

Infine una nuova versione del set, il nuovo Lego Mindstorms NXT 2.0, è stata rilasciata il 1 Agosto 2009, con un miglioramento delle capacità e un nuovo sensore di colore.

Proprio quest'ultima versione è quella utilizzata nel corso della tesi, insieme a NXC.

3.1 Hardware

Il brick LEGO MINDSTORM NXT usa varie avanzate componenti elettroniche per fornire le funzionalità dichiarate.

Le caratteristiche tecniche[13] sono le seguenti:

- Processore principale:
  - Atmel® 32-bit ARM® processor, AT91SAM7S256
  - 256 KB FLASH
  - 64 KB RAM
  - 48 MHz
- Co-processor:
  - Atmel® 8-bit AVR processor, ATmega48
  - 4 KB FLASH

  - 512 Byte RAM
  - 8 MHz
- Bluetooth wireless communication
  - CSR BlueCore (TM) 4 v2.0 +EDR System
  - Supporting the Serial Port Profile (SPP)
  - Internal 47 KByte RAM
  - External 8 MBit FLASH
  - 26 MHz
- USB 2.0 communication: Full speed port (12 Mbit/s)
- 4 input ports: 6-wire interface supporting both digital and analog interface
  - 1 high speed port, IEC 61158 Type 4/EN 50170 compliant
- 3 output ports: 6-wire interface supporting input from encoders
- Display: 100 x 64 pixel LCD black & white graphical display
  - View area: 26 X 40.6 mm
- Loudspeaker: Sound output channel with 8-bit resolution
  - Supporting a sample rate of 2-16 KHz
- 4 button user-interface: Rubber buttons
- Power source: 6 AA batteries
  - Alkaline batteries are recommended
  - Rechargeable Lithium-Ion battery 1400 mAH is available
- Connector: 6-wire industry-standard connector, RJ12 Right side adjustment

Per quanto riguarda i motori ed i sensori collegabili:

- i servomotori hanno un sensore di rotazione integrato che permette di ottenere un controllo preciso del movimento del motore, con una precisione di +/- 1°;
- il sensore di contatto permette al sensore di “percepire” e di reagire all’ambiente circostante;

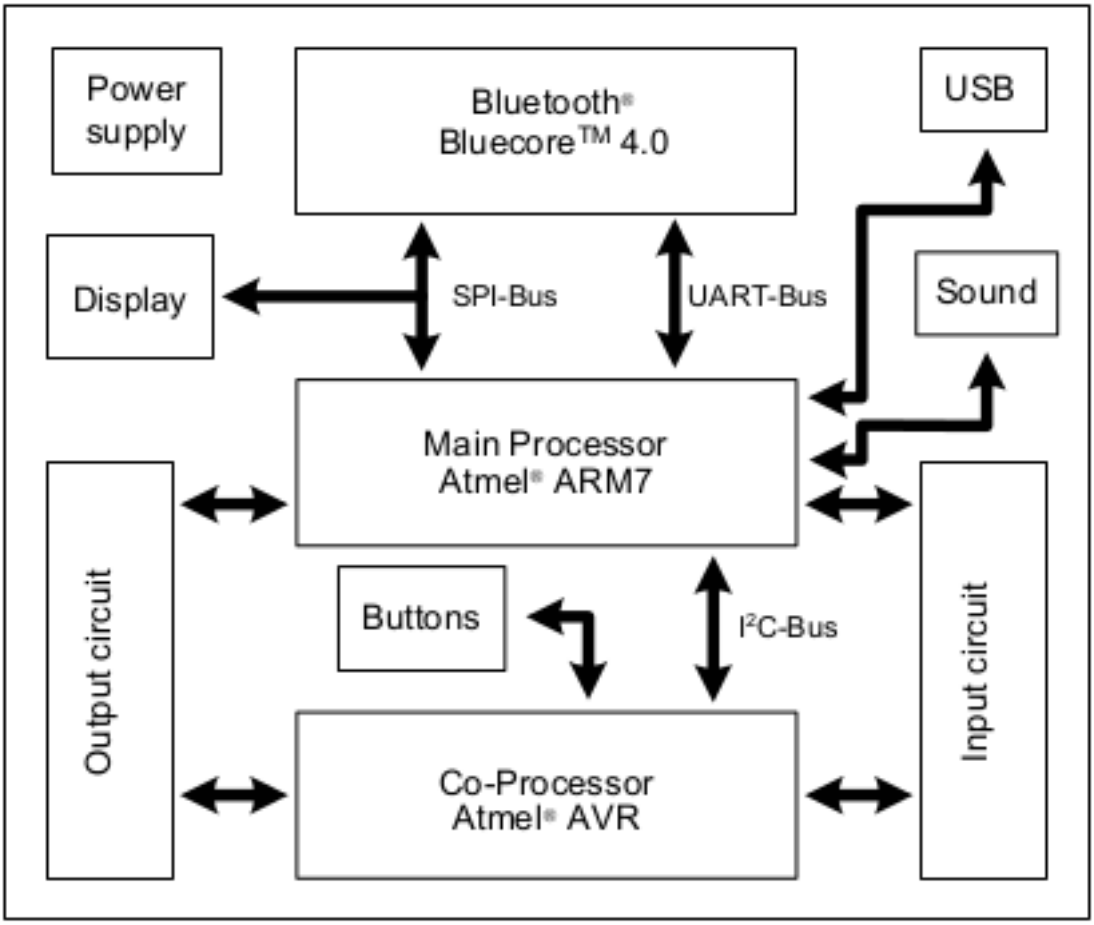


Figura 3.1: Schema hardware del brick NXT

- il sensore ad ultrasuoni funziona come un radar, misurando la distanza e reagendo ai movimenti;
- il sensore di colore può distinguere i colori ed agire come sensore di luminosità, individuando la luce diretta e la luce riflessa.

3.2 Software/firmware

Lego ha rilasciato il firmware per lo NXT Intelligent Brick come software Open Source, insieme agli schemi di tutte le componenti hardware. Questo rende il brick Lego NXT un sistema hardware open source, anche se non è commercializzato come tale.

Il firmware NXT consiste in diversi moduli, inclusi quelli per i sensori, i motori e la virtual machine. Il brick memorizza i programmi accessibili all'utente come file eseguibili, in maniera similare a quello che avviene su un PC. Questi file usano l'estensione .RXE e contengono tutte le informazioni necessarie ad eseguire il programma. In altre parole, un file .RXE rappresenta un programma.

Si può scomporre la struttura del programma in 3 livelli logici: istruzioni bytecode, le informazioni di scheduling e i dati run-time. Il file .RXE contiene le informazioni che rappresentano queste componenti logiche.

Quando la VM esegue un programma, essa legge il file .RXE dalla memoria flash ed inizializza 32KB di memoria RAM riservata per l'uso dei programmi caricati dall'utente. Il file .RXE specifica il layout e il contenuto di default di questo spazio di memoria dedicata. Dopo che lo spazio è inizializzato, il programma è da considerarsi attivo, o pronto per essere eseguito.

Si rimanda alla documentazione SDK[14] per il dettaglio delle specifiche software del brick LEGO NXT. A tal proposito sono disponibili numerosi kit di sviluppo ufficiali:

- Software Delevoper Kit (SDK), include informazioni sui driver USB necessari, il formato dei file eseguibili e una referenza sul bytecode;
- Hardware Delevepoer Kit (HDK), include documentazione e schemi per il blocchetto NXT e i sensori;
- Bluetooth Develpoer Kit (BDK), contiene documentazione sul protocollo usato per la comunicazione bluetooth.

Tutti disponibili sul sito della casa produttrice.

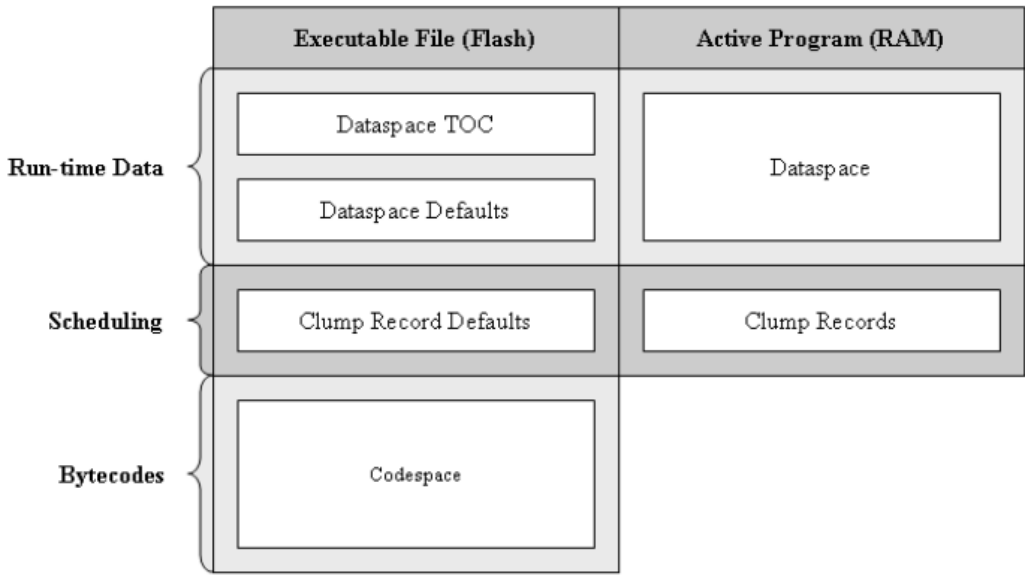


Figura 3.2: La figura mostra una visione logica di come le tre componenti principali di un programma sono divise nelle sub-componenti e come queste sub-componenti sono organizzate mentre il programma è attivo.

3.3 Programmazione

Data la versatilità del sistema e la fornitura delle specifiche, software ed hadware, in open-source, sono moltissimi i progetti nati attorno alla programmazione dei brick. In questo paragrafo verrà data una veloce lista dei software a disposizione.

Linguaggi forniti dalla Lego:

- NXT-G
- RCX Code
- ROBOLAB, basato su LabView e sviluppato alla Tufts University

Linguaggi sviluppati da soggetti terzi:

- GNAT SPL: permette di programmare il brick usando un linguaggio ADA per programmi real-time e embedded
- NXC: programmazione simil-C opensource di alto livello
- NQC: alla base di NXC
- pbFORTH: estensione di Forth
- pbLua: versione di Lua

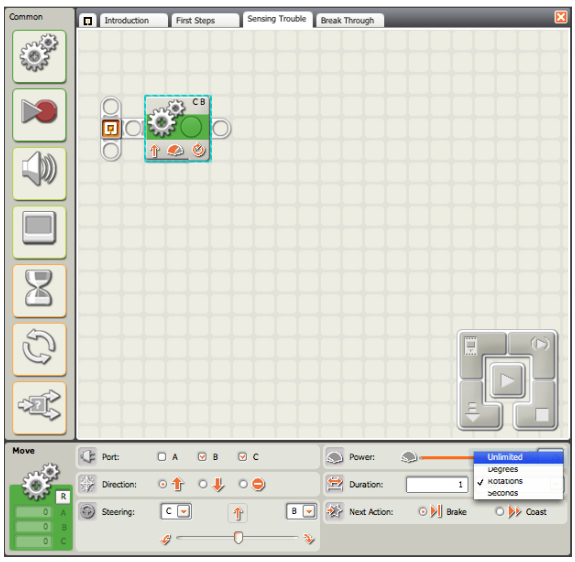


Figura 3.3: NXT-G un esempio del software fornito da LEGO, si può notare lo spazio di lavoro con il blocco che si occupa di impartire il comando di movimento ai motori collegati alle porte CB.

- Visual Basic: attravero l'interfaccia COM+
- LeJOS: un porting di Java

4 Scratch e LEGO NXT

L'idea di far comunicare Scratch con dispositivi LEGO NXT non è originale, naturalmente esistono già altri progetti che si muovono in tale direzione.

Le potenzialità dei due mondi possono sicuramente sommarsi tra loro per raggiungere un nuovo livello alla didattica nei campi delle scienze informatiche.

Nei prossimi paragrafi verranno illustrati un breve stato dell'arte, una delle connessioni più interessanti tra i due mondi, dalla quale si è poi partiti per il lavoro di tesi, ed infine verrà accennato al dimostrativo fatto su BYOB.

4.1 Stato dell'arte

Nativamente esiste LEGO Education WeDo Robotics Kit, un semplice kit di sensori (di distanza e tilt) e attuatori progettato per bambini in età compresa tra i 7 e gli 11 anni. Permette agli utenti di progettare i propri robot, e di programmare i robot usando un software drag-and-drop come lo è Scratch. [21]

La Playful Invention Company (PICO) sviluppa nuove tecologie ed attività per coinvolgere i bambini nell'esperienza dell'apprendimento creativo, fornendo a loro nuove opportunità per esplorare, sperimentare ed esprimere se stessi.

I prodotti PICO sono basati sulla ricerca e sulle idee provenienti dal Lifelogn Kinderfarten froup al MIT Media Lab, struttura leader nel progettare tecnologie educative innovative e ambienti per l'apprendimento creativo.

I ricercatori del Lifelong Kindergarten, in collaborazione con la LEGO Company, hanno creato il primo blocchetto programmabile cercando di ottenere il massimo della potenza computazionale in esso. Questa ricerca ha portato al kit robotico LEGO MINDSTORM, utilizzato ora da milioni di bambini nel mondo per costruire e programmare i loro robot.

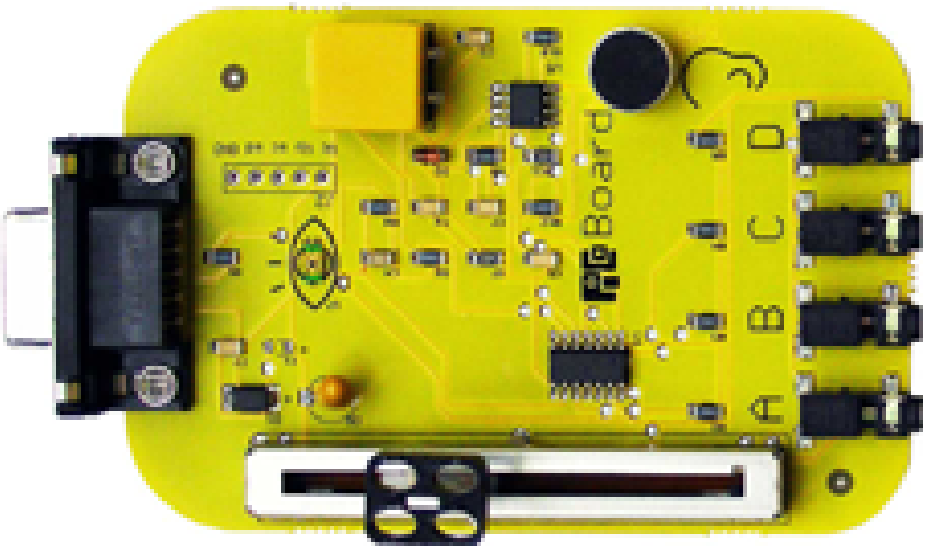


Figura 4.1: Pico Board

Anch'esso basato sulla ricerca del MIT Media Lab, l'ultimo prodotto, il PicoBoard, è una scheda di sensori che lavora con il linguaggio di programmazione Scratch. Con il PicoBoard è possibile connettere sensori reali ai progetti Scratch. Il PicoBoard (in passato Scratch Board) è un scheda che è possibile collegare al computer attraverso le porte USB cosicché i programmi realizzati in Scratch possano interagire con il mondo esterno.

Con PicoBoard, i progetti Scratch possono percepire e rispondere agli stimoli del mondo reale.

Per esempio, usando il sensore di suono, è possibile fare in modo che uno sprite cambi come appare ogniqualvolta vi sia un suono forte. Oppure, usando le letture provenienti dal sensore di luminosità, è possibile programmare lo sprite in modo che saltelli su e giù ogni volta che legge un'ombra, é possibile usare lo slider e il bottone per controllare un personaggio in un video game.

La PicoBoard viene fornita anche con un cavo USB e 4 clip che misurano la resistenza elettrica in un circuito. E' possibile utilizzare queste clip per costruire ogni tipo di sensore personalizzato. Per esempio collegando le clip ad un paio di braccialetti, è possibile determinate quando i polsi si toccano.

Un altro progetto interessante è S4A[29], ovvero Scratch for Arduino, realizzato da Marina Conde, Victor Casado, Joan Güell, Jose García e Jordi Delgado con l'aiuto

Figura 4.2: PICO e i sensori di resistenza elettrica.

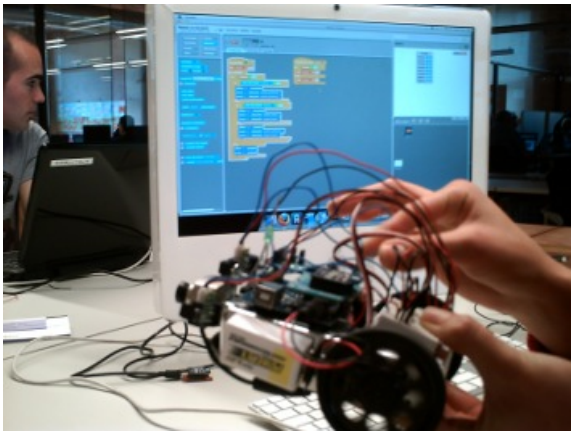


Figura 4.3: S4A

del “Citilab Smalltalk Programming Group” e di collaboratori come Jorge Gómez, istruttore al “Training Center Padre Piquer” di Madrid. Arduino è una piattaforma elettronica per prototipi open-source, basata su un hardware e software flessibile e facile da usare. È pensata per artisti, progettisti professionisti o amatoriali e per chiunque sia interessato a creare oggetti e ambienti interattivi. S4A interagisce con Arduino inviando lo stato degli attuatori e ricevendo periodicamente le informazioni dei sensori: tale scambio di dati è realizzato usando il protocollo PicoBoard che richiede un programma specifico, chiamato firmware, installato sulla scheda.

4.2 Enchanting

Enchanting [22] è un’estensione ’robotica’ di Scratch e si integra con leJOS NXJ (Java per NXT) per la comunicazione e la programmazione del robot LEGO MINDSTORM NXT.

E’ stato sviluppato da Clinton Blackmore, sviluppatore Canadese esperto di videogiochi. Il software sfrutta la programmazione a drag-and-drop e la semplicità di utilizzo di Scratch per integrare blocchi di controllo dei motori e di lettura dei sensori e realizzare cosi un ambiente di sviluppo per robot LEGO MINDSTORM.

In Enchanting sono previste due modalità di utilizzo: interattiva e autonoma .

Nella modalità interattiva il robot è istruito passo-passo su quello che deve fare dal computer e può interagire con gli sprite in Scratch. Per esempio è possibile:

- avere un robot con un sensore ultrasonico con un display che riproduce un sonar;
- aggiustare i parametri (quali velocità o grado di rotazione) con un cursore sullo schermo mentre il programma sta funzionando;
- usare un NXT come un device di input: per esempio si può pensare di costruire una manovella per far girare un motore, e sul computer leggere la posizione dell’encoder del motore per far girare un pesce nello spazio dell’animazione;
- vedere nell’interfaccia la lettura in tempo reale dei sensori collegati;
- o semplicemente utilizzare questo metodo per prototipizzare programmi .

Quando invece si è pronti per programmare un software che giri direttamente sul robot, è possibile esportare il codice e caricarlo direttamente sul robot.

La ragione che sottostà alla decisione da parte di Blatimore di sviluppare una modifica a Scratch con tali caratteristiche è duplice:

1. esistono già molti ambienti di sviluppo che possono essere usati per programmare un NXT e non ha senso reinventarli;
2. si vogliono esporre i bambini all'idea dei programmi testuali, cosicché siano in grado di muoversi verso di essi quando lo vorranno.

Per quanto il programma si presti bene ad un semplice utilizzo, esso presenta alcune limitazioni:

- richiede la sostituzione dell firmware LEGO con quello fornito da leJOS,operazione relativamente delicata;
- mancano blocchi più complessi di comando dei motori più complessi (come lo steering, l'avanzamento di una determinata distanza, rotazione di dati gradi, ...), per la cui realizzazione bisogna procedere a calcoli non alla portata di tutti (o importare una libreria di comandi più complessi);
- la scelta di utilizzare leJOS impedisce che il robot venga utilizzato anche con altri tool che invece sfruttano il firwmare originale.

Al di là di tutto, Enchanting rappresenta al momento l'unico progetto legato a Scratch che permetta una buona interazione con LEGO MINDSTORM NXT; inoltre implementa la possibilità di realizzare programmi ricorsivi grazie all'integrazione con BYOB fornito dalla versione 0.0.9.

Si è pensato fosse opportuno partire dalla realizzazione di un breve dimostrativo che è servito a capire le reali potenzialità di avere la ricorsione a disposizione, e di come funzioni Enchanting nel suo complesso.

4.3 Dimostrativo

Il dimostrativo da realizzare doveva essere tale da permettere di testare le funzionalità del progetto e la semplicità effettiva di Scratch nello sviluppare anche algoritmi non proprio banali.

E' stato scelto di implementare due algoritmi che realizzano rispettivamente la curva di Hilbert e la curva di Koch, due frattali tra i più semplici, ma utili nel testare le funzionalità dell'applicativo e nel contempo la ricorsione fornita da BYOB.

Un frattale è un oggetto geometrico che si ripete nella sua struttura allo stesso modo su scale diverse. Questa caratteristica è spesso chiamata auto similarità. Il termine frattale venne coniato nel 1975 da Benoit Manderlbrot, e deriva dal latino fractus (rotto, spezzato), cosi come il termine frazione; infatti le immagini frattali sono considerate dalla matematica oggetti di dimensione frazionaria. Questo genere di fenomeni nasce dalla definizione di curve od insiemi tramite funzioni o algoritmi ricorsivi.

Ciò che interessa per il nostro dimostrativo è la possibilità di descrivere questa curva con un algoritmo ricorsivo che preveda delle semplici regole che non fanno altro che descrivere degli spostamenti e delle rotazioni di un cursore su un piano cartesiano. L'utilizzo di tale descrizione ricade nell'ambito dei linguaggi di programmazione Logo, linguaggi che attraverso la rappresentazione di una tartaruga come elemento grafico (cursore) permettevano un primo approccio didattico allo sviluppo di software.

4.3.1 La Curva di Koch

La Curva di Koch è una delle prime curve frattali di cui si conosca una descrizione. È apparsa in un documento del 1904 intitolato "Sur une courbe continue sans tangente, obtenue par une construction géométrique élémentaire" del matematico svedese Helge von Koch.

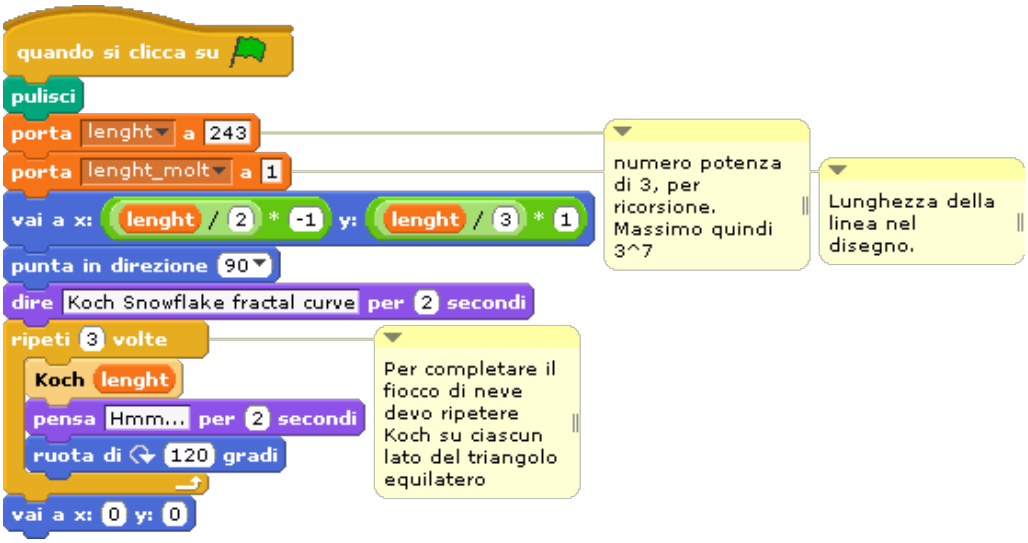
L'algoritmo della curva di Koch è molto semplice, consiste in un ripetizione del ciclo seguente.

Partendo da un segmento di determinata lunghezza:

1. dividere il segmento in tre segmenti uguali;
2. cancellare il segmento centrale, sostituendolo con due segmenti identici che costituiscono i due lati di un triangolo equilatero;
3. tornare al punto 1 per ognuno degli attuali segmenti.

Nell'applicativo è stato implementato in realtà quello che viene conosciuto come Il fiocco di neve di Koch, ovvero la figura che si ottiene applicando l'algoritmo di Kock ai lati di un triangolo, figura iniziale del fiocco.

Con l'applicazione ricorsiva dell'algoritmo su ciascun dei 3 lati si ottiene una figura frattale che ricorda un fiocco di neve.



(a)

(b)

Figura 4.4: Implementazione dell'algoritmo di Koch, adattato alla realizzazione su Scratch/BYOB. (a) Procedura principale (b) procedura ricorsiva

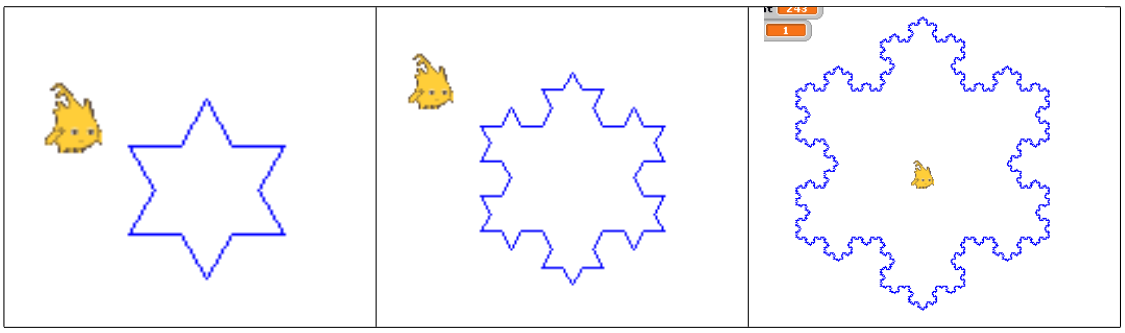


Figura 4.5: Tre livelli di applicazione dell'algoritmo implementato.

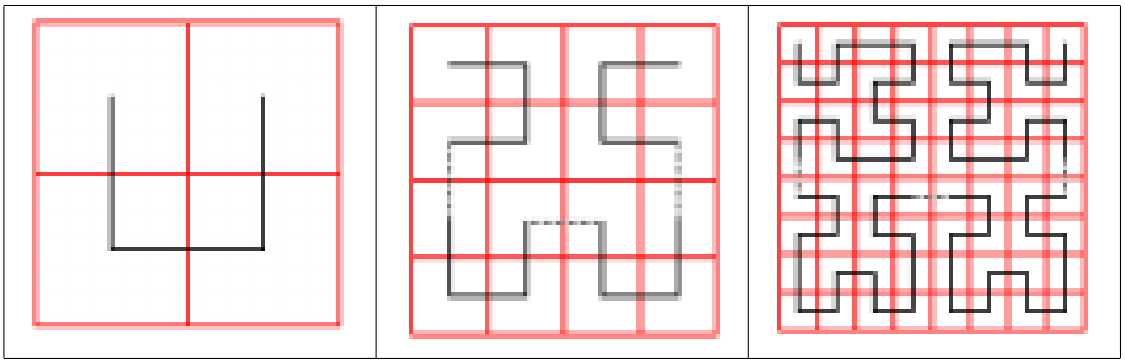


Figura 4.6: Esempio 3 applicazioni dell'algoritmo.

4.3.2 La curva di Hilbert

La curva di Hilbert è un frattale descritto per la prima volta dal matematico tedesco David HIlbert nel 1891, come una variante della curva descritta dal Giuseppe Peano nello stesso anno.

E' chiamata "space-filling curve", in quanto è in grado di riempire un piano dopo diverse iterazioni.

I passi dell'algoritmo sono semplici:

1. si parte da un quadrato suddiviso in 4 parti uguali, nel quale si disegna una linea retta spezzata che unisce i 4 centri dei quadrati interni
2. ognuno dei 4 quadrati viene diviso in 4 quadrati uguali e si ripete lo stesso procedimento al punto 1. Si collega infine i nuovi segmenti per ottenere la curva al passo successivo.

Come nel caso precedente non è interessante tanto il frattale quanto la sua implementazione ed esecuzione in BYOB, che risultano semplici e molto immediate una volta che si è analizzato il problema e immaginato l'algoritmo.

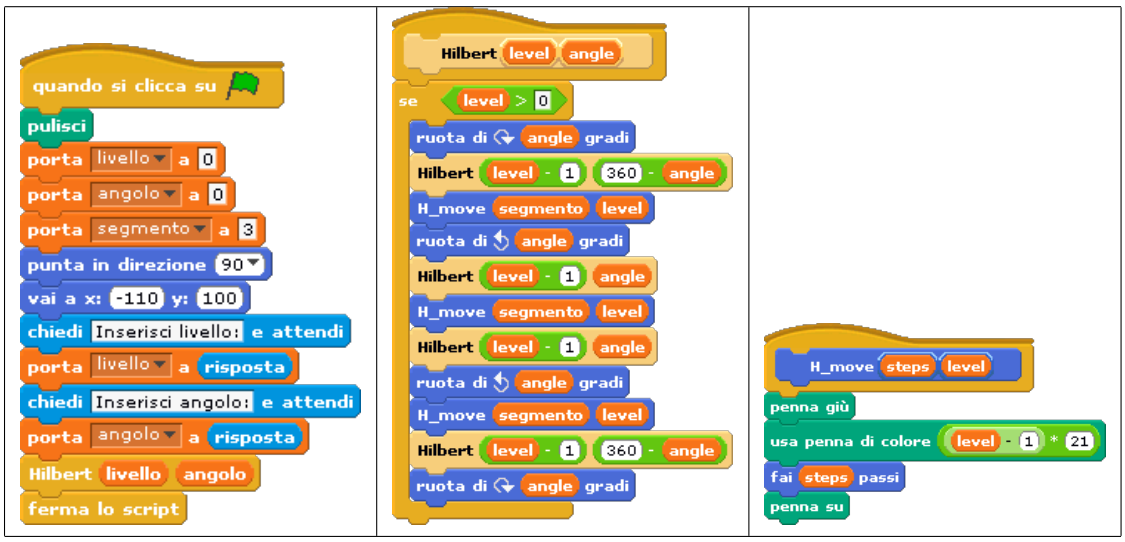


Figura 4.7: Implementazione in BYOB

Data la natura molto semplice dei movimenti, specialmente delle rotazioni ad angolo retto, questo algoritmo si è prestato bene ad un'implementazione con risultati più accurati sul robot NXT.

4.3.3 Risultati

In entrambi i dimostrativi è stata dapprima realizzata una version in BYOB, per simularne il funzionamento sullo sprite per poi riportare l'implementazione su Enchanting traducendo le varie componenti di movimento dello sprite in movimenti dei blocchi motore del veicolo costruito.

Per l'appunto per il dimostrativo in Enchanting è stato costruito un veicolo con LEGO MINDSTORM NXT dotato di due ruote motrici sullo stesso asse e una ruota libera.

Effettivamente Enchanting, ovvero Scratch, si è dimostrato uno strumento davvero semplice per la realizzazione degli algoritmi, con il drag-and-drop dei blocchi funzionali è immediato costruire il flusso logico di operazioni che implementano i due algoritmi, esaltandone se possibile anche l'espressività e la comprensione senza per questo inficiare di libertà implementativa.

Un vincolo realizzativo degli algoritmi ricorsivi è stata la "sensibilità" di movimento del robot, che con un algoritmo come quello di Koch si è dimostrata insufficiente per

Figura 4.8: Veicolo costruito per il dimostrativo.

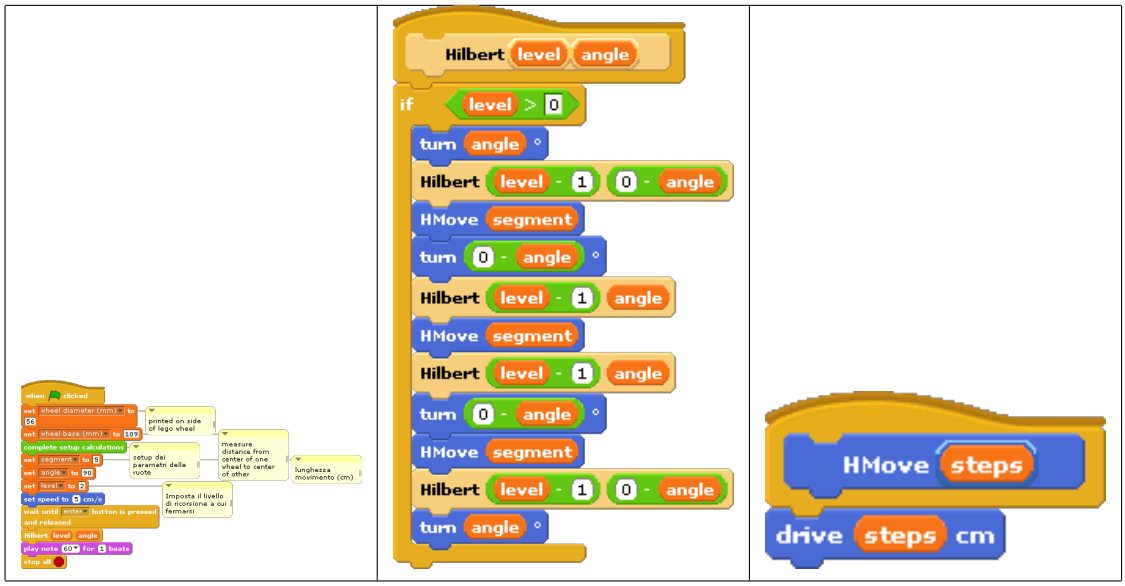


Figura 4.9: Implementazione della curva di Hilbert in Enchanting.

le numerose rotazioni ad angoli “difficili” fin dalle prime ricorsioni. Per questo si è preferito concentrarsi sull'implementazione della curva di Hilbert su Enchanting.

Un altro vincolo riscontrato è nel numero di chiamate ricorsive, vincolo interno allo sviluppo di BYOB, che limita le stesse ad una massima profondità di 6. La limitazione non ha comunque costretto il dimostrativo in paletti troppo stretti in quanto, in particolare nella realizzazione pratica col robot, ricorsioni troppo profonde portavano il robot a carichi computazionali troppo alti e a movimenti imprecisi dovuti ai numerosi e piccoli cambi di rotazione richiesti.

Per quanto si siano volute testare le funzionalità più avanzate, come appunto la ricorsione, su Enchanting 0.0.9, le necessità che un ambiente del genere ha, dato anche il suo target medio, non coprono di certo algoritmi ricorsivi. Per il proseguo del lavoro si è quindi deciso di abbandonare Enchanting e di partire dalla versione base di Scratch, anche per avere un sistema più pulito eventualmente espandibile in futuro con i moduli BYOB specifici.

Rimane comunque interessante la possibilità di utilizzare l'interfaccia grafica e il box dell'animazione come una sorta di anteprima di ciò che viene realizzato dall'algoritmo implementato nello sprite. Aspetto interessante da tenere sicuramente presente per eventuali sviluppi successivi del progetto.

5 SILENT

SILENT è l'acronimo dato all'applicativo realizzato in questa tesi: Scratch Interface for LEGO NXT.

Nel capitolo seguente verranno illustrate le motivazioni che hanno portato alla realizzazione del software, a partire dalle considerazioni fatte nei capitoli precedenti, le problematiche affrontate, l'idea generale del sistema e le tecnologie applicate nel corso dello sviluppo.

5.1 Motivazioni

Dopo aver testato a fondo Enchanting si sono delineati i seguenti punti critici:

- mancanza dei comandi standard per pilotare i servo motori: per esempio mancano comandi per lo steering o per la messa in moto sincronizzata (o meno), comandi per impartire movimenti di una determinata distanza, sia essa in gradi, rotazioni delle ruote o spazio percorso;
- utilizzo di un firmware diverso da quello originale, che comporta la sovrascrittura dello stesso, un'operazione magari complessa e che comunque comporta dei rischi, specialmente se eseguita da utenti inesperti. Inoltre questa operazione di flash del firmware con quello LEJos comporta la perdita della possibilità di utilizzare il brick con i software di programmazione più comuni;
- la traduzione dal linguaggio "grafico" utilizzato in Scratch al codice "Java" di LEJos avviene in maniera abbastanza contorta sfruttando il motore di traduzione interno di Scratch, cosa che rende difficile l'eventuale adattamento del software alla traduzione in altri linguaggi.

Per questo, e per le considerazioni fatte nei capitoli precedenti, si è deciso di sviluppare una modifica a Scratch che permettesse di interfacciarsi "nativamente" al Brick

LEGO NXT, evitando quindi di dover sovrascrivere il firmware originale (a meno di aggiornamenti), di implementare le principali funzioni di controllo disponibili a livello di programmazione testuale, di mantenere un alto livello di utilizzo ma che permetta di scendere nei dettagli per analizzarli e che richieda competenze specifiche solo per il primo setup del modello del robot e che possa essere espansa/modificata in futuro.

5.2 Idea generale del sistema

La scelta è stata quella di ripartire dalla versione originale di Scratch, quella scaricabile dal sito ufficiale, modificandolo affinché realizzi le necessità di interfacciamento con l'utente e quelle di esportazione della rappresentazione grafica del programma, in modo da metterlo in comunicazione ad un software esterno, che in background si occupi sia di comunicare con Scratch e con il robot LEGO NXT sia di effettuare la traduzione del programma realizzato dall'utente da un formato di scambio a quello specifico del firmware NXT.

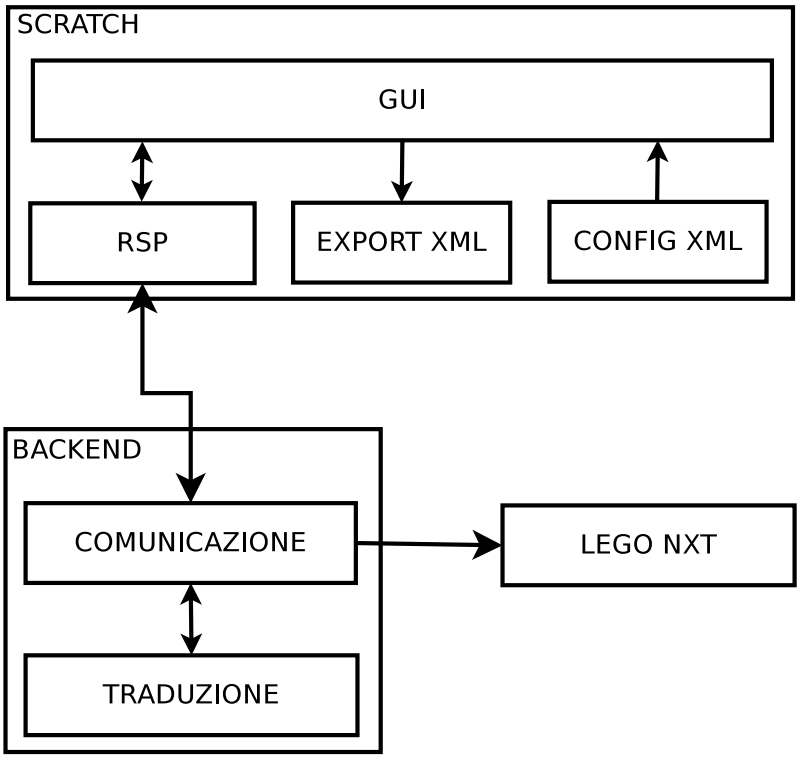


Figura 5.1: Schema dell'architettura di SILENT

L'idea generale del sistema è semplice, almeno in una visione ad alto livello; i suoi componenti fondamentali sono:

- Scratch: modificato nella sua interfaccia per rendere i blocchi funzionali alla programmazione di un robot LEGO NXT e nella possibilità di tradurre il programma realizzato negli sprite in un formato di scambio quale è XML.
- Backend di comunicazione: permette di comunicare con Scratch, con il motore di traduzione e con il Brick NXT collegato. Si occupa in sostanza di ricevere ed inviare messaggi da e verso Scratch o il Brick LEGO NXT. Si occupa inoltre della compilazione e/o upload del codice verso il robot.
- Core di traduzione: traduce la rappresentazione del programma ricevuta da Scratch in formato XML, in codice NXC.

5.3 Problematiche affrontate

Le sfide affrontate nella realizzazione di SILENT sono state innanzi tutto di carattere decisionale ed implementativo.

5.3.1 Sfide decisionali

Le scelte decisionali riguardano quali modifiche apportare all'interfaccia utente di Scratch, ovvero quali blocchetti conservare, modificare, cancellare od aggiungere.

Molti dei blocchi risultano inutili ai fini della programmazione del brick NXT (quali i looks, pen, sensing, sound e motion) e vanno sostituiti con altri più adeguati (in looks, sensing, sound e motion andranno sistemate le funzioni rispettivamente per l'output su display, i sensori, la produzione di suono e il movimento dei motori).

Un altro aspetto importante è stata la scelta su quali funzioni inserire nell'interfaccia, ovvero fino a quale livello di dettaglio rendere disponibili le funzionalità di NXC a livello di interfaccia utente, tenendo sempre presente che il software realizzato andrà poi utilizzato da bambini in età scolare. In questo caso si è scelto di dare un controllo completo sulle principali funzioni di movimento dei motori e di lettura dei dati da sensori. Oltre alle semplici funzioni per far produrre suoni e scrivere output

su schermo. Infatti per quanto dettagliate, tali funzionalità non risultano troppo complesse, permettono un maggior controllo sui movimenti ed infine ricalcano le possibilità date da altri software di programmazione per NXT, come ad esempio lo stesso Mindstorms Edu NXT, software ufficiale della LEGO.

Nel capitolo sucessivo verranno illustrate le scelte fatte.

5.3.2 Sfide implementative

Le sfide tecniche riguardano in sostanza la necessità di modificare e/o implementare software con diversi linguaggi e di metterli in comunicazione tra loro.

Scratch, Squeak Per modificare Scratch si è reso necessario lo studio di Squeak, linguaggio in cui è sviluppato, attraverso il sito ufficiale e i tutorial messi a disposizione.

Si è passati poi allo studio dei plugin, realizzati dalla comunità di utenti, che permettessero di effettuare un'esportazione del programma realizzato in Scratch in un formato che potesse essere utilizzato come interfaccia con l'esterno, nel nostro caso XML.

Con gli strumenti acquisiti e i riferimenti presenti sul sito ufficiale è stata analizzata la struttura interna di Scratch in modo da comprendere quali parti di codice modificare per poter ottenere il risultato voluto, ovvero blocchi personalizzati, modellazione del robot in uso, esportazione del codice in XML e comunicazione con l'esterno.

Infine si è reso necessario comprendere come venissero implementate le variabili e liste in Scratch, in modo da poter effettuare delle scelte sulla loro traduzione in codice NXC ed eventualmente sulla semplificazione del loro utilizzo. In Scratch infatti esistono sostanzialmente soltanto due tipi di variabili: stringhe e numeri. Le liste, non avendo una tipizzazione, possono contenere sia stringhe che numeri, senza limitazioni.

Per ricondurre il tutto al modello di variabili di NXC, e per semplificare la traduzione, in questa prima implementazione si sono associate le variabili al tipo string e float, rispettivamente, mentre sono state eliminate le liste e sono stati inseriti gli Array come sono conosciuti in NXC, quindi con una loro tipizzazione (string o float) ed una dimensione fissa.

Java, Bash, NXC Per sviluppare il backend di comunicazione ed il core di traduzione sono stati adottati tre linguaggi differenti: Java, Bash scripting (Linux/Windows) e NXC. Nel capitolo 8 verranno illustrati i dettagli implementativi relativi a ciascuno di questi aspetti, per ora ci si limita ad una esposizione descrittiva.

Java è stato utilizzato per la realizzazione delle parti più importanti del software esterno.

La prima si occupa di comunicare con Scratch attraverso l'implementazione del "Remote Sensor Protocol"[20], utilizzando quanto presentato su "Killer Game Programming in Java"[23] nel capitolo 13.7 e in Enchanting. Essa comunica quindi con Scratch e impartisce i "comandi" di traduzione, compilazione o compilazione e download del codice.

La seconda si occupa di effettuare la traduzione nel codice NXC: riceve l'XML da Scratch e, scorrendo l'albero che rappresenta il flusso del programma, traduce i blocchi funzionali in codice NXC .

Il linguaggio di scripting Bash è stato utilizzato per rendere portabile su diversi sistemi operativi, per ora Windows e Linux, le procedure:

- di avvio di SILENT: con l'impostazione delle variabili d'ambiente corrette, l'avvio del software java e della versione modificata di Scratch
- di compilazione NXC: attraverso l'invocazione di "nbc", software che si occupa di compilare e spostare sul robot il codice NXC prodotto.

Il linguaggio NXC infine è stato studiato per poter effettuare la conversione dal formato XML del programma, non soltanto per ottenere una traduzione corretta ma anche per la scelta di come tradurre i costrutti di Scratch, in particolare per quanto riguarda le variabili e gli Array.

5.4 Tecnologie utilizzate

Le tecnologie utilizzate sono quindi:

- Squeak: attraverso la virtual machine che viaggia incorporata con l'immagine è possibile effettuare modifiche "al volo" al codice sviluppato per Scratch e vederne gli effetti immediatamente senza bisogno di ricompilare. Inoltre ogni immagine girerà su qualsiasi interprete anche se essa è stata salvata in hardware completamente diversi e con differenti OS.

- JAVA: sfruttando la sua portabilità e la sua facilità di implementazione è la tecnologia che più si adatta allo sviluppo della parte esterna a Scratch. Le classi principali utilizzate sono quelle che si occupano di:
  - gestire connessioni di rete TCP: utilizzate per comunicare con Scratch
  - gestire XML: attraverso la navigazione tra i nodi
  - leggere/scrivere file

  L'ambiente di sviluppo scelto è stato Eclipse e naturalmente JDK, aggiornato all'ultima versione.
- BASH: per gestire l'avvio del software su diversi SO
- Brick NXT: unità di elaborazione dei robot LEGO che permette di collegarsi tramite interfaccia USB o Bluetooth (nel nostro caso è stata utilizzata soltanto la connessione USB).
- Oracle VirtualBox: è un prodotto di virtualizzazione disponibile sia per uso professionale che per uso privato. E' disponibile una versione open-source sotto i termini della GNU General Public License (GPL) versione 2. Nel corso di questo lavoro è stato utilizzato per virtualizzare un sistema operativo Windows XP nel quale sono state effettuate le prove per SILENT e il dimostrativo in BYOB.

6 Realizzazione SILENT

In questo capitolo verranno affrontati in maniera più approfondita gli aspetti legati allo sviluppo dell'applicativo SILENT, tenendo come riferimento lo schema a blocchi della Fig. 5.1. Si rimanda comunque al manuale tecnico per i dettagli di carattere implementativo.

6.1 XML Configurazione del robot

Una delle necessità per un applicativo che permetta di programmare un robot è sicuramente quella di poter configurare il "modello" di robot a disposizione per la sessione di lavoro.

Data l'architettura hardware messa a disposizione dal kit LEGO MINDSTORM NXT, i componenti di cui è necessario specificare la configurazione sono i motori e i sensori, unici dispositivi che hanno la possibilità di essere collegati o meno al brick.

Per ciascuno di essi è opportuno definire:

- un LABEL: non è altro che un'etichetta di testo da associare al dispositivo, lo identificherà sull'interfaccia utente e nel codice prodotto.
- un PORT: rappresenta la porta alla quale il dispositivo è collegato. Il brick LEGO mette a disposizione 3 porte per i motori (A,B,C) e 4 porte per i sensori (1,2,3,4).
- un TYPE: campo che specifica la tipologia di dispositivo collegato, per i motori esso è stabilito a TA, mentre per i sensori esso dipende dalla tipologia, può essere touch, sound, light, ultrasonic.

La specifica della configurazione avviene in un file XML denominato "robotmodel.xml", nel quale vengono raccolti in un albero le informazioni relative al modello che si vuole definire.

```
<?xml version="1.0" encoding="UTF-8"?>
    <robotmodel>
        <motors>
            <motor>
                <label>MotorA</label>
                    <port>A</port>
                    <type>TA</type>
            </motor>
            <motor>
                <label>MotorB</label>
                <port>B</port>
                <type>TA</type>
            </motor>
    </motors>
    <sensors>
        <sensor>
            <label>Touch</label>
            <port>1</port>
            <type>touch</type>
        </sensor>
        <sensor>
            <label>Sound</label>
            <port>2</port>
            <type>sound</type>
        </sensor>
        <sensor>
            <label>Light</label>
            <port>3</port>
            <type>light</type>
        </sensor>
        <sensor>
            <label>Ultrasonic</label>
            <port>4</port>
            <type>ultrasonic</type>
        </sensor>
    </sensors>
</robotmodel>
```

Attraverso un semplice file XML quindi si definisce che cosa e dove è collegato al brick, spetterà poi a SILENT caricare la configurazione, dopo averla validata.

Per verificare la validità di quanto scritto in robotmodel.xml esso viene infatti sottoposto ad un processo di "well-formness" e ad una validazione attraverso un file di grammatica XSD [http://www.w3schools.com/schema/default.asp], il tutto prima di avviare il programma vero e proprio.

Il processo di well-formed controlla che non ci siano errori di sintassi nel file XML, mentre la grammatica XSD controlla che il file associatogli rispetti le regole definitevi.

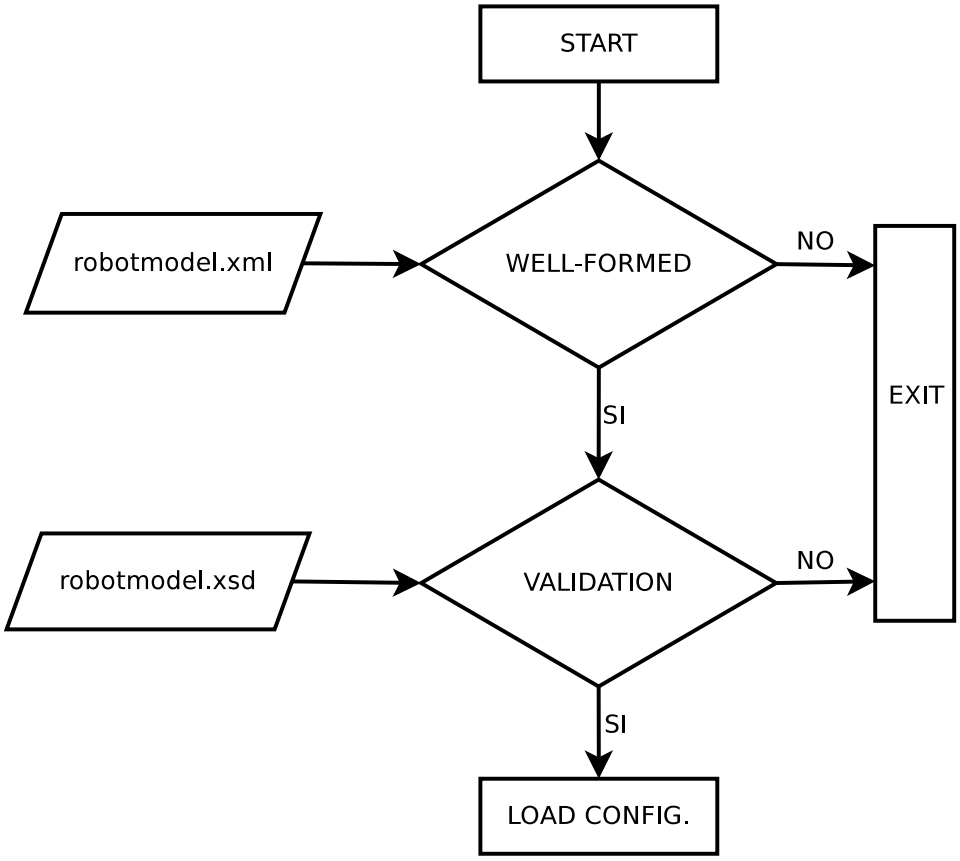


Figura 6.1: Schema di flusso del processo di check e validazione del file di configurazione del robot.

La grammatica XSD controlla che:

- vi sia un nodo radice "robotmodel"
- vi sia un nodo "motors", contenente minimo 0, massimo 3 nodi "motor", ciascuno dei quali contenente 3 nodi: label, port e type definiti come string

- vi sia un nodo “sensors”, contenente minimo 0, massimo 4 nodi “sensor”, ciascuno dei quali contenente 3 nodi: label, port e type definiti come string

Di seguito il listato della grammatica utilizzata:

```
<?xml version="1.0" encoding="UTF-8"?>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       elementFormDefault="qualified">
    <xs:element name="robotmodel">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="motors">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="motor" minOccurs="0"
                 maxOccurs="3">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="label" type="xs:string
                       "/>
                    <xs:element name="port" type="xs:string"
                       />
                    <xs:element name="type" default="TA">
                      <xs:simpleType>
                        <xs:restriction base="xs:string">
                          <xs:enumeration value="DC"/>
                          <xs:enumeration value="TA"/>
                        </xs:restriction>
                      </xs:simpleType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="sensors">
```

```
            <xs:complexType>
              <xs:choice>
                <xs:element name="sensor"  minOccurs="0"
                    maxOccurs="4">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="label" type="xs:string
                          "/>
                           <xs:element name="port" type="
                              xs:string"/>
                           <xs:element name="type" type="
                              xs:string"/>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
              </xs:choice>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
</xs:schema>
```

6.2 Modifiche a Scratch

Come accennato in precedenza, buona parte del lavoro è stato svolto nel modificare Scratch, nello specifico la versione 1.4. Le modiche hanno interessato i seguenti aspetti:

- blocchetti per la programmazione: sono stati sostituiti tutti i blocchi di movimento, “sensing”, “looks”, “sound” e “variable”. La palette “pen” è stata sostituita da “advanced”, che è stata pensata come spazio dove inserire blocchetti per funzioni più avanzate. Interessati da minori cambiamenti invece sono state le palette “operation” e “control”, in quanto tutte composte da blocchi in larga parte riutilizzabili allo scopo del progetto.

- caricamento configurazione robot: questo aspetto è quello che si occupa di caricare il file “robotmodel.xml”, effettuarne il parsing e ricavarne quindi le informazioni di configurazione dei blocchi relativi al robot. Tali informazioni si traducono nella disponibilità o meno di blocchi “sensing” e “motors”, questo viene gestito dalla classe “RobotModel” in SILENT che viene inizializzata con i dati ricavati dal file di configurazione ed espone all’interfaccia utente i blocchi relativi.

- esportazione degli script: per esportare tutto quello che viene realizzato dall’utente in Scratch, ovvero tutti gli Script presenti in tutti gli Sprite, è stato sfruttato un modulo “XMLexporter” incluso nel mod di Scratch CHIRP [27], modificandolo in modo che potesse esportare il complesso di informazioni di cui si aveva bisogno. Il modulo originario infatti permette di esportare soltanto singoli script. La decisione di utilizzare il formato XML come interfaccia tra il mondo rappresentato in Scratch e l’esterno è stata dettata dalla necessità di separare i due livelli, in modo da poter rendere in futuro espandibile e/o modificabile ciascuno dei due aspetti (Scratch e il Backend) separatamente.

- comunicazione con il mondo esterno: implementazione dell’RPC richiede la realizzazione di classi lato Scratch che si occupino inviare i messaggi di esportazione, compilazione e compilazione/download verso il backend. Queste semplici classi sono state poi collegate al menu “File” in modo che l’utente possa comunicare al backend la volontà di effettuare tali operazioni.

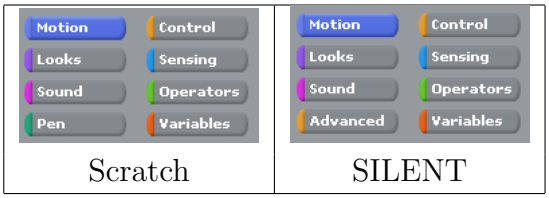


Figura 6.2: Cambiamenti nel blocco selettore delle palette.

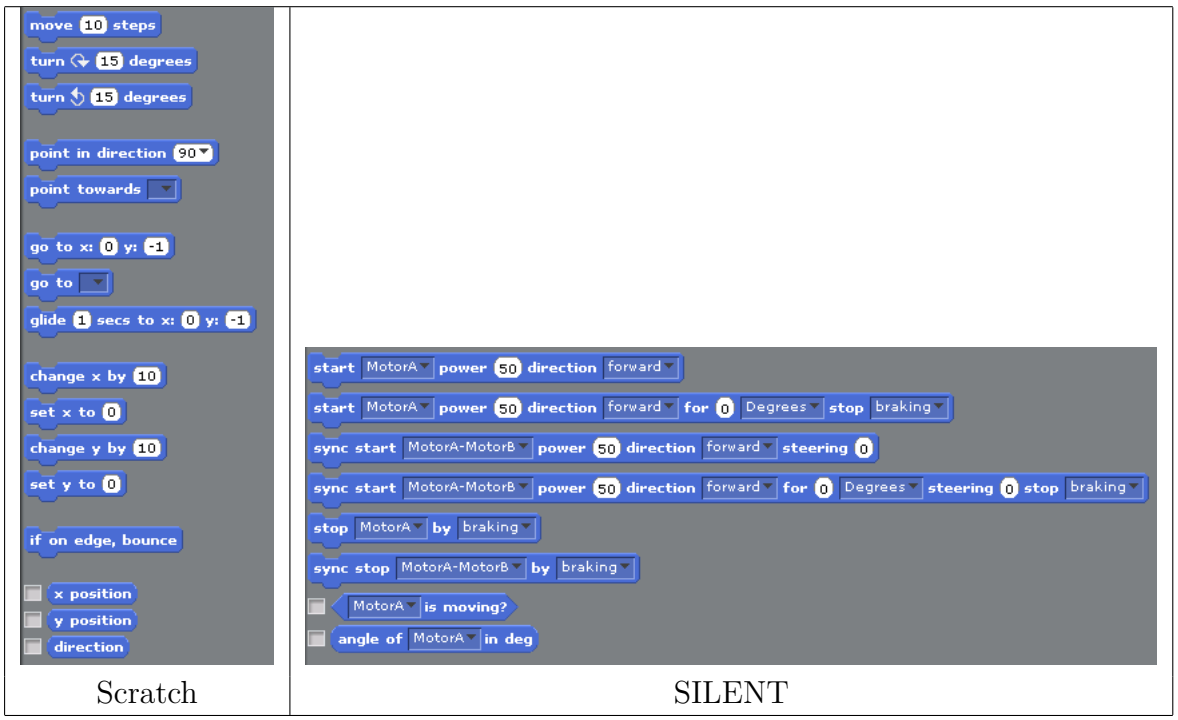


Figura 6.3: Cambiamenti alla palette “Motors”

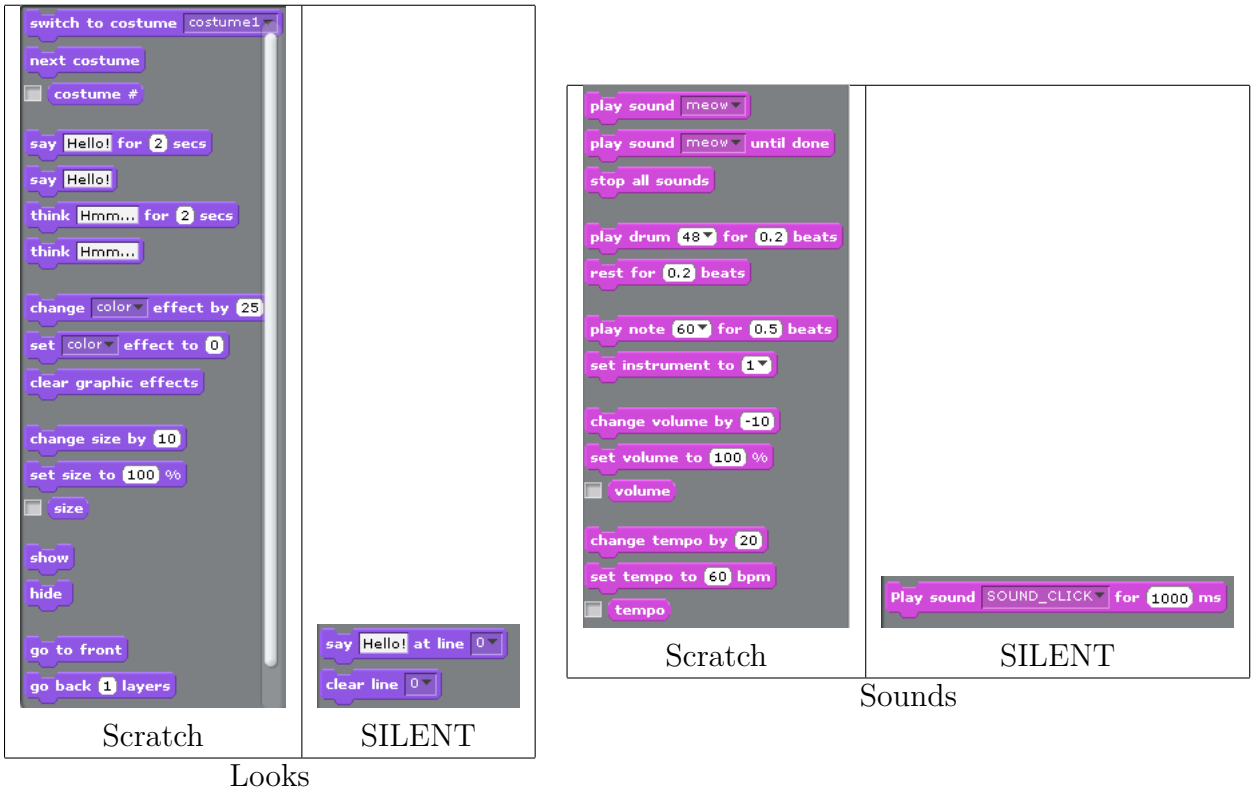


Figura 6.4: Cambiamenti alle palette “Looks” e “Sounds”

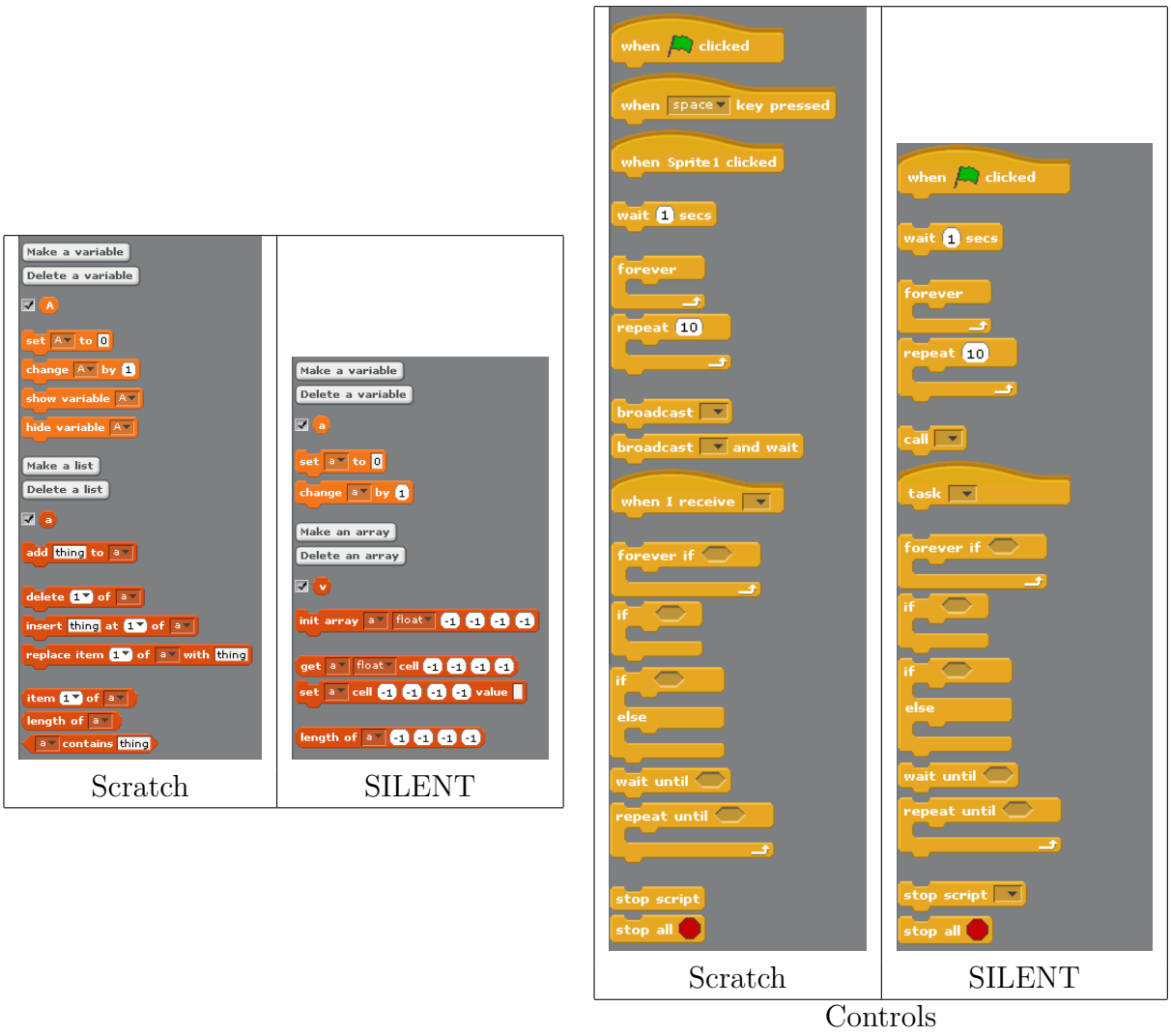


Figura 6.5: Cambiamenti alle palette “Variables” e “Controls”

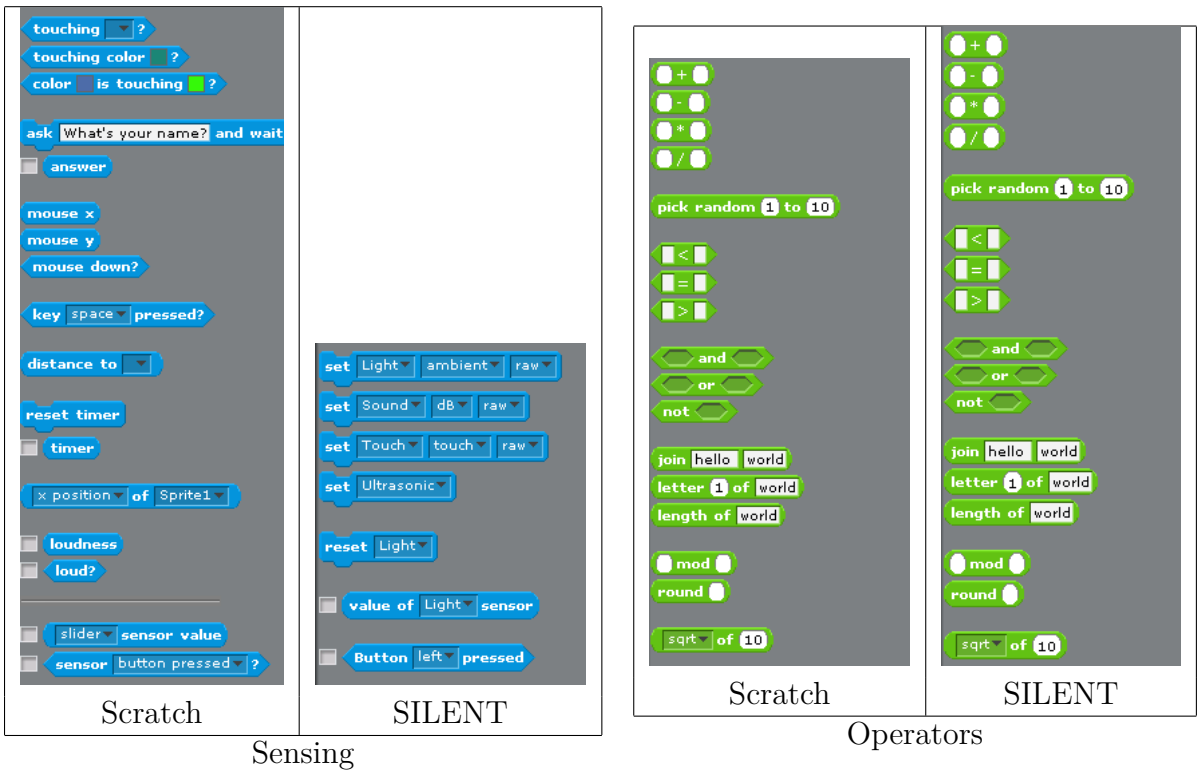


Figura 6.6: Cambiamenti alle palette “Sensing” e “Operators”

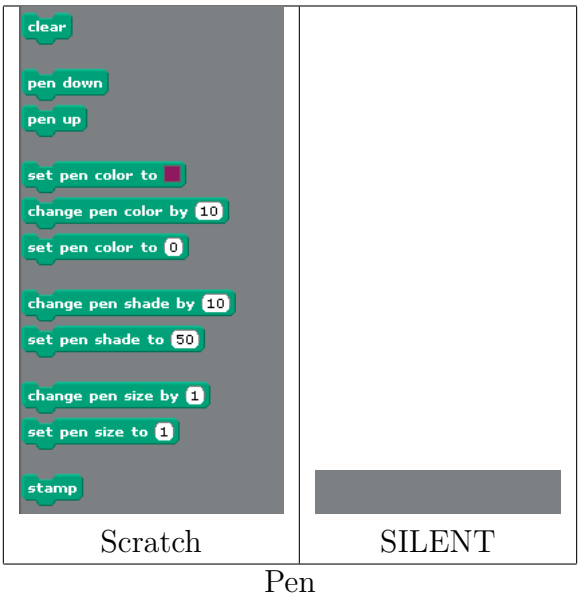


Figura 6.7: Cambiamenti alla palette “Pen”

6.3 Backend

Il backend è il motore vero e proprio di SILENT infatti si occupa di: eseguire tutte quelle operazioni che permettono di realizzare l'interfaccia tra Scratch ed il dispositivo LEGO NXT.

Le sue componenti funzionali fondamentali sono le seguenti:

- avviare SILENT: ricezione parametri dagli script bash, controllo di wellness e validazione attraverso il file di grammatica del file XML di configurazione del robot collegato
- gestire la comunicazione tra i vari componenti del sistema: scratch, traduttore, LEGO NXT
- tradurre il codice ricevuto sotto forma di XML in codice NXC
- compilare ed inviare il codice al dispositivo LEGO

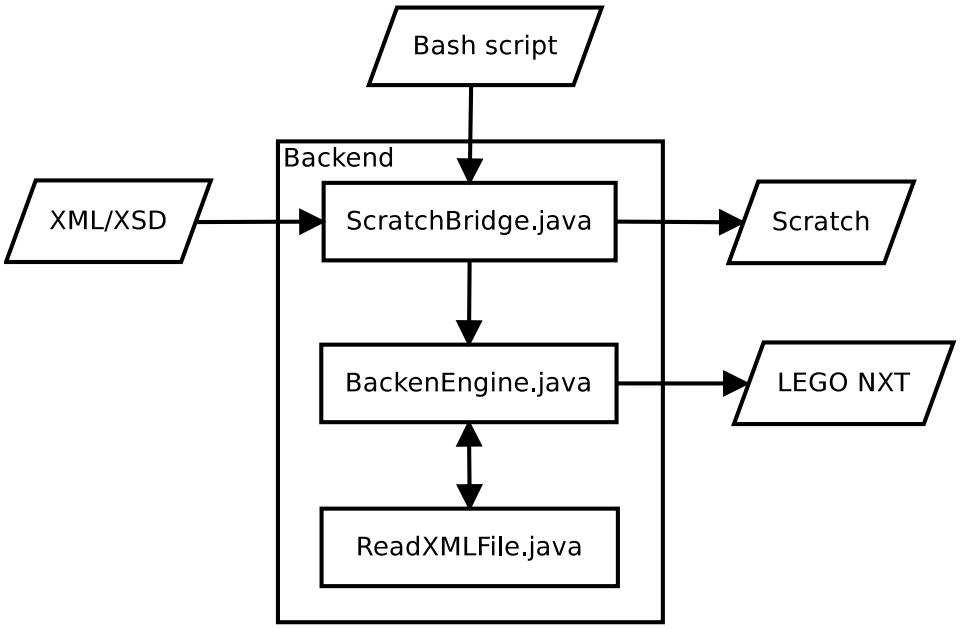


Figura 6.8: Backend: si possono notare le tre classi fondamentali che ne costituiscono l'ossatura.

Nella figura è possibile individuare le classi che si occupano delle operazioni appena descritte:

- ScratchBridge: classe che si occupa di ricevere i parametri di avvio dallo script bash, di leggere i file XML e XSD per la configurazione del robot, di avviare il thread di backend e di avviare Scratch.

- BackendEngine: classe che si occupa di gestire la connessione e la ricezione dei messaggi con Scratch, di avviare i processi di traduzione e compilazione e di comunicare con il dispositivo LEGO NXT

- ReadXMLFile: classe che si occupa della traduzione del codice XML in codice NXC

Ciascuno di questi componenti utilizza classi aggiuntive di supporto che verranno descritte nel dettaglio nel manuale tecnico.

6.4 Traduzione del codice

Come già accennato nel capitolo precedente, la traduzione del codice è una parte fondamentale di SILENT, in quanto permette di integrare due rappresentazioni diverse del flusso di un programma.

Figura 6.9: Esempio di programma nell'interfaccia Scratch.

Algoritmo 6.1 Rappresentazione in formato XML del programma.

```
1  <scratchWorkplane>
2      <scratchSprite type="a ScratchStageMorph()" name="Stage"
           />
3      <scratchSprite type="a ScratchSpriteMorph()" name="
           Sprite1">
4          <scratchScript>
5              <block event="Scratch-StartClicked"/>
6              <block selector="say:at:">
7                  <argument>Hello World!</argument>
8                  <argument>0</argument>
9              </block>
10         </scratchScript>
11     </scratchSprite>
12 </scratchWorkplane>
```

Algoritmo 6.2 Traduzione in NXC del programma.

```
1  task main() {
2      //start variable declaration
3      //end variable declaration
4      TextOut (0,0,"Hello World!");
5  }
```

Innanzi tutto avviene l'esportazione del programma "scritto" in Scratch (Fig 6.9) nel formato XML che lo rappresenta (Algoritmo 6.1).

Una volta recuperato il contenuto del file xml, viene inizializzato il contenuto DOM che descrive l'albero del flusso del programma realizzato in Scratch. Eseguendo una visita in depth-first dell'albero XML si può ricostruirne la struttura originale.

La visita dell'albero XML avviene con l'analisi dei nodi incontrati, catturando i nomi dei nodi viene effettuata una decisione sulla traduzione da effettuare, caso per caso. Sfruttando poi gli attributi e il contenuto dei tag xml viene caratterizzato ogni singolo costrutto del flusso. A seconda del costrutto da tradurre, possono essere avviate procedure di traduzione annidate in modo da risolvere casi quali chiamate a funzione o espressioni complesse.

Prendendo per esempio l'algoritmo 6.2, quando SILENT incontra <block selector="say:at:">, vengono effettuati una serie di controlli per recuperare correttamente il contenuto da mostrare a schermo.

Algoritmo 6.3 Frammento di codice di ReadXMLFile.java che si occupa della traduzione del blocco say:at:.

```
1  if (attribs.item(0).getNodeValue().equalsIgnoreCase("say:at:
       ")) {
2      NodeList childList = node.getChildNodes();
3      String str1=parseNode(childList.item(0));
4      String type_out=getExpressionTypification(childList.item
           (0),0);
5      int str2=Integer.parseInt(childList.item(1).
           getTextContent());
6      int line=str2*8; //The line value must be a multiple of
           8
7      if(type_out.equalsIgnoreCase("string"))
8          parsedNode += alignment + "TextOut (0," + line +","
               + str1  + ");";
9      else
10         parsedNode += alignment + "NumOut (0," + line +"," +
               str1  + ");";
11 }
```

Il primo "argument" (linea 3) non viene semplicemente recuperato come valore testuale in quanto in esso è possibile trovare variabili o espressioni più complesse che quindi vanno elaborate correttamente: in questo caso infatti viene richiamato il metodo parseNode() che si occupa proprio di recuperare il "valore" corretto da inserire come argomento nella funzione di *Out NXC.

Il secondo "argument" (linea 5), invece, essendo un valore scelto da un menù a tendina e quindi prefissato, viene semplicemente recuperato come valore testuale e poi convertito in intero, questo perché la particolare funzione di stampa su schermo di NXC richiede il passaggio del valore intero del numero di riga sulla quale scrivere l'output.

Con "type_out" viene semplicemente individuato il tipo di input passato al blocco say:out: in modo da poter tradurre nella funzione corretta NXC.

6.5 Riepilogo funzionamento generale

E' possibile riassumere il funzionamento generale di SILENT con il seguente schema, esso racchiude le sue componenti principali e le connessioni che esistono tra loro, i file presi come input (MyRobot.xml, MyRobot.xsd) e quelli prodotti in output (<my_project>.sb, MyRobotSS.xml, MrRobot_<spriteName>.NXC).

Per una descrizione dettagliata, di carattere implementativo, si rimanda al Manuale Tecnico.

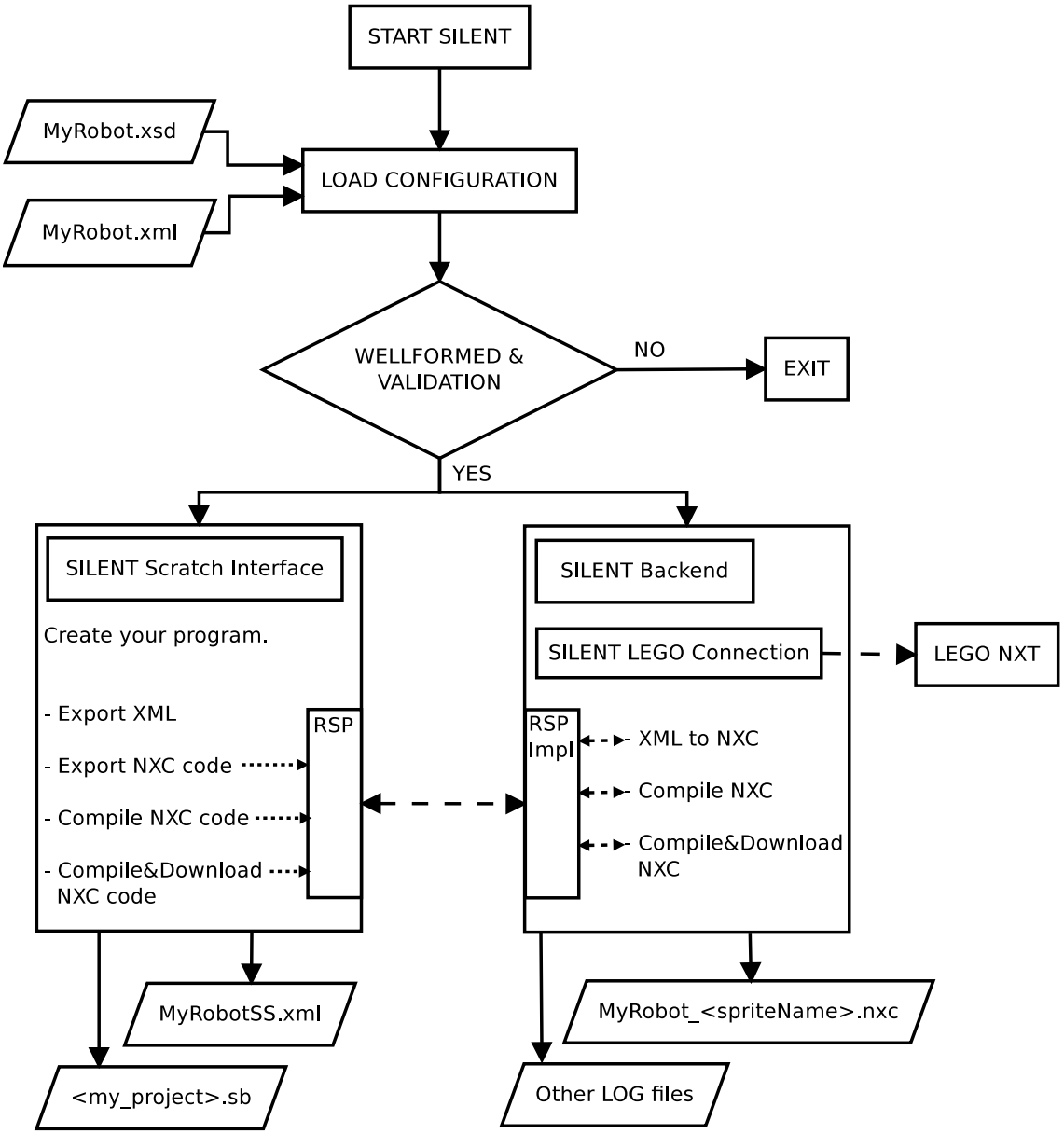


Figura 6.10: Schema del funzionamento generale di SILENT

7 Manuale utente

In questo capitolo verranno illustrate le funzionalità di SILENT, i suoi requisiti e come realizzare un primo semplice programma.

7.1 Caratteristiche

Funzionalità di SILENT :

- permette di realizzare in maniera semplice ed intuitiva programmi per controllare i robot LEGO MINDSTORM NXT;
- mette a disposizione blocchi di controllo di flusso, variabili, array, operatori, controllo di motori, utilizzo dei sensori, stampa su display e produzione di suoni;
- gestisce autonomamente la compilazione ed il download del programma sul brick LEGO;
- supporta la connessione USB con il brick;
- grazie a Java e Squeak è cross OS: supporta sistemi operativi Linux e Windows. E' stato testato con successo su Ubuntu, Windows XP e Windows 7,
- permette di lavorare senza aver collegato un dispositivo NXT, ne viene richiesta infatti la presenza solo se si sceglie di effettuare il download del codice
- permette di organizzare il programma in task;
- permette di realizzare più programmi contemporaneamente, utilizzando sprite diversi infatti verranno prodotti file sorgenti NXC differenti, ciascuno indipendente dall'altro. Sarà poi compito dell'utente scegliere quali avviare sul robot.

Cosa non fa SILENT:

- non comunica in modo bidirezionale con il brick LEGO: permette infatti soltanto di inviare il programma realizzato, ma è non ancora previsto che possa ricevere input dal brick quali ad esempio i valori dei sensori;
- non supporta la ricorsione: si basa infatti su Scratch 1.4, non implementa le funzionalità di BYOB, anche perchè la ricorsione non è nativamente supportata dal firmware della LEGO.
- non traduce il programma realizzato negli sprite in animazione nella GUI, come invece avviene in Scratch;
- non supporta la connesione BLUETOOTH con il brick LEGO.

7.2 Requisiti

Requisiti generali:

- computer con uno dei sistemi operativi testati: Linux Ubuntu 10.04+, Windows 7, Windows XP. Dovrebbe funzionare correttamente anche su altre versioni di sistemi Linux o Windows;
- computer dotati di porta USB 2.0 o maggiore e un kit LEGO MINDSTORM NXT 2.0;
- Java 6 o superiore installato.

Requisiti per Scratch:

- Schermo: 800 x 480, 16-bit o maggiore.
- Sistema Operativo: Windows 2000 o più, Mac OS X 10.4 o più, Ubuntu Linux 9.04-10.04.
- Memoria Fissa: almeno 120 MB per installare Scratch.
- CPU e memoria: la maggior parte dei computer hanno memoria sufficiente, ma su alcuni la velocità può essere ridotta.
- Suono / Video: Avere un microfono e degli altoparlanti non è obbligatorio, ma sono indispensabili se si vogliono ascoltare e/o registrare i suoni.

Requisiti per collegare LEGO MINDSTORM NXT:

- USB 2.0 o superiore

- Driver windows[http://mindstorms.lego.com/en-us/support/files/default.aspx#Driver] o per Linux controllare che sia correttamente riconosciuto utilizzando:

```
luca@luca-nb: lsusb | grep Lego Bus
002 Device 002: ID 0694:0002 Lego Group Mindstorms NXT
```

7.3 Avvio

Dalla directory root del programma basterà effettuare un doppio click su:

- Linux: "run" e scegliere di eseguire lo script
- Windos: "run.bat"

Una volta fatto apparirà la finestra principale di SILENT:_

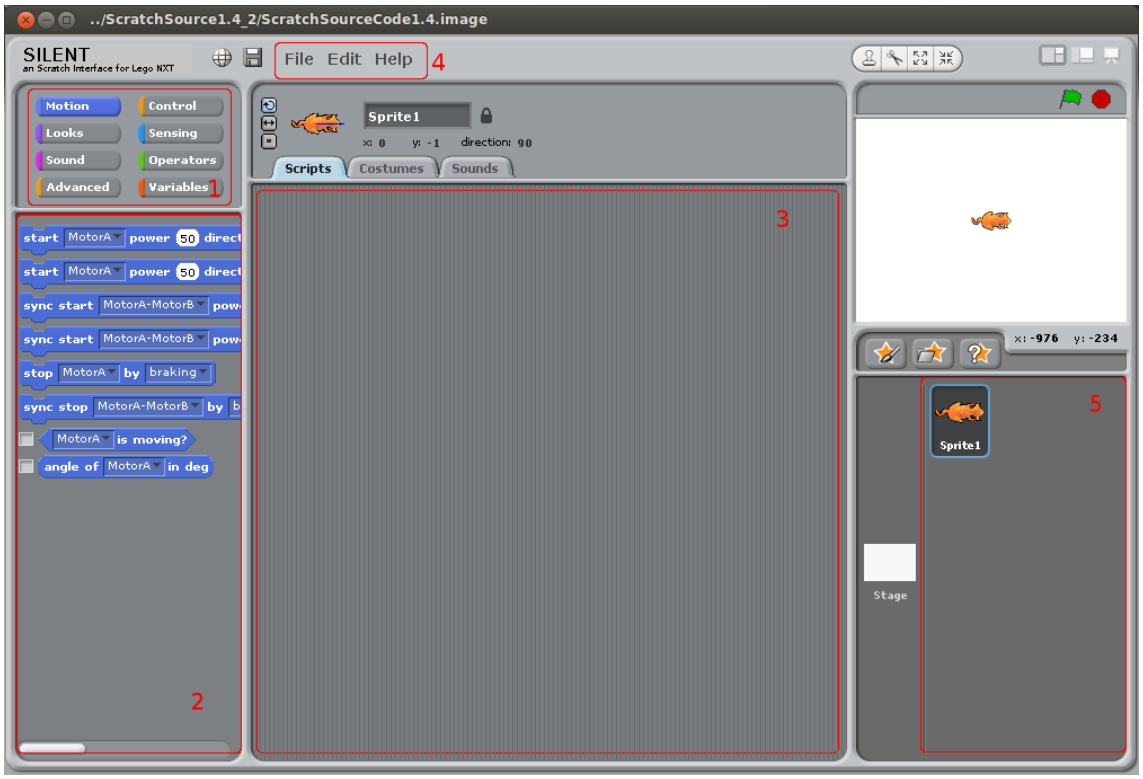


Figura 7.1: Schermata principale di SILENT.

In essa è possibile distinguere i componenti principali:

1. selettore palette: qui è possibile scegliere tra le categorie di blocchi a disposizione, cliccando su ciascuna di esse apparirà nella sezione 2 la lista dei blocchi a disposizione

2. selettore blocchi palette: qui viene mostrato l'elenco dei blocchi a disposizione per la particolare palette. Da qui i blocchetti possono essere trascinati sul workspace per essere assemblati. Dal workspace di lavoro possono essere trascinati su questa zona per essere eliminati dal progetto in lavoro.

3. workspace: qui è dove si sviluppa il programma sul quale si lavora, dove i blocchetti sono trascinati e organizzati in un flusso. In ogni workspace (che corrisponde ad uno Sprite) è possibile rilasciare un solo blocchetto di tipo "When green flag clicked", in quanto rappresenta il main del programma, mentre si possono realizzare quanti task si desiderano.

4. barra dei menù: cliccando sulle varie voci si apriranno i rispettivi menù che danno accesso alle funzioni più disparate. Il più interessante è il menu "File" che contiene le funzioni di collegamento al mondo reale, ovvero con il dispositivo LEGO NXT.

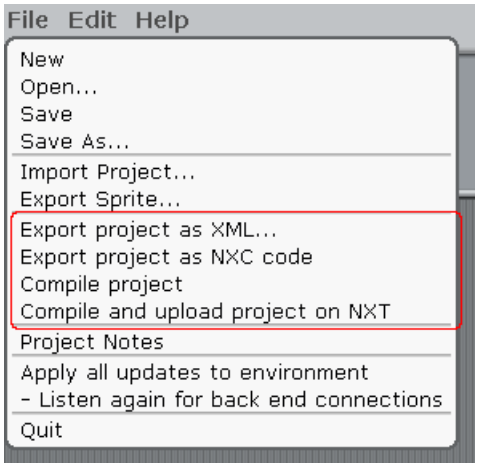


Figura 7.2: Nel riquadro rosso la sezione più interessante, quella che permette di accedere alle funzionalità di esportazione del codice.

5. elenco sprite: in questo frame è possibile creare un nuovo sprite, eliminare e vedere la lista di quelli in uso. A ciascuno sprite corrisponderà poi un file NXC indipendente.

7.4 Collegamento dell'NXT

Il collegamento del brick NXT al programma è molto semplice e può essere effettuato in qualsiasi momento dato che è richiesta la presenza di un dispositivo NXT solo nel momento in cui viene selezionato dal menù "File" la voce "Compile & upload project on NXT".

Per collegare il brick basta prendere il cavo USB in dotazione e collegarlo da una parte al brick e dall'altra ad una qualsiasi porta USB del computer sul quale sta eseguendo SILENT. Occorre assicurarsi che il brick sia acceso, altrimenti è come se non fosse collegato e SILENT restituirà un messaggio d'errore nel caso si tenti di esportare il proprio progetto verso il robot.

Naturalmente devono essere stati precedentemente installati i driver necessari per gestire il dispositivo NXT:

- Windows\Mac: http://mindstorms.lego.com/en-us/support/files/default.aspx#Driver
- Linux: non ha bisogno di driver aggiuntivi, almeno per Ubuntu\Debian.s

7.5 Blocchi e funzionalità di SILENT

In questo paragrafo affronteremo i blocchi e le relative funzionalità introdotti in SILENT, mentre per quanto riguarda tutto ciò che è stato ereditato da Scratch si può far riferimento al wiki del progetto originale [28].

7.5.1 Motors

Figura 7.3

Questo blocco permette soltanto di accendere il motore etichettato "MotorA" con una potenza del 50% in direzione "forward".

La scala delle potenze è inclusa in [-100,100], mentre la direzione può essere "forward" o "backward".

Figura 7.4

Questo blocco fornisce un controllo più accurato permettendo di scegliere per quanto tenere acceso il motore, specificando la quantità e il tipo di conteggio da effettuare, in questo caso 360 gradi (i valori possibili sono "Degree", "Rotation", "Seconds"). Inoltre permette di specificare in quale maniera il motore verrà fermato: "Braking" oppure "Coasting". La differenza consiste nel fatto che il primo ferma la rotazione bruscamente, mentre la seconda rallenta la rotazione più gradualmente fino al raggiungimento della fermata.

Figura 7.5

Questo blocco permette di azionare due motori diversi in maniera sincronizzata (si noti infatti l'etichettatura "MotorA-MotorB"). Nella casella dei motori verranno mostrate tutte le coppie possibili con la configurazione di motori modellata.

L'effetto è simile al primo visto, la differenza consiste nella presenza del comando di "steering" che permette di specificare un'eventuale movimento a curva della traiettoria seguita da un possibile veicolo spinto dalla coppia di motori utilizzata.

Figura 7.6

Questo comando ricalca il secondo visto in Fig. 7.4, con l'aggiunta dello steering.

(a)

(b)

Figura 7.7

I comandi sopra permettono di fermare la rotazione di un motore (7.7a) o di una coppia di motori (7.7b), utilizzando i due metodi già visti: “Braking” o “Coasting”.

(a) **(b)**

Figura 7.8

I due blocchi della figura 7.8 permettono invece di:

(a) controllare se un motore è in movimento o meno, blocchi di questo tipo sono dei reporter booleani.

(b) ottenere la misura dell’angolo attuale del motore in gradi.

7.5.2 Looks

Figura 7.9: Say <Hello!> at line N

Questo blocco permette di scrivere sul display del brick un contenuto passato nel primo campo. Col menù a tendina del secondo è possibile scegliere la riga sulla quale scrivere.

Figura 7.10: Clear line N

Questo blocco permette di pulire la linea indicata dal valore selezionato nel menù a tendina.

7.5.3 Sound

Figura 7.11

7.5.4 Advanced

Non ci sono blocchi implementati.

7.5.5 Control

Figura 7.12

Questo blocco definisce il task "main" del programma che si sta realizzando. E' il task principale, quello che viene eseguito dall'interprete e che dà inizio all'esecuzione del flusso di istruzioni.

Deve quindi essere sempre incluso un blocco di questo tipo, ma non è possibile inserirne più d'uno.

Senza la presenza di questo blocchetto, il programma non compila correttamente.

Figura 7.13

Il blocco "task" permette di definire task da eseguire sul brick: premendo sul selettore a tendina verranno visualizzati la lista dei task già realizzati e sarà possibile realizzarne uno nuovo.

Visto che NXC supporta il multi-threading, un task in NXC corrisponde direttamente ad un task in NXT.

Figura 7.14

Il blocco “call <task>” permette di chiamare un task precedentemente predefinito nel nostro codice.

Figura 7.15

Il blocco “stop script <taskname>” permette di effettuare lo stop di un task tra quelli definiti nel nostro codice.

Nota bene che un task si ferma quando raggiunge naturalmente la fine del corpo del proprio codice.

Figura 7.16

Il blocco “stop all” permette di stoppare tutti i task in esecuzione nel momento in cui viene richiamata la funzione.

7.5.6 Sensing

I blocchi seguenti sono quelli dedicati alla gestione dei sensori collegabili al brick NXT.

Non sono sempre visibili, a seconda della configurazione del modello di robot potrebbero non essere mostrati tutti e potrebbero essere differenti anche le etichette date ai sensori.

Per quanto riguarda i blocchi “set” non fanno altro che configurare il sensore etichettato come da blocchetto alla porta specificata nel file di configurazione del robot.

Figura 7.17

Il blocco “set <light sensors> <type> <type of value>” permette di impostare il sensore di luminosità. I valori dei tre campi possono essere:

1. etichetta del sensore
2. ambient o reflected: luce ambientale o riflessa
3. raw o percent: valore raw o in percentuale.

Figura 7.18

Il blocco “set <sound sensors> <type> <type of value>” permette di impostare il sensore di suono. I valori dei tre campi possono essere:

1. etichetta sensore
2. dB o dBA (dB normalizzato)
3. raw o percent

Figura 7.19

Il blocco “set <touch sensors> <type> <type of value>” permette di impostare il sensore di contatto. I valori dei tre campi possono essere:

1. etichetta sensore
2. touch

3. bool, edge, pulse, raw: bool indica se il sensore è premuto o meno, edge indica quante volte è stato premuto, pulse quante volte è stato rilasciato e raw dà un valore "grezzo" del sensore.

Figura 7.20

Effettua un set del sensore ad ultrasuoni, unica possibilità è scegliere l'etichetta del sensore che si vuole impostare.

Figura 7.21

Questo blocco permette di effettuare un reset del sensore selezionabile dal menù a tendina: in esso infatti verranno visualizzati tutti i sensori configurati.

Figura 7.22

Questo blocco permette di ottenere la lettura di un sensore, nel menù a tendina verranno mostrate le etichette dei sensori che prevedono questa possibilità.

Il valore ritornato sarà del tipo specificato con i blocchi di "set", che naturalmente vanno utilizzati prima di questo.

Figura 7.23

Un reporter booleano che indica se il pulsante "left" o "right" è stato premuto sul brick.

7.5.7 Variable

Figura 7.24

Permette di impostare il valore di una variabile creata precedentemente con “make variable”.

Accetta valori numerici con virgola e stringhe.

Figura 7.25

Permette di incrementare il valore di una variabile del valore indicato.

In caso di variabili di tipo stringa ne fa la concatenazione.

Figura 7.26

Permette di inizializzare un array, operazione preliminare e necessaria prima di poter farne uso nel programma che si vuole realizzare.

I sei campi disponibili rappresentano:

- etichetta rappresentante il nome dell’array
- il tipo di array che si vuole inizializzare, possibili valori sono string e float
- numero di celle per ogni dimensione, sono concessi array fino a 4 dimensioni. Le dimensioni vanno utilizzate in maniera crescente, da sinistra verso destra, quelle non utilizzate vanno lasciate col -1. Per esempio se vogliamo inizializzare un array di stringhe ad una dimensione con 10 celle:

Figura 7.27

Figura 7.28

Questo blocco ci permette di ottenere il valore memorizzato in una determinata cella di un array.

Le celle non inizializzate/non necessarie vanno lasciate col -1.

Figura 7.29

Questo blocco permette di impostare il valore di una cella di un array, specificando il campo “value”.

Le celle non necessarie/non inizializzate vanno lasciate al valore di default -1.

Figura 7.30

Questo blocco permette di ottenere la lunghezza di un array, o di parte di esso nel caso di array multidimensionali.

7.6 Il tuo primo programma: "Hello world!"

In questo paragrafo verrà mostrato come realizzare un semplicissimo programma che stampa una scritta sul video del brick NXT.

Una volta aperto SILENT bisogna assicurarsi di aver collegato il brick LEGO NXT e che sia acceso, si eviterà cosi di incontrare errori nella fase finale dell’esercizio.

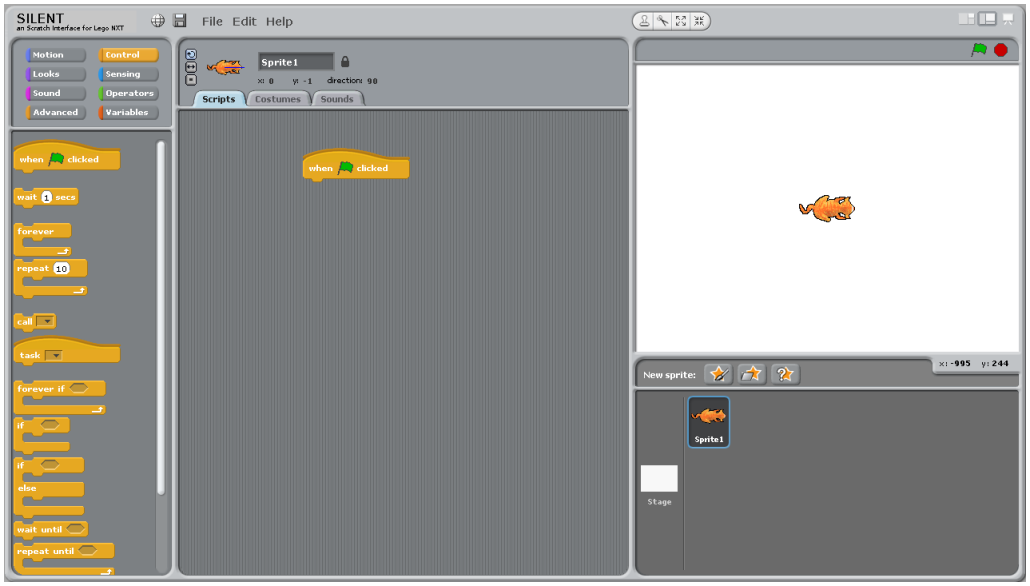


Figura 7.31

Si vada nella sezione Control e facendo click col sinistro sul blocco main lo si trascini nello spazio di lavoro.

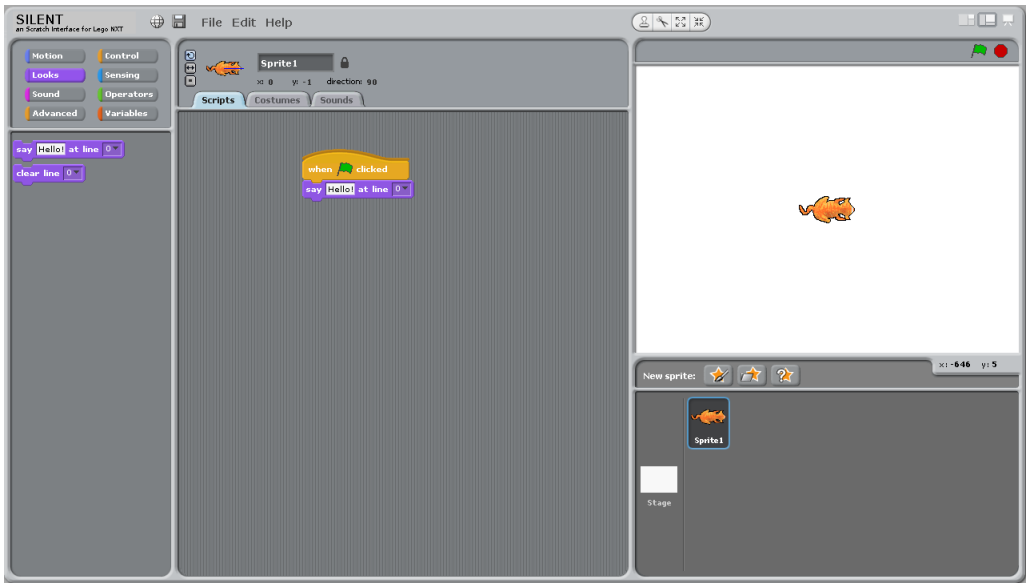


Figura 7.32

Spostandosi ora nella sezione Looks e si trascini il blocco say sotto il blocco main finché non si agganceranno, quindi è possibile rilasciare il tasto sinistro del mouse.

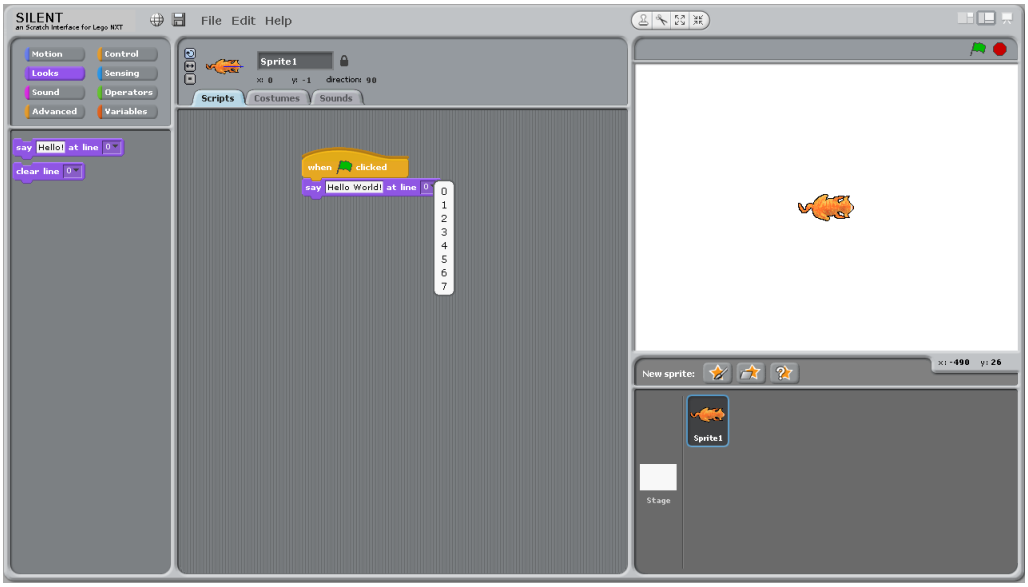


Figura 7.33

Si sposti il mouse sopra la casella di testo del blocco say appena inserito, selezionalo con un click col sinistro ed è possibile iniziare a scrivere il testo che si preferisce. Dopodiché ci si sposti sul menù a tendina e si selezioni su quale riga si vuole visualizzare la scritta.

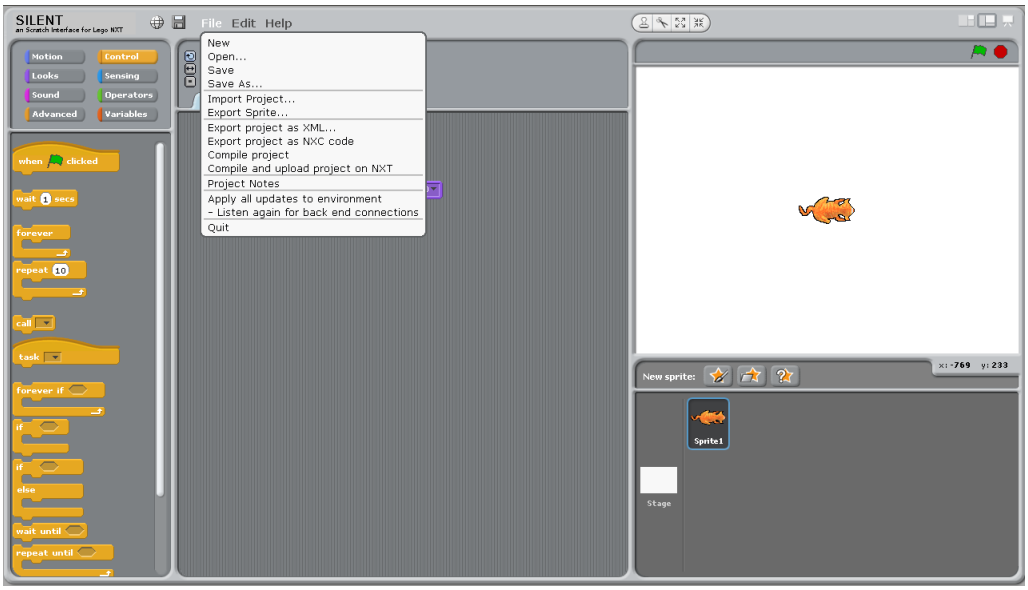


Figura 7.34

Infine dal menù file si scorra fino ad Compile&Upload per esportare, compilare e inviare il programma al brick NXT.

Utenti Linux: in questa fase può essere richiesto di inserire la tua password utente, perché il download del codice sul dispositivo LEGO richiede i permessi di amministratore che è possibile ottenere solo attraverso un sudo.

Ricevuto il messaggio di operazione completata con successo, si dovrebbe sentire anche un bip sul brick, si può quindi avviare il tuo programma sul robot andandolo a selezionare tra quelli caricati direttamente attraverso il menù sul brick.

7.7 Messaggi di errore

Gli errori più comuni in cui ci si può imbattere sono quelli relativi al mancato collegamento del robot con il computer.

Infatti in fase di upload se SILENT non rileva la presenza del robot solleva un'errore segnalandolo sull'interfaccia con un scritta rossa e terminando il processo di compilazione ed upload.

In questo caso basta accendere il brick o assicurarsi che sia ben collegato e riconosciuto dal tuo computer, e ripetere l'operazione.

Eventuali errori di esportazione e/o compilazione sono sempre segnalati sull'interfaccia e possono essere esaminati in dettaglio attraverso dei file di log che SILENT produce nella cartella home dell'utente che lo sta eseguendo. Nel caso di SO Linux si tratta della cartella /home/<username>, nel caso di SO Windows nella cartella C:\Documents and Settings\nome_utente in sistemi Windows 2000, XP e Server 2003 C:\Users\nome_utente in sistemi Windows Vista, Server 2008 e 7.

8 Manuale tecnico

In questo capitolo verranno illustrate nel dettaglio le implementazioni, le scelte effettuate e come apportare le modifiche per eventuali miglioramenti o ampliamenti del programma.

8.1 Configurazione modello robot

Per configurare SILENT è necessario scrivere il proprio file di modello del robot. Questa operazione è da fare necessariamente prima di avviare SILENT, poichè esso carica la configurazione del modello solo all'avvio.

Per configurare il robot è necessario editare il file "robotmodel.xml" localizzabile nella cartella "ScratchSource1.4_2" nella cartella principale di SILENT. Per modificare tale file è sufficiente un qualsiasi editor di testo.

Viene qui riportato un esempio esaustivo della configurazione di un robot con 3 motori e 4 sensori tutti diversi, ovvero con tutte le porte del brick collegate. Questo esempio può essere preso come template per compilare la configurazione desiderata.

Per configurare i motori è necessario editare la parte compresa tra i tag <motors/>, ogni motore è incluso nel tag <motor/>. In tale campo è necessario definire:

- <label/> permette di definire un'etichetta da assegnare al motore, che lo identificherà nei blocchi e nella traduzione, ed è quindi necessario che sia univoca.
- <port/> permette di definire a quale porta è collegato il motore, le porte dedicate ai motori nel brick NXT sono le A,B,C.
- <type/> lascia la possibilità in futuro di poter configurare l'utilizzo di motori diversi da quelli standard NXT. Per ora SILENT accetta solo il valore "TA" per questo campo, che quindi è sostanzialmente fisso.

Algoritmo 8.1 Esempio di "robotmodel.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
<robotmodel>
  <motors>
    <motor>
      <label>MotorA</label>
      <port>A</port>
      <type>TA</type>
    </motor>
        <motor>
      <label>MotorB</label>
     <port>B</port>
      <type>TA</type>
    </motor>
        <motor>
      <label>MotorC</label>
     <port>C</port>
      <type>TA</type>
    </motor>
  </motors>
   <sensors>
        <sensor>
                <label>Touch</label>
                <port>1</port>
                <type>touch</type>
        </sensor>
        <sensor>
                <label>Sound</label>
                <port>2</port>
                <type>sound</type>
        </sensor>
        <sensor>
                <label>Light</label>
                <port>3</port>
                <type>light</type>
        </sensor>
        <sensor>
                <label>Ultrasonic</label>
                <port>4</port>
                <type>ultrasonic</type>
        </sensor>
  </sensors>
</robotmodel>
```

Per configurare i sensori è necessario editare la parte compresa tra i tag <sensors/>, ogni sensore è incluso nel tag <sensor/>. In tale campo è necessario definire:

- <label/> permette di definire un'etichetta da assegnare al sensore che lo identificherà nei blocchi e nella traduzione, è quindi necessario che essa sia univoca.
- <port/> permette di definire a quale porta è collegato il sensore, le porte dedicate ai sensori nel brick NXT sono le 1,2,3,4.
- <type/> permette di definire il tipo di sensore che si sta configurando, che può essere:
  - touch
  - sound
  - light
  - ultrasonic

Una volta scritto il proprio file di configurazione occorre salvarlo, avendo cura che il suo nome sia "robotmodel.xml" altrimenti non verrà letto correttamente al momento del caricamento di SILENT.

Si può avere una conferma di aver scritto correttamente il file di configurazione attraverso la validazione con "robotmodel.xsd", essa viene fatta anche da SILENT nella procedura di avvio, ma è sempre possibile utilizzare un editor che permetta la validazione per avere un controllo preventivo sul xml appena scritto.Importazione della configurazione

Il processo di importazione inizia con una validazione preventiva del file di configurazione letto.

Nella classe "ScratchBridge.java" viene inizializzato un oggetto della classe "DocBookXSDChek.java" che si occupa della validazione del file xml con la grammatica xsd entrambi passati come argomento al metodo "XSDCheck(String xml, String xsd)".

Superata la fase di validazione viene avviata l'interfaccia di Scratch (linea 18 di Algorithm 8.2). Come già accennato nei capitoli precedenti Scratch è stato modificato per poter leggere il file di configurazione del robot e così preparare a sua volta la configurazione dei blocchetti che rappresentano le componenti del dispositivo robotico da programmare.

Algoritmo 8.2 Frammento di "ScratchBridge.java" nel quale avviene la validazione del file di configurazione "robotmodel.xml", in caso di successo viene avviato il Backend e l'interfaccia di Scratch.

```
1  [...]
2          DocbookXSDCheck XSDCheck=new DocbookXSDCheck();
3          boolean validator;
4          String scratch_command;
5          if(portability.getOs()=="linux")
6          {    validator=XSDCheck.XSDCheck("../ScratchSource1.4
               _2/robotmodel.xml", "../ScratchSource1.4_2/
               robotmodel.xsd");
7               scratch_command=UNIX_SCRATCH_COMMAND;
8          }
9          else
10         {    validator=XSDCheck.XSDCheck("..\\ScratchSource1
               .4_2\\robotmodel.xml", "..\\ScratchSource1.4_2\\
               robotmodel.xsd");
11              scratch_command=WIN_SCRATCH_COMMAND;
12         }
13         if(validator==true)
14         {
15              BackendEngine BE=new BackendEngine();
16              BE.start();
17
18              SaferExec se = new SaferExec(SaferExec.
                   NO_TIMEOUT);
19              System.out.println("Executing: " +
                   scratch_command + args[0]);
20              se.exec(scratch_command + args[0]);
21
22              System.out.println("Scratch Front End has quit, 
                   Backend is exiting.");
23         }
24         else
25         {    System.out.println("\nERROR: XML configuration 
               is not valid!!\nCheck your XML robot 
               configuration in Scratch folder.\n\nType any key 
               for terminate.");
26              System.in.read();
27         }
28     }
29 [...]
```

Algoritmo 8.3 Frammento di codice della classe "DocbookXSDCheck" nel quale avviene la validazione della configurazione.

```
1  [...]
2  public class DocbookXSDCheck {
3
4      public DocbookXSDCheck(){
5      }
6      public static boolean XSDCheck(String xml, String xsd)
           throws SAXException, IOException{
7          SchemaFactory factory = SchemaFactory.newInstance("
               http://www.w3.org/2001/XMLSchema");
8
9          File schemaLocation = new File(xsd);
10         Schema schema = factory.newSchema(schemaLocation);
11
12         Validator validator = schema.newValidator();
13
14         Source source = new StreamSource(xml);
15
16         try {
17             validator.validate(source);
18             System.out.println(xml + " is valid.");
19             return true;
20         }
21         catch (SAXException ex) {
22             System.out.println(xml + " is not valid because
                   ");
23             System.out.println(ex.getMessage());
24             return false;
25         }
26     }
27 }
28 [...]
```

Algoritmo 8.4 Robotmodel inizializzazione

```
1  initialize
2  | t1 t3 t2 |
3          super initialize.
4
5          t1 := 'robotmodel.xml'.
6          "leggo file configurazione robot"
7          t3 := FileStream oldFileNamed: t1.
8      t3 ifNil: [self error: 'could not open ' , t1].
9      t2 := XMLObject new importFromFileStream: t3.
10         robotmodel := t2 children.
11     t3 close.
```

Viene inizializzata la classe RobotModel, che si occupa di leggere il file robotmodel.xml, caricarne l'xml in un XMLObject che sucessivamente verrà analizzato per costruire i blocchi necessari.

Nel listato 8.4 viene mostrato il metodo di inizializzazione squeak della classe "RobotModel", che non fa altro che leggere il file XML dal filesystem, costruire lo "XMLObject" effettuando un parsing dello stream passatogli e salvare l'oggetto DOM risultante in "robotmodel". A tale oggetto accederanno successivamente i metodi di costruzione dei blocchetti per recuperare le informazioni necessarie all'operazione.

Nella classe RobotModel esistono una serie di metodi che permettono di recuperare informazioni sui motori e i sensori configurati dall'oggetto xml inizializzato, vengono utilizzati per popolare gli array necessari all'inizializzazione corretta delle palette relative ai motori e sensori. Tali metodi sono raggruppati nella categoria "getter" della classe RobotModel (Figura 8.1).

Un esempio di utilizzo di questi metodi è proprio la produzione del codice per i blocchetti specifici per la configurazione inserita visibile nell'Algortimo 8.5

Alla riga 7 viene chiesto al modello tutte le periferiche di tipo "sensors", questo comporta che i metodi sopra analizzino l'XMLObject e recuperino la lista degli elementi presenti tra i tag <sensors/>. In questo caso specifico serve solo per verificare che ci siano sensori configurati, cosi da mostrare i blocchi utili per ottenre la lettura del valore del sensore.

In maniera del tutto simile nell' ifTrue linea 14 viene controllata la presenza di ciascun tipo di sensore, in modo da restituire una lista dei blocchi utili per effettuare

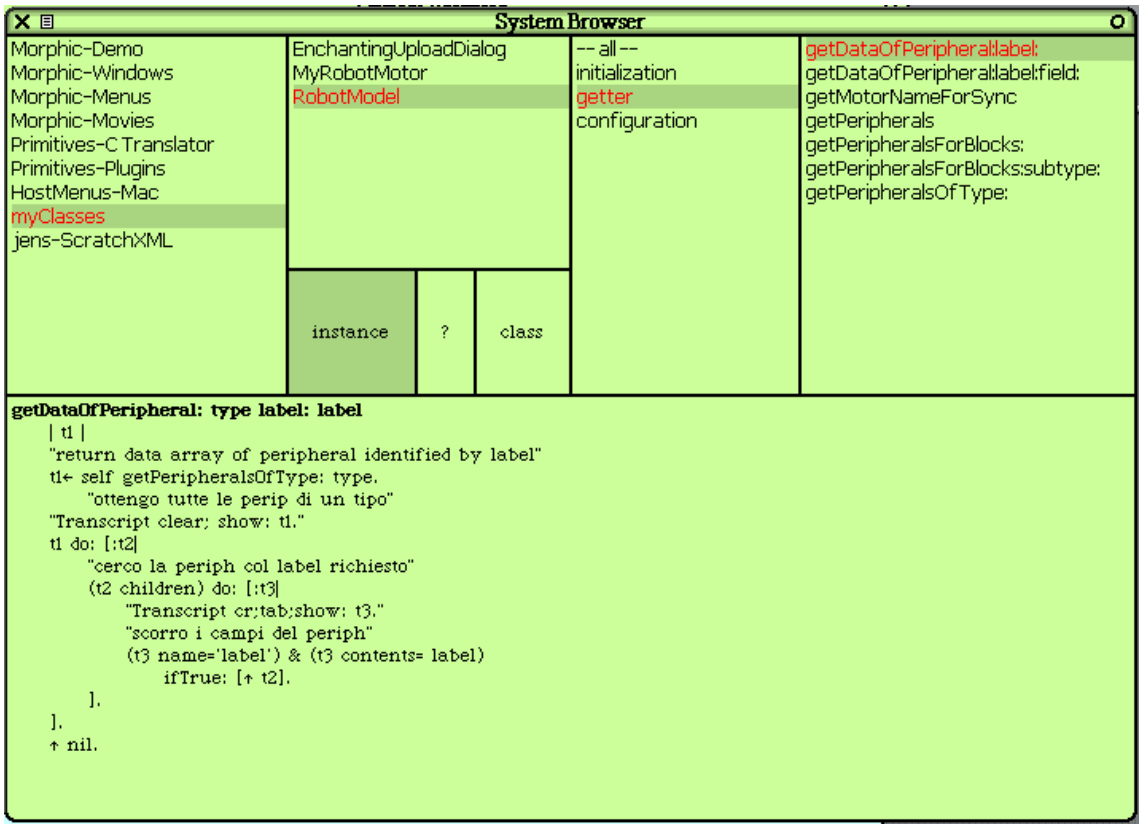


Figura 8.1: Metodi getter della classe RobotModel utilizzati per recuperare le informazioni necessarie per costruire i blocchetti motori e sensori.

Algoritmo 8.5 metodo "blocksFor:" di RobotModel che restituisce la configurazione dei blocchetti per il tipo di device passato come argomento.

```
[...]
    "now simply gets value from sensor, sets before"
     (aType='normvalue')
        ifTrue: [
            sensingrow:=OrderedCollection new.
            sensingrow add: 'sensing'.
            t4 := self getPeripheralsOfType: 'sensors'.
            (t4 isEmpty)
                ifFalse: [ sensingrow add: #('value of %T
                   sensor'      r   myRobotSensorValue:)
                   ].
            sensingrow add: #-.
            ^sensingrow.
        ].
    (aType='setsensor')
        ifTrue: [
            sensingrow:=OrderedCollection new.
            sensingrow add: 'sensing'.
            t4:=self getPeripheralsForBlocks: 'sensors'
               subtype: 'light'.
            (t4 isEmpty)
                ifFalse: [ sensingrow add: #('set %p %a01 %
                   a02'     -   myRobotSetSensor:type:mode:)
                   ].
            t4:=self getPeripheralsForBlocks: 'sensors'
               subtype: 'sound'.
            (t4 isEmpty)
                ifFalse: [ sensingrow add: #('set %K %a03 %
                   a04'     -   myRobotSetSensor:type:mode:)
                   ].
            t4:=self getPeripheralsForBlocks: 'sensors'
               subtype: 'touch'.
            (t4 isEmpty)
                ifFalse: [ sensingrow add: #('set %F %a05 %
                   a06'     -   myRobotSetSensor:type:mode:)
                   ].
            t4:=self getPeripheralsForBlocks: 'sensors'
               subtype: 'ultrasonic'.
            (t4 isEmpty)
                ifFalse: [ sensingrow add: #('set %q'
                   -   myRobotSetSensor:)].
            sensingrow add: #-.
            sensingrow add: #('reset %T'       -
               myRobotSensorReset:).
86          sensingrow add: #-.
            ^sensingrow.
        ].
[...]
```

Algoritmo 8.6 Frammento di ScriptableScratchMorph-blockSpecs, nella parte interessata dalla modifica per SILENT.

```
robotmodel := RobotModel new.
        t3:=robotmodel blocksFor: 'motors'.
        t4:=robotmodel blocksFor: 'setsensor'.
        (t4 isNil)
                ifFalse:        [t3:=t3,t4.].
        t4:=robotmodel blocksFor: 'normvalue'.
        (t4 isNil)
                ifFalse:        [t3:=t3,t4.].
        t4:=robotmodel blocksFor: 'playsound'.
        (t4 isNil)
                ifFalse:        [t3:=t3,t4.].
        t4:=robotmodel blocksFor: 'buttons'.
        (t4 isNil)
                ifFalse:        [t3:=t3,t4.].
```

il setting del sensore.

8.2 Come modificare i blocchi

Per modificare i blocchi mostrati in SILENT è necessario sapere quale sia la categoria di appartenenza del blocco. Le categorie seguono le palette mostrate nell'interfaccia, quindi sono Motoion, Looks, Advanced, Control, Sensing, Operator, Variable.

La classe "ScriptableScratchMorph" contiene il metodo di classe "blockSpecs" nel quale è possibile aggiungere\modificare i blocchetti da mostrare nell'interfaccia utente e quindi da rendere disponibili all'utente (Algoritmo 8.6)

Nello specifico di SILENT sono state commentate le categorie, o parte di esse, interessate dalle modifiche, aggiungendo la chiamata a metodi di RobotModel per poter recuperare i nuovi blocchetti.

Per ciascuno degli argomenti passati a "robotmodel blocksFor:" viene restituito l'array che realizza i blocchi desiderati:

- motors: restituisce l'array costitutivo per i blocchi motore
- setsensors: restituisce l'array costitutivo per i blocchi che permettono il setting dei sensori

- normvalue: restituisce l'array costitutivo per la lettura dei valori dai sensori configurati
- playsound: restituisce l'array costitutivo per i blocchi che permettono la produzione di suoni
- buttons: restituisce l'array costitutivo per la gestione dei pulsanti del brick

Andando quindi a modificare il metodo di classe "blocksFor:" della classe "RobotModel" è possibile personalizzare i blocchi specifici di SILENT.

Modificare i blocchi significa modificare i valori che ne definiscono aspetto e comportamento nell'array della categoria, prendendo per esempio un frammento di "blocksFor" (Algoritmo 8.7).

Le righe da 5 a 14 corrispondono ciascuna ad un blocchetto che poi verrà visualizzato nella palette "motors": ognuna di esse è un valore di un array in cui il primo campo è la definizione del contenuto (etichetta) del blocchetto, la seconda definisce il tipo di blocchetto, mentre il terzo definisce la classe e il metodo corrispondente per l'esecuzione del blocchetto (inoltre è anche ciò che viene esportato poi nell'XML).

Nel primo campo si trovano dei parametri, preceduti da %, a ciascun dei quali corrisponde una entry nella classe "CommandBlockMorph" al metodo "uncoloredArgMorphFor:" che si occupa di costruire quello che poi diverrà la parte editabile a disposizione dell'utente. Essi sono in sostanza dei placeholders che vengono poi analizzati per essere sostituiti da quanto definito nel metodo apposito (Algoritmo 8.8).

In questo caso si può notare che i metodi di get del robotomodel vengono utilizzati per costruire le selezioni a tendina dei blocchetti: ad esempio la classe "ChoiceArgMorph" viene inizializzato un nuovo oggetto "getOptionsSelector" con l'array di valori restituiti da "#whichMotor", un metodo della classe ScriptableScratchMorph che restituisce un array contenente tutti i motori configurati, inoltre viene detto di mostrare come valore di default, "choice:", il motore nella prima posizione dell'array "((robotmodel getPeripheralsForBlocks: 'motors') at: 1)".

Nel secondo campo viene specificato il tipo di blocco che stiamo definendo, le possiblità sono:

- - nessuna definizione
- b repoter booleano

Algoritmo 8.7 frammento di "blocksFor:" di RobotModel che restituisce la configurazione dei blocchetti per i motori

```
1  [...]
2      (aType='motors')
3          ifTrue: [
4          ^ #('motion'
5              ('start %A power %G direction %B'              -
                   myRobotMotor:power:start:)
6
7              ('start %A power %G direction %B for %n %o stop
                   %E'          -    myRobotMotor:power:start:
                   duration:misuredin:stopby:)
8              ('sync start %O power %G direction %B steering %
                   r'           -    myRobotSyncMotor:power:start:
                   steering:)
9
10             ('sync start %O power %G direction %B for %n %o
                   steering %r stop %E'                -
                   myRobotSyncMotor:power:start:duration:
                   misuredin:steering:stopby:)
11             ('stop %A by %E'                             -
                   myRobotMotor:stopby:)
12             ('sync stop %O by %E'                        -
                   myRobotSyncMotor:stopby:)
13             ('%A is moving?'                             b
                   myRobotismoving:)
14             ('angle of %A in deg'                        r
                   myRobotangleof:)
15             ).
16         ].
17 [...]
```

Algoritmo 8.8 Frammento di codice di "uncoloredArgMorphFor:" dove vengono definiti i parametri inseriti nella configurazione dei blocchetti.

```
#A = code ifTrue: [^ ChoiceArgMorph new getOptionsSelector:
    #whichMotor; choice: ((robotmodel getPeripheralsForBlocks
    : 'motors') at: 1)].
#B = code ifTrue: [^ ChoiceArgMorph new getOptionsSelector:
    #whichMotorDirection; choice: 'forward'].
#E = code ifTrue: [^ ChoiceArgMorph new getOptionsSelector:
    #whichMotorStopMode; choice: 'braking'].
#G = code ifTrue: [^ ExpressionArgMorph new
    numRangeExpression: 50 rangemin: -100 rangemax: 100].
#o = code ifTrue: [^ ChoiceArgMorph new getOptionsSelector:
    #whichMotorRotationDurationMode; choice: 'Degrees'].
#O = code ifTrue: [^ ChoiceArgMorph new getOptionsSelector:
    #whichMotorForSync; choice: ((robotmodel
    getMotorNameForSync) at: 1)].
#n = code ifTrue: [^ ExpressionArgMorph new numExpression:
    '0'].
#r = code ifTrue: [^ ExpressionArgMorph new
    numRangeExpression: 0 rangemin: -100 rangemax: 100].
```

- c blocco contenitore di una sequeza di comandi
- r reporter
- s comando speciale con una sua regola di valutazione
- t comando di tempo, come il wait
- E 'hat' per evento su messaggio
- K key event hat
- M 'hat' per evento su click del mouse
- S 'hat' per l'evento di start

I blocchi più utilizzati sono -, r, b. I blocchi di tipo reporter sono quelli che hanno associato un metodo che ritorna qualche tipo di valore..

Infine il terzo campo è il luogo dove viene associato un metodo che viene richiamato quando un blocco viene eseguito in SILENT e che viene esportato quando avviene la procedura relativa verso l'NXT.

Questi metodi sono metodi d'istanza della classe "ScriptableScratchMorph", raggruppati sotto le categorie che iniziano con "my" (mySensor ops, myMotor ops, ...),

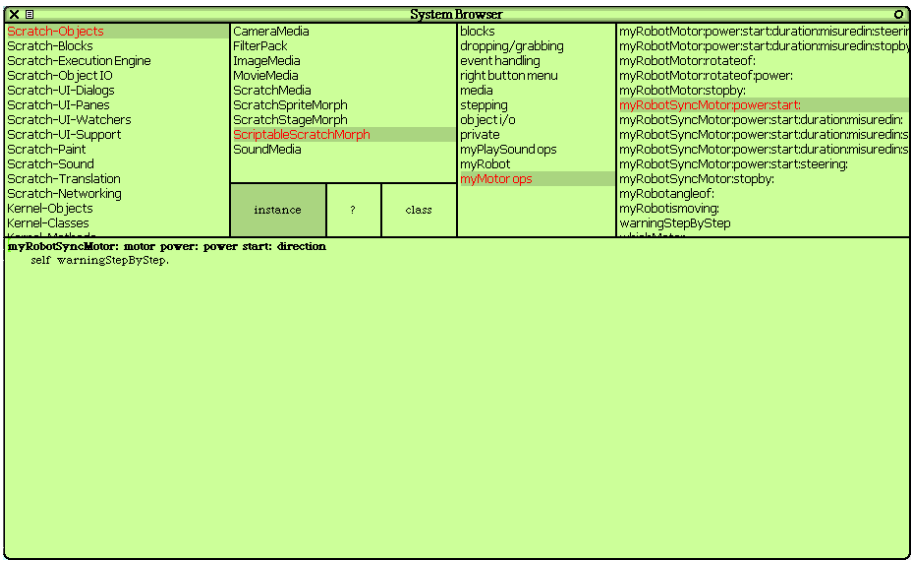


Figura 8.2: Esempio di metodo associato ad un blocco motore

in cui vanno necessariamente definiti per evitare problemi di esecuzione e di esportazione. Senza di essi infatti SILENT\Scratch non trova l'operazione da associare al blocchetto e la stringa da esportare nel codice XML.

Nella versione attuale di SILENT questi metodi sono metodi vuoti, ovvero non compiono alcuna operazione effettiva al momento della loro esecuzione nell'interfaccia, il loro unico scopo è permettere la corretta esportazione in codice XML. Per uno sviluppo futuro, teso ad una maggiore interazione in Scratch e alla realizzazione di una corrispondente animazione nello stage, potranno essere implementati a piacere (Figura 8.2).

Un semplice esempio riassuntivo può essere fatto sul seguente valore di definizione:

('start %A power %G direction %B' – myRobotMotor:power:start:)

Il primo campo definisce che il blocchetto conterrà il testo inserito tra apici con i placeholders sostituiti da:

- %A: elenco dei motori
- %G: campo numerico con valore di default 50 e range in [-100, 100]
- %B: elenco delle direzioni possibili per i motori

Il secondo campo definsice che si tratterà di un blocchetto senza particolari funzioni.

Il terzo campo definisce che ad esso è associato il metodo myRobotMotor:power:start, visibile in Figura 8.2.

Per una panoramica generale sulle modifiche ai blocchi di Scratch è possibile visitare il tutorial sul sito ufficiale [24].

8.3 Esportazione del codice

L'esportazione del codice avviene sfruttando una libreria "jens-ScratchXML" [25], che mette a disposizione la funzionalità di esportazione ed importazione di singoli script in formato XML. La libreria è stata modificata a dovere in modo che si occupi di esportare tutto il progetto presente in SILENT in una sola volta.

La voce di menù Export XML richiama il metodo "exportSXML" della classe "ScratchFrameMorph". Tale metodo si occupa di recuperare tutti gli oggetti presenti negli sprite e nello stage, inizializzare un nuovo ScratchXMLConverter ed infine eportare il codice in XML (Algoritmo 8.9).

L'esportazione degli oggetti avviene in "ScratchXMLConverter:xmlFromSprite2:", in cui vengono analizzati tutti gli oggetti e ad ognuno di essi viene associato un nodo dell'albero XML in costruzione:

- nome del nodo = nome del tipo di oggetto
- attributi del nodo = attributi dell'oggetto: tipo, nome e selettore. Il tipo e il nome sono specificati solo per gli oggetti workplane e script. Il selettore è quel metodo visto in precedenza, definibile per i blocchi personalizzati in "blocksFor", ciascun blocco in Scratch ha associato un selettore univoco.
- valore del nodo = valore dell'oggetto: il/i valore/i del nodo possono essere o altri nodi oppure gli argomenti passati al selettore. Nel primo caso quindi come contenuto del nodo avremmo altri blocchi, nel secondo invece dei campi <argument/> contenenti i valori degli argomenti.

Algoritmo 8.9 Frammento del codice che si occupa di avviare l'esportazione in xml degli oggetti presenti nel progetto di lavoro in SILENT

```
exportSXML
    | xml t1 |
        t1 := self stageAndSprites.
        xml := ScratchXMLConverter new target: t1.
        xml exportSpriteAsXML: xml target.
```

Quando si incontrano oggetti che, per la loro tipologia, ne contengono altri innestati, vengono analizzati ricorsivamente in modo da costruire correttamente l'albero corrispondente.

Quindi a partire dal workplane, si analizzano tutti gli sprite, per ciascuno di essi tutti gli script e quindi tutti i blocchi che appaiono nel sotto albero di oggetti dello script.

Per analizzare i singoli script è stato implementato il metodo di istanza "ScratchXMLConverter:addBlock:toXML:", dove vengono passati il nodo radice da analizzare e l'oggetto XML che conterrà il risultato della traduzione (Algoritmo 8.10).

Questo metodo una volta analizzati tutti gli oggetti presenti nello script ritorna l'oggetto XML costruito al chiamante "xmlFromSprite2" che si occuperà di passare allo script successivo.

Al termine del processo il file XML risultate viene scritto su disco nella directory home dell'utente.

L'operazione di esportazione in codice XML avviene ogni volta che l'utente sceglie di eseguire una delle operazioni di export disponibili.

8.4 Backend di comunicazione

Il backend di comunicazione è realizzato dalla classe BackendEngine.java che viene avviata come thread dalla classe di avvio ScratchBridge.java.

La classe è molto semplice e sfrutta una libreria di comunicazione già implementata da Andrew Davison[26] che implementa a sua volta il Remote Sensor Connection di Scratch 1.3 per instaurare una comunicazione con componenti esterne.

8.4.1 Remote Sensor Protocol

Il Remote Sensors Protocol o Scratch Extension Protocol è un'estensione sperimentale che permette l'interazione tra Scratch (v 1.3+) e gli altri programmi.

Sono supportati tre tipi di interazione:

- messaggi di broadcast
- sensori virtuali

Algoritmo 8.10 Frammento del metodo che si occupa della traduzione degli oggetti nel workplane in codice XML. Nel caso di blocchi che possono contenerne altri (righe 22 e 24) viene richiamato un ulteriore metodo che tratta questi casi particolari e chiama poi a sua volta di nuovo "addBlock:" sui blocchi contenuti in modo da costruire correttamente la struttura annidata del codice.

```
1  addBlock: t1 toXML: t2
2      | t3 t4 |
3      t3 := XMLObject named: 'block'.
4      (t1 isKindOf: EventHatMorph)
5          ifTrue: [t3 attributeAt: 'event' put: t1 eventName
               .].
6      (t1 isKindOf: VariableBlockMorph)
7          ifTrue: [t3 attributeAt: 'variable' put: t1 variable
               ]
8          ifFalse: [(t1 isKindOf: CommandBlockMorph)
9                  ifTrue:
10                     [t4 := t1 selector.
11                     t4 = #< ifTrue: [t4 := #lesserThan].
12                     t4 = #> ifTrue: [t4 := #largerThan].
13                     t4 = #/ ifTrue: [t4 := #dividedBy].
14                     t4 = #& ifTrue: [t4 := #and].
15                     t4 = #| ifTrue: [t4 := #or].
16                     t3 attributeAt: 'selector' put: t4
                          asString.].
17
18     "Aggiungo gli <arguments> ai blocchi"
19     self addArgumentsIn: t1 toXML: t3.
20
21     (t1 isKindOf: CBlockMorph)
22         ifTrue: [self addNestedBlocksIn: t1 toXML: t3].
23     (t1 isKindOf: IfElseBlockMorph)
24         ifTrue: [self addTrueFalseBlocksIn: t1 toXML: t3].
25     t2 addChild: t3.
26     t1 nextBlock ifNotNil: [self addBlock: t1 nextBlock
          toXML: t2].
27     ^ t2
```

- variabili globali di Scratch rese visibili

Questa caratteristica sperimentale è normalmente abilitabile usando il menu che appare col tasto destro del mouse in uno dei due blocchi di sensori. Quando i sensori remoti sono abilitati, Scratch ascolta per connessioni sulla porta 42001.

Quando una connessione è stabilita, i messaggi sono spediti in entrambe le direzioni attraverso il socket di connessione.

Ogni messaggio consiste in un campo della dimensione di 4 byte, col byte più significativo all'inizio, seguito dal messaggio stesso:

<size: 4bytes><msg: size bytes>

Il campo di 4 byte non viene contato nella dimensione del messaggio, quindi un messaggio vuoto è di 4-0 bytes.

Il messaggio fino al primo carattere blank-space (qualsiasi byte <=32) è case-insensitive e viene utilizzato per decidere come manipolare il messaggio.

Il client deve estrarre e controllare la stringa del tipo di messaggio dal messaggio stesso prima di continuare con l'elaborazione. L'insieme delle tipologie di messaggi verrà esteso nel corso del tempo da Scratch, quindi il codice del client deve essere scritto in modo da scartare i messaggi di tipo sconosciuto.

I tipi di messaggi più comuni contengono stringe leggibili costruite con gli elementi seguenti:

- singole parole non quotate (gatto, topo-x)
- stringhe quotate ("una stringa di 5 parole", "embedded", ...)
- numeri (1, -1, 3.14, ...)
- booleani (true o false)

Parole e stringhe sono codificate in UTF-8.

I due tipi di messaggi più utili sono:

- broadcast <string>
- sensor-update <var-name_1> <new-value_1> ...

Un messaggio di sensor-update include una o più coppie di (nomi di variabile, valore). I nomi delle variabili sono stringhe, il loro valore può essere sia numerico che stringa.

Per esempio:

```
sensor-update "note" 60 "seconds" 0.1
broadcast "play note"
```

Il primo messaggio imposta il valore di due sensori virtuali chiamati “note” e “seconds”. Il secondo invia in broadcast il messaggio “play note”. Uno script in Scratch potrebbe rispondere a questo broadcast suonando una nota come specificato nei sensori.

Scratch invia questi due tipi di messaggi quando un broadcast o un cambiamento di variabile globale occorre.

Scratch inoltre risponde anche a questi tipi di messaggi. Un broadcast spedito a Scratch causa un’occorrenza del blocco broadcast. Un messaggio sensor-update causa la variazione dei valori dei sensori virtuali disponibili nel blocco relativo.

8.4.2 Backend engine

Nel progetto di tesi sfruttando la libreria di Dawinson utilizziamo il RSP e i messaggi di broadcast per far comunicare il backend con SILENT.

La classe inizializza un socket di comunicazione sulla porta 42001, sulla quale Scratch è in ascolto, e implementa i due tipi di messaggi standard. Il socket viene attivato automaticamente all’avvio di un progetto in SILENT, senza bisogno di abilitarlo manualmente, modificando i metodi “installNewProject” e “newScratchProject” della classe “ScratchFrameMorph” si è resa l’operazione automatica.

Il metodo “allowForBackEndConnections” della classe “ScratchFrameMorph” si occupa infine di attivare il server.

Come abbiamo detto precedentemente il messaggio di nostro interesse è il primo (broadcast), in quanto lo scopo, almeno per questa prima versione, è quello di impartire da Scratch comandi al BackEnd per avviare la traduzione, la compilazione e il download del bytecode NXC sul robot collegato.

Il backend di comunicazione quindi una volta instaurata la connessione con scratch si mette in ascolto di messaggi di broadcast del tipo “broadcast <Export-NXC|Compile|Compile-upload>”, ricevutone uno valido esegue i metodi previsti ed al loro termine invia un messaggio di broadcast di ritorno a scratch col risultato dell’operazione richiesta.

Attraverso il messaggio di risposta dal BackEnd a Scratch vengono anche inviati eventuali dettagli su errori incontrati nell’esecuzione dell’operazione richiesta, in

Algoritmo 8.11 "ScratchStageMorph:receiveNarrowcastNamed:" metodo modificato in SILENT che si occupa di gestire i messaggi ricevuti dal backend . Questo metodo viene richiamato dallo ScratchServer ogni qualvolta riceve messaggi narrowcast, che non sono altro che dei messaggi broadcast modificati "ad hoc" per lo scopo del progetto.

```
1  receiveNarrowcastNamed: t1
2          | t2 t3 t6 |
3          t5 := 'NXC'
4          t6 := 'Generic Error: '.
5          t3 := t1 indexOf: $:.
6          t3 > 0
7                  ifTrue: [t2 := t1 allButFirst: t3 + 1]
8                  ifFalse: [t2 := t1].
9          (t1 beginsWith: t5)
10                 ifTrue: [owner compilationMessage: t1].
11         (t1 beginsWith: t6)
12                 ifTrue: [[DialogBoxMorph warn: (t2
                      withNoLineLongerThan: 40)]
13                                 forkAt: Processor
                                      userBackgroundPriority].
```

modo da semplificare la risoluzione dei problemi. A tale scopo è stato modificato il server di connessione in Scratch in modo che possa trattare correttamente i messaggi ricevuti dal BackEnd e mostrarne il contenuto in popup nell'interfaccia utente.

Le modifiche al server hanno interessato il metodo che gestisce la ricezione di messaggi (Algoritmo 8.11), in modo che possa gestire casi di messaggi generici o provenienti dal processo di esportazione in NXC. Nel primo caso viene semplicemente aperto un popup di notifica, mentre nel secondo viene richiamato un metodo che si occupa di effettuare un ulteriore parsing della stringa contenuta nel broadcast per ricavare ulteriori informazioni e quindi mostrarle in maniera corretta all'utente.

Quindi per i messaggi ricevuti da SILENT, possono accadere:

- broadcast "NXC <'NXC Compilation Status: '|'NXC Compilation Error: '|'NXC Compilation Ok: '|'NXC Compilation Finished.'|NXC Exporting Status: '|'NXC Exporting Finished.'|'NXC Exporting Error: '><messaggio>"
- broadcast "Generic Error: <messaggio>"

Algoritmo 8.12 Costruttore di ReadXMLFile.java

```
1  public ReadXMLFile(){
2          try {
3              File fXmlFile = new File(portability.
                   getUser_home() + portability.getSlash() + "
                   myRobotSS.xml");
4              DocumentBuilderFactory dbFactory =
                   DocumentBuilderFactory.newInstance();
5              DocumentBuilder dBuilder = dbFactory.
                   newDocumentBuilder();
6              doc = dBuilder.parse(fXmlFile);
7              doc.getDocumentElement().normalize();
8          } catch (Exception e) {
9              e.printStackTrace();
10         }
11     }
```

8.5 Traduzione da XML a NXC.

Il processo di traduzione viene avviato dalla classe BackendEngine.java nel momento in cui riceve un messaggio da Scratch di esportazione o compilazione.

Il processo vero e proprio è implementato nella classe ReadXMLFile.java (Algoritmo 8.12), il cui costruttore legge dal file "MyRobotSS.xml" il contenuto XML del progetto da esportare e ne crea un DOM Document pronto per essere elaborato.

Un volta creato un oggetto "ReadXMLFile" su di esso sarà possibile richiamare il metodo "emitNXCCode" che si occupa della traduzione da codice XML a codice NXT:

- effettua il parsing del DOM Document e ne estrae il codice da esportare in NXC
- crea un file sorgente NXC per ogni sprite utilizzato nell'interfaccia Scratch di SILENT
- per ogni file sorgente ne definisce l'header che conterrà essenzialmente gli include necessari al funzionamento del codice.
- per ogni file sorgente ne scrive il rispettivo codice sorgente tradotto
- chiude e salva i file prodotti.

La generica struttura del file NXC sorgente che viene generato può essere rappresentata nel modo illustrato vell'Algoritmo 8.13.

Il parsing del DOM Document avviene con una visita in profondità dell'albero XML da tradurre, dal workspace <scratchWorkplane/> si analizzerà quindi il primo sprite <scratchSprite/>, il primo script <scratchScript/> e il suo primo blocco. Di ciascun blocco, se necessario, verrà ricorsivamente analizzato il contenuto degli argument alla ricerca di contenuti complessi come variabili o funzioni.

Ciascun <scratchSprite/> è caratterizzato da due attributi:

- type: che è sempre "a ScratchSpriteMorph()"
- name: che invece è inserito a piacere dall'utente e che identificherà poi il nome del file sorgente NXC che verrà prodotto.

Di seguito ora verranno analizzati tutte le tipologie di blocchi che possono essere incontrati e come vengono tradotte. Ciascun blocco è rappresentato da un tag <block/> e da un attributo event (solo per i blocchi task) o selector (per tutti gli altri), l'attributo selector è quello utilizzato per discriminare i vari tipi di blocchi.

Quando di seguito nella traduzione in codice NXC viene riportato un tag, ad esempio <argument/>, questo significa che viene applicata un'ulteriore elaborazione ricorsiva del nodo per tradurre ciò che è contenuto all'interno. In caso di <argumentN/>, con N numero naturale, si vuole indicare per semplicità i diversi argomenti. Nel file XML in realtà non sono discriminati cosi esplicitamente.

8.5.1 Bocchi task

I blocchi che identificano un task sono sempre i primi a comparire nella sequenza di tag all'interno di <scratchSprite/>.

Di essi viene recuperato il contenuto dell'attributo "event": se contiene la stringa riservata "Scratch-StartClicked" allora si tratta dello script che corrisponde al task main, altrimenti si tratta di un task generico, e come tale verrà tradotto.

8.5.2 Blocchi di controllo del flusso

doIf

- <block selector="doIf"><argument/></block>

Algoritmo 8.13 "template" del sorgente NXC generato dalla traduzione. In [...] vengono inseriti i frammenti di codice tradotto. Con <task1>,<task2> si vuole indicare l'eventuale presenza di altri task oltre al "main".

```
myRobot_<nome sprite>.nxc:
        /*
                Automatically generated code from SILENT Project.
                Code elaborated from /home/luca/myRobotSS.xml
                Author: Luca Zenatti
                Version: 0.5
        */
        //include
        #include  "define.nxc"
        #include  "array_0.5.nxc"
        #include  "mylib.nxc"

        //tasks prototypes
        [...]
        task main () {
                //sezione dichiarazione variabili e array
                [...]

                //corpo del codice
                [..]
        }

        task <task1> () {
                //sezione dichiarazione variabili e array
                [...]

                //corpo del codice
                [...]
        }
        task <task2> () {
                //sezione dichiarazione variabili e array
                [...]

                //corpo del codice
                [...]
        }
        [...]
```

- if(<argument/>){ }

Per blocco condizionale verrà poi effettuato il parsing dei blocchi presenti dopo <argument/> che verranno quindi inseriti tra le parentesi graffe. Questo comportamento è da tenere presente per tutti i blocchi che prevedono "sottoblocchi annidati".

doIfElse

- <block selector="doIfElse"><argument/><true></true><false></false></block>
- if(<argument/>){<true/>} else {<false/>}

doWaitUntil

- <block selector="doWaitUntil"><argument/></block>
- while(!(<argument/>)){ }

doForeverIf

- <block selector="doForeverIf"><argument/></block>
- while(<argument/>){ }

wait:elapsed:from:

- <block selector="wait:elapsed:from:"><argument/></block>
- Wait(<argument/>)

doForever

- <block selector="doForever"></block>
- while(true){ }

doRepeat

- <block selector="doRepeat"><argument/></block>
- for(int _i=0; _i< <argument/>; _i++) { }

broadcast

- <block selector="broadcast:"><argument/></block>
- start <argument/>;
  add task prototyping;

myRobotdoReturn

- <block selector="myRobotdoReturn:"><argument/></block>
- StopTask(<argument/>)

stopAll

- <block selector="stopAll"/>
- StopAllTasks()

8.5.3 Blocchi operator

+ - *

- <block selector="+"><argument1/><argument2/></block>
- <argument1/> + <argument2/>

dividedBy <block selector="dividedBy:"><argument1/><argument2/></block>
<argument1/> / <argument2/>

randomFrom:to

- <block selector="randomFrom:to:"><argument1/><argument2/></block>
- Random(<argument1/> - <argument2/>) + <argument2/>

=

- <block selector="="><argument1/> <argument2/></block>
- <argument1/> == <argument2/>

lesserThan

- <block selector="lesserThan:"><argument1/> <argument2/></block>
- <argument1/> < <argument2/>

largerThan

- <block selector="largerThan:"><argument1/> <argument2/></block>
- <argument1/> > <argument2/>

AND | OR

- <block selector="and"><argument1/> <argument2/></block>
- (<argument1/> && <argument2/>)
- (<argument1/> || <argument2/>)

NOT

- <block selector="not"><argument/></block>
- (!(<argument/>))

concatenate:with:

- <block selector="concatenate:with:"><argument1/><argument2/></block>
- StrCat(<argument1/>,<argument2/>)

letter:of:

- <block selector="letter:of:"><argument1/><argument2/></block>
- ByteArrayToStr (StrIndex (<argument1/> , <argument2/>))

stringLength

- <block selector="stringLength:"><argument/></block>
- StrLen(<argument/>)

rounded

- <block selector="rounded"><argument/></block>
- myRound(<argument/>)

computeFunction:of

- <block selector="computeFunction:of:"><argument1/><argument2/></block>

Attraverso una mappa delle funzioni presenti in Scratch e quelle presenti in NXC viene recuperata la corretta ed inserita in

<argument1/> (<argument2/>)

Mappa per <argument1/>:

- Scratch MATH function: "abs,sqrt,sin,cos,tan,asin,acos,atan,ln,log,e^,10^";
- NXC MATH functions: "abs,sqrt,sin,cos,tan,asin,acos,atan,log,log10,exp,pow(10,x)";

8.5.4 Blocchi variabili

setVar:to:

- <block selector="setVar:to:"><argument1/><argument2/></block>

In questo caso viene chiamato un metodo parseVariableValue(node, var) che, se la variabile non è ancora stata tipizzata, si occupa di effettuare un riconoscimento attraverso l'uso di espressioni regolari del tipo di variabile incontrata, ovvero che sia essa string o float, a partire dal contenuto presente nel secondo <argument/>. Una volta tipizzata la variabile viene preparata la dichiarazione della stessa e messa nel pool di variabili da dichiarare nel task in esame. Tali dichiarazioni verranno poi inserite nel codice NXC nella posizione opportuna.

Se invece la variabile è già stata analizzata, e quindi tipizzata e inserita nel pool delle dichiarazioni, non viene eseguito nulla.

Per gestire la "raccolta" di variabili è stata creata una classe "Variables.java" che si occupa appunto di raccogliere, organizzare e ritornare il pool di variabili da dichiarare.

Al termine comunque viene tradotto, a seconda:

- <argument1/> = <argument2/>;
- <argument1/> = "<argument2/>";

changeVar:by:

- <block selector="changeVar:by:"><argument1/><argument2/></block>

Dopo aver riconosciuto il tipo di variabile di cui viene chiesto il cambiamento, sempre attraverso il metodo di tipizzazione, viene tradotto:

- <argument1/> += <argument2/>;
- <argument1/> = concat(<argument1/>, <argument2/>);

initArray:

- <block selector="initArray:type:d1:d2:d3:d4:"><argument1/>
  <argument2/><argument3/> <argument4/><argument5/>
  <argument6/></block>

Viene dichiarato, quindi inserito nel pool di variabili da dichiarare, il seguente array:

- <argument2/> <argument1/>[<argument3/>][<argument4/>]
  [<argument5/>][<argument6/>]

In questo caso non c'è necessità di un metodo che si occupi della tipizzazione in quanto il blocco di init in Scratch richiede di esplicitarne il tipo.

Le dimensioni vengono dichiarate solo se sono diverse da -1.

setArrayValue

- <block selector="setArrayValue:d1:d2:d3:d4:value:"><argument1/>
  <argument2/><argument3/> <argument4/><argument5/><argument6/></block>
- <argument1/>[<argument2/>][<argument3/>][<argument4/>]
  [<argument5/>] = <argument6/>;

Anche qui la presenza delle dimensioni è condizionata dal loro valore diverso da -1.

getArrayValue

- <block selector="getArrayValue:type:d1:d2:d3:d4:"><argument1/>
  <argument2/><argument3/> <argument4/><argument5/><argument6/></block>

A seconda di quando scelto in <argument2/>:

- getArrayNumValue(<argument1/>[<argument3/>][<argument4/>] [<argument5/>][<argument6/>])
- getArrayStringValue(<argument1/>[<argument3/>][<argument4/>] [<argument5/>][<argument6/>])

Anche qui la presenza delle dimensioni è condizionata dal loro valore diverso da -1.

lengthArray:

- <block selector="legthArray:d1:d2:d3:d4:"><argument1/> <argument2/> <argument3/> <argument4/><argument5/></block>
- ArrayLen(<argument1/>[<argument2/>][<argument3/>] [<argument4/>][<argument5/>])

Anche qui la presenza delle dimensioni è condizionata dal loro valore diverso da -1.

8.5.5 Blocchi motore

NOTA: in caso di OUT_<argument1/> in realtà viene recuperata la porta a cui è collegato il motore di etichetta <argument1/>

myRobotMotor:power:start:

- <block selector="myRobotMotor:power:start:"><argument1/><argument2/> <argument3/></block>

A seconda di <argument3/>

- OnFwd(OUT_<argument1/>, <argument2/>)
- OnRev(OUT_<argument1/>, <argument2/>)

myRobotMotor:power:start:duration:misuredin:stopby:

- <block selector="myRobotMotor:power:start:duration:misuredin:stopby:"> <argument1/><argument2/> <argument3/> <argument4/><argument5/> <argument6/></block>

A seconda di <argument5/>

- [degrees] a seconda di <argument3/>
  - [forward] RotateMotorEx(OUT_<argument1/>, <argument2/>, <argument4/>, 0, false, false)
  - [backward] RotateMotorEx(OUT_<argument1/>, <argument2/>, - <argument4/>, 0, false, false)
- [rotations] a seconda di <argument3/>
  - [forward] RotateMotorEx(OUT_<argument1/>, <argument2/>, <argument4/>*360, 0, false, false)
  - [backward] RotateMotorEx(OUT_<argument1/>, <argument2/>, - <argument4/>*360, 0, false, false)
- [seconds] a seconda di <argument3/>
  - [forward] OnFwd(OUT_<argument1/>, <argument2/>)
  - [backward] OnRev(OUT_<argument1/>, <argument2/>)

  Wait(abs(<argument4/>)*1000);

A seconda di <argument6/>:

- [braking] Off(OUT_<argument1/>)
- [coasting] Coast(OUT_<argument1/>)

myRobotSyncMotor:power:start:steering:

- <block selector="myRobotSyncMotor:power:start:steering:"><argument1/>

  <argument2/> <argument3/><argument4/></block>

A seconda di <argument3/>

- [forward] OnFwdSyncEx(OUT_<argument1/>,<argument2/>,<argument4/>, RESET_ALL)
- [backward] OnRevSyncEx(OUT_<argument1/>,<argument2/>,<argument4/>, RESET_ALL)

myRobotSyncMotor:power:start:duration:misuredin:steering:stopby:

- <block selector="myRobotSyncMotor:power:start:duration:misuredin:steering:stopby:"> <argument1/> <argument2/><argument3/><argument4/> <argument5/><argument6/> <argument7/></block>

A seconda di <argument5/>

- [degrees] a seconda di <argument3/>
  - [forward] RotateMotorEx(OUT_<argument1/>, <argument2/>, <argument4/>, 0, true, false)
  - [backward] RotateMotorEx(OUT_<argument1/>, <argument2/>, - <argument4/>, 0, true, false)
- [rotations] a seconda di <argument3/>
  - [forward] RotateMotorEx(OUT_<argument1/>, <argument2/>, <argument4/>*360, 0, true, false)
  - [backward] RotateMotorEx(OUT_<argument1/>, <argument2/>, - <argument4/>*360, 0, true, false)
- [seconds] a seconda di <argument3/>
  - [forward] OnFwdSyncEx(OUT_<argument1/>, <argument2/>)
  - [backward] OnRevSyncEx(OUT_<argument1/>, <argument2/>)

  Wait(abs(<argument4/>)*1000);

A seconda di <argument6/>:

- [braking] Off(OUT_<argument1/>)
- [coasting] Coast(OUT_<argument1/>)

myRobotMotor:stopby:

- <block selector="myRobotMotor:stopby:"><argument1/> <argument2/></block>

A seconda di <argument2/>

- [braking]: Off(OUT_<argument1/>)
- [coating]: Coast(OUT_<argument1/>)

myRobotSyncMotor:stopby:

- <block selector="myRobotSyncMotor:stopby:"><argument1/>
  <argument2/></block>

A seconda di <argument2/>

- [braking]: Off(OUT_<argument1/>)
- [coating]: Coast(OUT_<argument1/>)

myRobotismoving:

- <block selector="myRobotismoving:"><argument/></block>
- isMoving(OUT_<argument1/>)

myRobotangleof:

- <block selector="myRobotangleof:"><argument/></block>
- MotorRotationCount(OUT_<argument1/>)

8.5.6 Blocchi sensore

myRobotSetSensor:type:mode:

- <block selector="myRobotSetSensor:type:mode:"><argument1/>
  <argument2/><argument3/></block>

A seconda del sensore:

- SetSensorType(S<argument1/>, <argument2/>)
- SetSensorMode(S<argument1/>, <argument3/>)

myRobotSetSensor:

- <block selector="myRobotSetSensor:"><argument1/></block>

Valido solo per il sensore ad ultrasuoni:

- SetSensorUltrasonic(S<argument1/>)

myRobotSensorReset:

- <block selector="myRobotSensorReset:"><argument1/></block>
- ResetSensor(S<argument1/>)

myRobotPressButton:

- <block selector="myRobotPressButton:"><argument1/></block>
- ButtonPressed(<argument1/>, true)

myRobotSensorValue:

- <block selector="myRobotSensorValue:"><argument1/></block>
- Se sensore ultrasuoni: SensorUS(S<argumeng1/>)
- Altrimenti: Sensor(S<argument1/>)

8.5.7 Blocchi Sound

myRobotPlaySound:duration:

- <block selector="myRobotPlaySound:duration:"><argument1/> <argument2/></block>
- SetSoundDuration(<argument2/>); Playsound(<argument2/>);

Conclusioni

Facciamo infine un rapido bilancio del lavoro svolto e delle possibilità che presenta per eventuali ampliamenti e approfondimenti. Vediamo in breve i risultati ottenuti e gli ostacoli incontrati, la cui risoluzione potrebbe permettere determinati sviluppi futuri.

Risultati

Dai test effettuati al termine delle implementazioni, SILENT è risultato semplice da utilizzare: l'immediatezza del paradigma a blocchi di Scratch fa diventare naturale costruire software e provare ad immaginare quali sviluppi possano essere realizzati. Il trasferimento del lavoro sul robot NXT e la trasformazione di un concetto astratto come un flusso di istruzioni in azioni reali rende ancor più divertente e stimolante stendere righe di "codice".

SILENT quindi può essere una scelta valida per quegli educatori, studenti o anche semplici appassionati che vogliano avvicinarsi al mondo della robotica con un approccio divertente, ma potrà in breve tempo diventare anche luogo di studio e approfondimento delle tematiche che ruotano attorno alla programmazione e al controllo di sistemi robotici.

A tal proposito sarebbe un buon campo di prova poter effettuare dei test di utilizzo di SILENT in ambito reale, ovvero mettendo a disposizione il software a docenti (e alle loro classi) e a tutti gli utenti interessati all'argomento. Questo potrebbe essere realizzato da una parte cercando qualche contatto con il mondo della scuola e dall'altra rendendo scaricabile il programma attraverso la implementazione di un piccolo sito, il quale potrebbe contenere anche la documentazione e tornare utile come fonte di feedback.

Nel corso della tesi è stato utilizzato soltanto un modello di robot, ovvero un veicolo con due ruote motrici e 4 sensori, tuttavia SILENT e LEGO NXT permettono di

lavorare su modelli di robot completamente diversi: umanoidi, macchine automatiche, animali (solo per citare i modelli portati come esempio dalla LEGO stessa). Per quanto alcuni dei blocchi di controllo dei motori siano specializzati per il modello veicolare, è possibile utilizzare i blocchi più semplici o addirittura pensare di implementare altri blocchi adatti a modelli diversi. La documentazione del manuale tecnico ha lo scopo fornire le basi per poter affrontare la modifica di SILENT in tutte le sue parti.

La scelta di utilizzare l'XML come formato di scambio tra Scratch ed il Backend è motivata dalla volontà di separare nettamente le due parti, questo permette di rendere più flessibile SILENT a modifiche che interessando l'una o l'altra, per esempio: modifiche o sostituzione della traduzione, modifiche ai blocchetti sull'interfaccia utente, implementazione di nuovi moduli, cambiamento dell'hardware del robot, per arrivare fino alla sostituzione completa di una delle parti. Il formato in questione si presta molto bene al ruolo di interscambio di dati: permette di definire e controllare il significato degli elementi contenuti nel file e permette di scambiare informazioni tra sistemi che accettano e gestiscono dati con formati incompatibili tra loro. Infine essendo un formato aperto può essere utilizzato da chiunque per qualsiasi scopo.

Problematiche aperte

La problematica aperta più importante è quella legata a come poter sfruttare lo stage, che in Scratch viene utilizzato per "animare" il programma appena scritto.

In Scratch ciascuno sprite è rappresentato nello stage da un'icona grafica, collegata a particolari blocchi (movimento, looks, pen) che, all'esecuzione del flusso del programma, "animano" lo stage facendole compiere delle azioni (spostamenti, rotazioni, stampa di frasi, richieste di input, disegno di linee).

Un'idea potrebbe essere quella di rendere lo stage una rappresentazione virtuale del modello di robot implementato, e di renderlo quindi una sorta di anteprima di quanto poi il robot andrà a fare nelle realtà. Questo spunto, per quanto interessante, si accompagna a numerose difficoltà legate al gap tra modello fisico reale e quello virtuale (per sempio: lettura dei sensori, imprecisioni), alle limitazioni di ciò che è rappresentabile nello stage e quello che invece è realizzabile fisicamente (per esempio: dimensioni dello stage, oggetti extra-modello come gli ostacoli).

Sviluppi futuri

Gli sviluppi che si possono prevedere su SILENT sono numerosi:

- connessione bluetooth: rendere disponibile il collegamento tra elaboratore e NXT attraverso la tecnologia bluetooth consentirebbe una maggiore comodità nel trasferimento dei dati tra i dispositivi. Oltretutto questa connessione è disponibile nella maggior parte dei software che permettono di lavorare con LEGO NXT, cosa che rappresenterebbe un ostacolo in meno per avvicinare gli utenti a SILENT.
- comunicazione bidirezionale: la versione attuale SILENT supporta una comunicazione monodirezionale, ovvero non viene gestito l'invio della lettura dei parametri dal robot NXT all'elaboratore. La realizzazione della bidirezionalità consentirebbe di poter utilizzare l'interfaccia utente del programma come monitor remoto del robot e addirittura pensare di integrare una separazione dell'elaborazione tra NXT e SILENT.
- blocchi avanzati: la palette "advanced" è dedicata proprio alla possiblità di sviluppare blocchi avanzati per funzioni NXC o per modelli di robot particolari. Ad esempio potrebbe essere plausibile inserire all'interno di questa sezione blocchi che realizzano procedure complesse per gestire robot umanoidi.
- integrazione con BYOB: come è stato illustrato nel corso dell'elaborato sono molti gli aspetti interessanti di questa modifica di Scratch, sarebbe quindi utile per ampliare la gamma di utilizzi di SILENT introdurre la ricorsione e la costruzione di blocchi personalizzata.
- upgrade firmware da GUI: in Enchanting esiste la possiblità di aggiornare il firmware attraverso l'interfaccia utente, la stessa cosa potrebbe essere utile averla anche su SILENT, cosicché sia possibile evitare di installare altro software in caso si volesse procedere ad un upgrade.
- tool di scrittura XML del modello robot: ad ora per modellare il robot è necessario scrivere l'XML relativo in modalità testuale, cosa che richiede una certa preparazione tecnica e dimestichezza con la sintassi. Potrebbe essere interessante la realizzazione di un tool grafico per modellare il dispositivo autonomo che realizzi in maniera completamente trasparente l'XML del modello.
- integrazione di robot diversi: per come è stato progettato SILENT è possibile pensare di integrare robot di natura diversa da quelli LEGO MINDSTORM

NXT, a seconda dell'integrazionenecessaria potrebbe essere sufficiente modificare soltanto la traduzione e la comunicazione backend-robot per avere a disposizione un sistema funzionante.

- sharing: Scratch prevede la possiblità di condividere i progetti realizzati sul sito ufficiale del programma, potrebbe essere interessante seguire questa filosofia di condivisione realizzando uno spazio di comune in cui i progetti implementati possano essere messi a disposizione degli altri utenti.

Difficoltà incontrate

Una delle maggiori difficoltà è stata la scelta dei blocchi da implementare, ovvero il livello di astrazione a cui portare le funzioni NXC: tenendo presente il target medio al quale è rivolto SILENT, si è cercato di trovare un compromesso (a priori) tra semplicità, disponibilità di controllo e rappresentazione del linguaggio NXC. Sarebbe comunque una buona cosa prevedere delle sessioni di test con esperti di didattica e con i bambini stessi per poter avere un riscontro sulle scelte effettuate ed eventualmente apportare le dovute modifiche.

L'astrazione della sintassi NXC e nel contempo il dover imporre dei vincoli alla libertà di azione di Scratch hanno portato a:

- restringere la presenza di un solo blocco "When green flag" per ogni sprite, in quanto a questo blocco "cappello" è stato associato il task main di ciascun programma NXC;
- associare uno sprite per un programma NXC, immaginando così uno sprite per un dispositivo robotico o una diversa implementazione dello stesso dispositivo;
- l'eliminazione delle liste di Scratch, nelle quali la mancanza di tipizzazione e di strutturazione le rendevano troppo distanti dal modello in NXC, il quale non prevede la presenza di tali strutture dati ma di array tipizzati e di dimensioni stabilite. Sono quindi state sostituite da un'astrazione degli array, così come sono conosciuti in NXC.

Le variazioni apportate a Scratch sono state portate avanti in carenza di una vera documentazione a riguardo, lo sviluppo di modifiche al programma originale non sono supportate da guide ufficiali o indicazioni esplicite sulla strutturazione del codice. Quanto si trova sul wiki ufficiale di Scratch è per la maggior parte dei casi

superficiale o utile soltanto per piccole variazioni/implementazioni. L'unico punto di riferimento è il forum, in cui utenti che si cimentano con gli sviluppi condividono esperienza e domande, tuttavia rimangono informazioni frammentate e disperse. Per questo si è reso necessario molto lavoro di studio sul codice sorgente di Scratch e di Enchanting con una ricostruzione del comportamento dei due software per le classi legate ai blocchetti.

Ringraziamenti

(o presunti tali)

Al Prof. Michele Moro per avermi seguito con precisione, consigliato la direzione da intraprendere e corretto il tiro nel momento opportuno, al di là degli intoppi e delle numerose difficoltà.

Alla mia famiglia: fun-da-men-ta-le!

A tutti gli amici, le mitiche colleghe (e colleghi) e i compagni di avventura universitaria: ognuno di voi mi ha accompagnato per un pezzo più o meno lungo di strada, dando il proprio contributo a tutto questo.

Alla mia editrice di fiducia: hai trasformato una cozzaglia di parole riconducibili alla lingua italiana in un testo accettabile. Eccoti i soldi per la pelliccia!

A me stesso.

Bibliografia

[1] http://en.wikipedia.org/wiki/Educational_software

[2] http://en.wikipedia.org/wiki/Leapfrog_Enterprises_Inc

[3] http://www.alice.org

[4] http://en.wikipedia.org/wiki/PythonTurtle

[5] http://it.wikipedia.org/wiki/Lego_Mindstorms_NXT

[6] http://www.squeak.org/About/

[7] http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC_Guide.pdf

[8] http://en.wikipedia.org/wiki/Bash

[9] http://www.smalltalk.org/

[10] "What's NXT? LEGO Group Unveils LEGO MINDSTORMS NXT Robotics Toolset at Consumer Electronics Show" (Press release). Las Vegas - NV: Lego Group. January 4 2006. Retrieved 2007-09-17.

[11] "8527Mindstorms NXT Kit". Mindstorms Website. Lego Group. Retrieved 2008-12-26.

[12] "LEGO MINDSTORMS Education NXT Base Set". LEGO Education Website. Lego Group. Retrieved 2011-09-30.

[13] LEGO MINDSTORMS NXT Hardware Developer Kit.pdf

[14] LEGO® MINDSTORMS® NXT Executable File Specification

[15] Programming for all. mitcheLResnicK, John maLoneY, anDRés monRoY-heRnánDez, nataLie RusK, eVeLYn eastmonD, KaRen BRennan, amon miLLneR, eRic RosenBaum, JaY siLVeR, BRian siLVeRman, anDYasmin Kafai. Communication of the ACM. 2009

[16] http://infoscratch.media.mit.edu/Educators

[17] http://info.scratch.mit.edu/it/Support/Reference_Guide_1.4

[18] http://wiki.scratch.mit.edu/wiki/Remote_Sensor_Connections

[19] http://wiki.scratch.mit.edu/wiki/JoyTail

[20] http://wiki.scratch.mit.edu/wiki/Remote_Sensors_Protocol

[21] http://info.scratch.mit.edu/WeDo

[22] http://enchanting.robotclub.ab.ca/tiki-index.php

[23] http://fivedots.coe.psu.ac.th/ad/jg/index.html

[24] http://wiki.scratch.mit.edu/wiki/Modding_Tutorial

[25] http://scratch.mit.edu/projects/Jens/93504

[26] http://scratch.mit.edu/forums/viewtopic.php?id=9458

[27] http://www.chirp.scratchr.org/

[28] http://wiki.scratch.mit.edu/wiki/Scratch_Wiki:Table_of_Contents/Blocks

[29] http://seaside.citilab.eu/scratch/arduino