ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA
CORSO LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

TESI DI LAUREA
in
FONDAMENTI DI INTELLIGENZA ARTIFICIALE M

UN AMBIENTE INTEGRATO BASATO SUL CALCOLO DEGLI EVENTI PER IL RICONOSCIMENTO DI POSE MEDIANTE KINECT

CANDIDATO

Steven Cadalt

RELATORE
Chiar.ma Prof.ssa Ing. Paola Mello

CORRELATORI
Dr. Stefano Bragaglia
Dr. Federico Chesani

Anno Accademico 2012/2013
Sessione III

Sommario

Introduzione

La riabilitazione fisica, in particolare la fisioterapia, è un importante strumento a disposizione della medicina per recuperare lo stato di salute in seguito a disturbi o danni provocati da patologie di varia natura. Essa consente, attraverso tecniche ti rieducazione motoria, funzionale, cardiologica, tecniche di elettroterapia, ginnastica vascolare e molte altre, il recupero funzionale da parte del paziente.
In particolare nel nostro studio ci siamo concentrati sulle disabilità motorie tipiche di soggetti che hanno subito un intervento o che a causa dell'età avanzata abbiano una serie di patologie che non consentono loro di avere un efficienza funzionale completa.

Immaginiamo il caso specifico di un anziano che in seguito ad un incidente sia costretto a sottoporsi ad un intervento chirurgico e che, nelle fasi successive, non abbia recuperato completamente le capacità fisiche che aveva prima dell'incidente.
Sarà quindi necessario, dopo che il paziente sarà stato dimesso, provvedere a fargli svolgere una serie di attività volte al recupero delle funzionalità perse.

E' qui che si inserisce la fisioterapia, come branca della medicina che si occuperà dell'anziano, accompagnandolo nel suo periodo di riabilitazione.

Generalmente il paziente viene indirizzato verso l'Azienda Sanitaria Locale (ASL) di competenza, oppure presso una struttura privata dove verranno messe in atto quelle pratiche riabilitative da parte di professionisti.
L'anziano dovrà recarsi quindi con una cera regolarità agli appuntamenti con il fisioterapista, con tutti gli inconvenienti dovuti dal doversi spostare presso la struttura scelta. Per velocizzare e ottimizzare

il processo di guarigione, lo specialista inoltre potrebbe voler istruire il paziente andando a specificare diversi tipi di esercizi che quest'ultimo dovrà svolgere in autonomia presso la propria abitazione, in assenza di una persona specializzata che lo segua nell'esecuzione dei movimenti.

Qui appaiono bene tutta un'altra serie di inconvenienti: l'anziano potrebbe non svolgere gli esercizi che gli sono stati assegnati, a causa per esempio di un eccessivo dolore residuo, oppure, peggio ancora, eseguire gli esercizi in maniera scorretta, rischiando di peggiorare la situazione o facendo insorgere altre patologie.
Inoltre un soggetto poco "diligente" potrebbe non svolgere gli esercizi con la frequenza e modalità specificate dal fisioterapista, ottenendo problemi assimilabili a quelli descritti sopra.

Inoltre lo spostarsi verso una struttura specializzata e la sua frequentazione comporta dei costi da parte del paziente (e da parte del Servizio Sanitario Nazionale se ci riferiamo ad una struttura pubblica) che possono lievitare considerevolmente se il periodo di riabilitazione richiesto risulta essere particolarmente lungo.

E' qui che il gruppo di lavoro sull' intelligenza artificiale si inserisce, cercando di ridurre questo tipo di problematiche, andando ad eliminare alla base alcuni degli elementi che sono causa delle inefficienze descritte.

Sarebbe utile avere un qualche strumento che consenta di far svolgere al soggetto presso la propria abitazione gli esercizi riabilitativi, con la capacità di definire e monitorare, da parte dello specialista, il comportamento del paziente e poter eventualmente attuare interventi specifici in base all'andamento della terapia.
Lo strumento dovrebbe consentire lo svolgere gli esercizi in modo completamente autonomo, e di valutare automaticamente la bontà delle pose effettuate dando al paziente un feedback su come sta svolgendo l'attività prescritta.

In questo modo il paziente potrebbe recarsi con meno frequenza dal fisioterapista in quanto il controllo della corretta esecuzione degli esercizi verrebbe fatta automaticamente dal sistema senza nessun supporto umano; il soggetto affetto dalla patologia avrebbe un riscontro in tempo reale sulla qualità della propria esecuzione degli esercizi, venendo quindi spronato a fare del proprio meglio per effettuarli il più

correttamente possibile; inoltre le potenzialità di raccoglimento dei dati sulle pose eseguite potrebbero essere utilizzate dallo specialista per valutare la bontà e la frequenza con la quale gli esercizi sono eseguiti e per andare eventualmente a modificare la terapia in modo specifico.

Il mio lavoro è stato quello di sviluppare una serie di funzionalità di sistema che consentissero allo strumento di rispettare i requisiti e risolvere le problematiche analizzate; in particolare sono stati definiti ed implementati questi elementi:

- Un *linguaggio* sufficientemente potente da poter esprimere in tutte le sue parti gli esercizi
- Un *editor*, che offra un interfaccia rivolta al fisioterapista che gli consenta, assistendolo nella fase di scrittura, di definire gli esercizi più adatti per il paziente utilizzando il linguaggio di cui sopra
- Un *motore*, ossia un qualche meccanismo che vada a valutare gli esercizi svolti dal paziente e li confronti con quelli proposti dallo specialista

Si è inoltre utilizzato un dispositivo in grado di rilevare le pose svolte dai soggetti che debbono essere monitorati.

Il nostro lavoro si è rivolto ad identificare le tecnologie e sviluppare gli strumenti che siano in grado di adempiere alle funzioni di cui sopra, in particolare:

- E’ stato in primo luogo analizzato il problema della riabilitazione, estraendone le problematiche e le inefficienze intrinseche, andando a descrivere ad alto livello quale sarebbe stata la soluzione ai problemi del caso in esame
- Sono stati scorporati i componenti da implementare, andando a descrivere nei dettagli ciascun aspetto del sistema che volevamo realizzare
- Si è passati quindi alla realizzazione e alla scelta delle tecnologie da utilizzare; è stato definito il linguaggio e sviluppato l’editor che consentisse di andare ad utilizzare la grammatica; è stato definito il motore in grado di interpretare la conoscenza e valutare gli esercizi svolti ed inoltre ci si è interfacciati con l’hardware necessario al riconoscimento di tali esercizi.

I problemi che la necessita di eseguire delle terapie fisioterapiche comportano sono molteplici e tra questi la necessita di doversi recare con una certa frequenza presso una struttura per essere seguiti da uno specialista che possa seguire il paziente nella fase di riabilitazione. Inoltre gli esercizi assegnati potrebbero essere svolti in modo incompleto o errato, andando a vanificare molti degli sforzi dello specialista e allungando i tempi di recupero.

La nostra soluzione consente di avere a casa del paziente uno strumento che vada a valutare come gli esercizi proposti sono effettivamente svolti, di dare al paziente un feedback sulla qualità delle proprie esecuzioni e di avere da parte del fisioterapista la possibilità di valutarne la bontà e consentendogli quindi di mettere in atto contromisure per rimediare a esecuzioni scorrette andando ad agire modificando gli esercizi.

Nel Capitolo 1 verrà presentata un'analisi degli strumenti software e hardware dei quali avremo bisogno e verrà proposto lo stato dell'arte sotto forma delle diverse alternative tecnologiche disponibili ad oggi. Nel Capitolo 2 verranno giustificate le scelte tecnologiche effettuate e verrà descritta l'implementazione del sistema come strumento per la rappresentazione di problemi di entità diversa. Infine nel Capitolo 3 verrà descritto il caso di studio specifico applicato al sistema sviluppato.

Capitolo 1

Stato dell'arte

Di seguito verranno presentate le criticità, le tipologie di strumenti necessari all'implementazione. Saranno valutate le diverse alternative tecnologiche presenti ad oggi valutandone i pro e i contro al fine di poter giustificare le scelte fatte in seguito.

1.1 Criticità

Il sistema, per come è stato concepito, necessità di una serie di strumenti software e hardware che consentano di poter gestire il flusso dei dati all'interno del sistema e garantirne la corretta interpretazione e manipolazione. Sostanzialmente abbiamo bisogno di diverse tecnologie:

- Una tecnologia che consenta di *definire il linguaggio*, ossia che permetta di specificare quali sono i mattoni della realtà che vogliamo rappresentare. Abbiamo bisogno di una tecnologia abbastanza potente da poter raggiungere la profondità espressiva necessaria che vogliamo riprodurre.
- Una tecnologia che ci consenta di *sviluppare un editor* per poter scrivere nel linguaggio definito, che permetta un'iterazione con lo strumento nel modo più semplice ed efficace possibile assistendo l'utente durante la fase di editing.

- Una tecnologia per *rappresentare lo stato*, ossia per memorizzare in una struttura organizzata i dati, al fine di mantenerne il significato e le correlazioni tra di essi.
- Abbiamo bisogno di *motore per elaborare l'informazione* fornitaci dall'utente, facendola interagire con i segnali provenienti dal mondo esterno.
- Infine è necessaria una tecnologia *hardware e software per recuperare le informazioni dal mondo esterno* secondo le specifiche del particolare caso d'uso in esame.

Quelle appena descritte sono i tipi di tecnologia dei quali abbiamo bisogno; di seguito verranno invece esplicitate le proposte valutate che sono effettivamente presenti ad oggi, che rappresentano pertanto lo stato dell'arte di tali tecnologie.

1.2 Editor e generatori di parser

Per creare un linguaggio abbiamo bisogno di uno strumento che permetta di definirne la grammatica, attraverso una sintassi BNF (Backus-Naur Form) [1] o simile e che realizzi l'infrastruttura software per riconoscere le frasi scritte nel linguaggio; questo tipo di tecnologia prende il nome di *generatore di parser*.

Le alternative a disposizione sono molte ma ne verranno presentate solo alcune, quelle di più ampio utilizzo e quelle che più si adattano agli scopi del nostro lavoro.

1.2.1 JavaCC

Il generatore di parser JavaCC [2], partendo da una specifica formale della grammatica del linguaggio, produce classi in puro codice Java con la capacità di leggere e analizzare frasi scritte con tale linguaggio. Vi è un'ampia comunità di utenti, tra i quali decine di migliaia sono utenti professionali, come testimoniato dal numero di partecipanti alle numerose mailing-list e newsgroup; sono inoltre presenti moltissimi esempi di grammatiche, accuratamente documentate.

Le specifiche sintattiche (scritte in EBNF) e quelle lessicali possono essere contenute in un unico file, consentendo una più facile lettura e manutenzione.
Sono presenti dei preprocessori come JJTree o JTB per la costruzione di alberi sintattici. Altre caratteristiche sono la presenza di un sistema di gestione avanzata degli errori che riesce a segnalare il punto nel quale si verificano e fornire una diagnosi completa del problema. JavaCC consente di generare documentazione (anche il formato HTML) in modo automatico direttamente dai file della grammatica. Essendo inoltre JavaCC scritto completamente in Java, è compatibile con praticamente qualsiasi piattaforma che preveda la possibilità di una JVM in esecuzione.

1.2.2 Gold

Gold [3] è un generatore di parser appositamente studiato per poter avere in output diversi linguaggi di programmazione.
E' strutturato infatti in due livelli, un primo livello dove la grammatica viene analizzata e un secondo livello dove è possibile utilizzare un motore diverso a seconda del tipo di linguaggio di output che si desidera utilizzare.

Al primo livello abbiamo il *Builder* che, analizzando il file della grammatica scritto con sintassi BNF, genera il file delle tabelle compilato. Questo file contiene le tabelle utili per gli algoritmi di parsing LALR e DFA che verranno implementati al livello successivo da uno degli engine. In questa fase avviene anche il controllo della grammatica per rilevare eventuali errori nella sua definizione.

Nel livello successivo è possibile andare ad agganciare uno degli *Engine* Gold, scelto in base al linguaggio di programmazione che ci è più congeniale. Il compito dell'engine è di fatto quello di implementare gli algoritmi di parsing LALR e DFA e di utilizzare il file delle tabelle compilato generato dal Builder, in modo da fornire, a fronte di frasi scritte nel linguaggio sviluppato, codice nel linguaggio di programmazione scelto.

In *Figura 1* si può vedere lo schema generale della struttura di Gold.

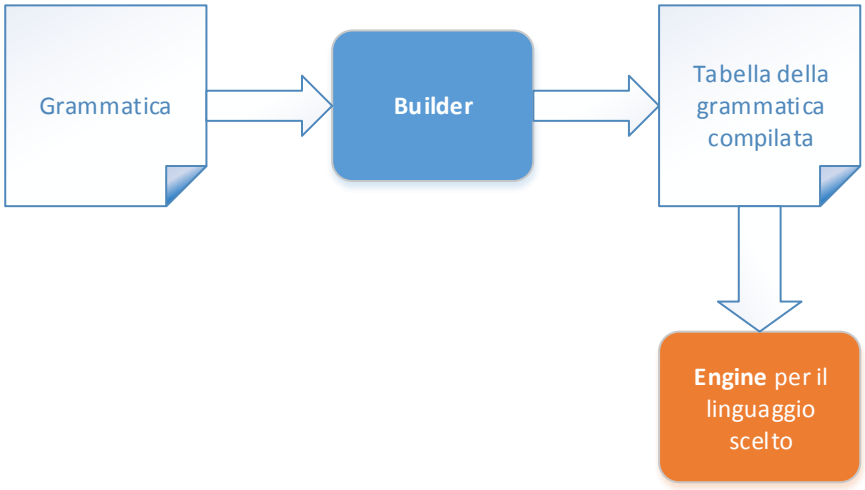


Figura 1 - Struttura generale di Gold

I linguaggi di output disponibili sono Assembly, ANSI C, C#, D, Delphi, F#, Java, Pascal, Phyton, Visual Basic 6, Visual Basic.NET, Visual C++, e per ciascun linguaggio sono presenti diversi engine. Gold è un ambiante open-source, con tutti i vantaggi quindi di avere alle spalle una comunità open. Open-source sono anche gli engine sviluppati per i diversi linguaggi. Ha lo svantaggio però di essere compatibile solo con la piattaforma Win32.

1.2.3 Antlr

Antlr (ANother Tool for Language Recognition) [4] è un potente generatore di parser per leggere, processare, eseguire o tradurre testi strutturati o file binari. Partendo da una grammatica, ANTLR genera un parser che può gestire e attraversare alberi sintattici.
La grammatica è scritta con una propria notazione, che risulta però essere molto simile a EBNF.
Come linguaggi di output l'ultima versione (Antlr4) supporta C++, Java, Python e C#. La versione 3 di Antlr prevedeva la compatibilità in output con molti più linguaggi come Ada, Javascript, Perl e Ruby.

Questo strumento consente di produrre in automatico il codice relativo per:

- Lexer: scandisce i caratteri contenuti nel file della grammatica e fornisce in uscita un flusso di token.
- Parser: contiene le regole di produzione che interpretano il linguaggio; se non vengono prodotti AST può semplicemente eseguire azioni in accordo con significato della frase. In alternativa può produrre un albero da passare ad un TreeParser.
- TreeParser: è un parser che consente di andare a gestire le strutture ad albero; accetta un AST che se richiesto viene prodotto dal parser.

Ognuno dei tre oggetti consente di andare a definire azioni semantiche. In *Figura 2* è rappresentata la struttura di Antlr dove sono evidenziati Lexter, Parser e TreeParser.

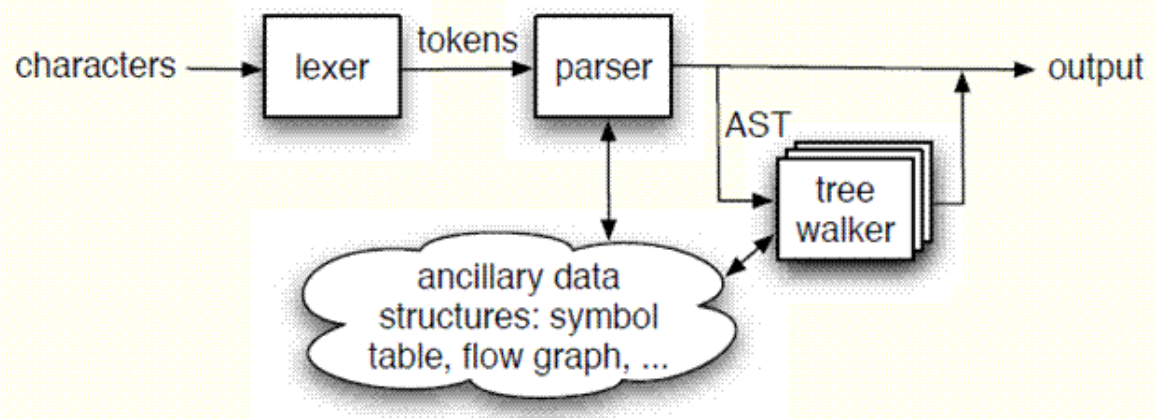


Figura 2 – Struttura generale di Antlr

Antlr è distribuito sotto licenza BSD e si appoggia su una estesa comunità open. Come JavaCC, Antlr lavora al di sopra della Java Virtual Machine, quindi può praticamente essere eseguito su qualsiasi tipo di macchina.

1.2.4 XText

Xtext [5] è un framework open-source per sviluppare linguaggi di programmazione e linguaggi specifici per un determinato dominio (DSLs - Domain-Specific Languages). A differenza degli altri generatori di parser, questo permette di generare un IDE personalizzabile per il linguaggio che è stato definito. Di seguito XText verrà esaminato focalizzando l'attenzione sulle funzionalità per la generazione del parser ma nella sezione successiva verranno analizzate le capacità di realizzare potenti editor per il linguaggio.

Xtext fa parte del progetto Eclipse ed è distribuito sotto licenza Eclipse Public.
Si appoggia ad Eclipse come piattaforma per lo sviluppo, agevolando il lavoro dello sviluppatore che opera già dentro questo ambiante. Grazie a questa integrazione l'utilizzo dello strumento può avvenire su qualsiasi piattaforma dove Eclipse può essere eseguito, quindi su macchine con sistema operativo Linux, Mac OSX e Windows.
Xtext si basa su Antlr [4] ma utilizza una propria sintassi per la definizione della grammatica del linguaggio che non risulta comunque però concettualmente diversa da EBNF. Una volta che la grammatica è stata fissata, il software genera un parser che, a fronte della somministrazioni di frasi scritte nel linguaggio definito, va a generare un Absract Syntax Tree (AST) ossia una rappresentazione secondo una struttura ad albero delle entità inserite.
L'AST può essere maneggiato con estrema facilità all'interno di Eclipse, utilizzando XTend (un linguaggio di programmazione java-like) per esplorare l'albero con l'ausilio di semplici chiamate a metodi, andando ad eseguire operazioni a seconda della tipologia di elementi incontrati. E' possibile per esempio scandire l'albero e generare automaticamente del codice personalizzato.
E' possibile integrare Test JUnit [6] per verificare la corretta generazione del codice riuscendo, attraverso l'affiancamento di output aspettato e output attuale, a semplificare questo aspetto dello sviluppo.

Grazie a queste funzionalità, Xtext risulta facile da usare e consente di produrre risultati professionali in tempi brevi.

1.2.5 Xtext per la generazione di editor

Nella sezione precedente è stato presentato lo strumento XText e sono state descritte le sue caratteristiche per quel che riguarda le funzionalità di generazione del parser. Ma c'è un'altra caratteristica importante di questo strumento, che semplifica enormemente il lavoro di sviluppo. XText infatti integra delle funzionalità per la creazione di un editor testuale che consenta di andare ad esprimere frasi nel linguaggio che viene definito. Infatti, quando viene creato un nuovo progetto XText, in realtà vengono generati più sotto-progetti: tra di essi uno è rivolto alla gestione dei file necessari per la creazione della grammatica, un altro consente di gestire la fase di test del linguaggio attraverso l'aggiunta di test JUnit e infine un progetto è rivolto a tutto quello riguarda la creazione e la personalizzazione dell'editor. Come già detto, a fronte di espressioni riconosciute correttamente nel nuovo linguaggio, viene generato un AST che può essere esplorato attraverso chiamate java-like, andando a creare nuovi documenti di output che possono quindi contenere anche codice in un linguaggio di programmazione noto.

Si noti che l'editor viene gestito come un plug-in aggiuntivo per Eclipse: all'avvio esso verrà quindi lanciato e verrà in automatico creata l'infrastruttura che consente di scrivere nuove frasi e di riconoscerle direttamente. Mano a mano che le frasi vengono scritte, l'editor genera in automatico i documenti di output previsti, fornendo all'utente in tempo reale i risultati che le frasi che sta scrivendo producono.

Essendo il plugin realizzato di fatto come una sessione di Eclipse, presenta molte delle funzionalità di supporto che sono presenti se si scrive del codice Java all'interno dell'IDE tra le quali:

- Colorazione del codice in base alla sintassi
- Auto-completamento del codice e suggerimenti per l'inserimento
- Analisi in tempo reale che evidenzia gli errori direttamente nel punto nel quale si verificano le problematiche
- Visione dell'outline, ossia della struttura delle frasi che si stanno scrivendo

- Indicizzazione degli elementi, con conseguente possibilità di modifica di un nome di un elemento una volta sola senza modificarlo ogni volta in cui compare

Di base XText fornisce tutte queste funzionalità ma l'editor può anche essere ulteriormente personalizzato andando a specificare come dovranno essere l'aspetto o altre funzionalità: per esempio è possibile andare a definire elementi grafici come finestre, pulsanti, etichette, associando ad esse delle funzioni.

Sono presenti altri strumenti che consentono di andare a specificare una grammatica, generare il parser e l'editor, come ad esempio applicazioni per l'IDE NetBeans [7] o il plug-in JetBrains MPS [8] per l'IDE IntelliJ IDEA, ma XText dalla sua possiede un'immediatezza superiore e una stretta integrazione con Eclipse che, essendo uno degli ambienti di sviluppo più utilizzati, risulta essere un punto di forza importante.

1.3 Motore

Senza avere un motore che elabori tutti i dati ricavati dalla fase di riconoscimento dell'attività umana non è possibile ottenere nuova conoscenza; abbiamo quindi bisogno di una tecnologia che riesca a desumere nuovi concetti utili "ragionando" sui dati a disposizione. Queste tecnologie ricadono nell'insieme dei *sistemi esperti*.

L'approccio classico ai problemi di intelligenza artificiale è di affrontare tutte le strade possibili e raggiungere dinamicamente la soluzione. A differenza di un algoritmo, dove la conoscenza è direttamente cablata nel procedimento per la soluzione, in un sistema esperto la conoscenza cambia dinamicamente e può espandersi nel tempo. Vengono tenute ben distinte le conoscenze dai procedimenti: attraverso quest'ultimi infatti la conoscenza può essere modificata. In un sistema esperto saranno presenti quindi due elementi principali:

- Una base di conoscenza che consiste nei fatti, nel sapere necessario ad affrontare un problema

- Un motore inferenziale che fa riferimento ai meccanismi con i quali si riescono a dedurre nuove informazioni partendo dalla base di conoscenza

Un classico esempio d'uso dei sistemi esperti è l'utilizzo in campo medico; in questo settore le tecnologie in questiono possono funzionare a supporto dello specialista, andando ad esempio, in base ai sintomi che il paziente riscontra, a dedurre una serie di patologie possibili. Altre applicazioni, sempre in ambito medico, possono essere l'utilizzo per migliorare la qualità del processo di validazione eseguito dai laboratori di analisi biochimica (DNSEV - Expert System for clinical result Validation) [9], per monitorare gli eventi infettivi all'interno dell'ospedale (ESMIS - Expert System for Microbiological Infection Surveillance) [10], o per il supporto ai medici per le prescrizioni e le visite per la Terapia Anticoagulante Orale (DNTAO - Expert System for supporting the Oral Anticoagulation Treatment) [11].

Le prime applicazioni dei sistemi esperti si hanno in ambiente universitario dei primi anni 60. E' del 1961 in progetto MACSYAMA, che nonostante cerchi di affrontare questi tipi di problematiche in modo nuovo si basa ancora su algoritmi tradizionali. DENDRAL [12] è una procedura euristica destinata al settore dell'industria chimica che viene sviluppata verso la metà del decennio. Dopo questo progetto i successivi studi si concentrarono soprattutto sulle applicazioni in campo medico; tra di esse ebbe risalto MYCIN, iniziato come un progetto all'Università di Stanford nel 1972.
EMYCIN nel 1979 fu sviluppato come uno strumento che mantenesse una certa generalità e che quindi consentisse di affrontare problemi di natura diversa: le applicazioni più sviluppate continuarono a essere comunque indirizzare al campo medico come PUFF per la diagnosi delle malattie polmonari e ONCOCIN per il cancro, anche se non mancarono sperimentazioni come DART in ambito industriale per la diagnosi dei guasti.
CASNET, sviluppato sempre negli anni 70', era orientato al risolvere problemi in campo medico ma lo strumento EXPERT al suo interno riusciva ad adattarsi a trattare problemi anche in ambiti diversi.
Di seguito vennero sviluppati sistemi esperti per la ricerca petrolifera (PROSPECTOR – 1979), per definire le configurazioni hardware dei computer di allora (XCON - 1980), e PLANT realizzato nel 1981 e che poteva diagnosticare malattie nelle coltivazioni di soia.

Come è possibile però realizzare un sistema esperto? Come organizzare le conoscenze a disposizione? Com'è costituito il motore di inferenza? Attraverso modelli che possiamo ricondurre al comportamento umano, nel 1972 Newell e Simon definirono il concetto di sistema di produzione, come sistemi che rappresentano la conoscenza con un sistema a regole.
In un applicazione di questo tipo è presente *(Figura 3*):

- Una *base di conoscenza*, dove sono contenute le regole di produzione.
- Una *memoria di lavoro* che contiene i dati in ogni istante dell'elaborazione, ovvero i fatti che il motore genera e che sono valutati nelle regole.
- Un *motore inferenziale* che si occupa di verificare le regole che possono essere eseguite e di generare nuovi fatti.

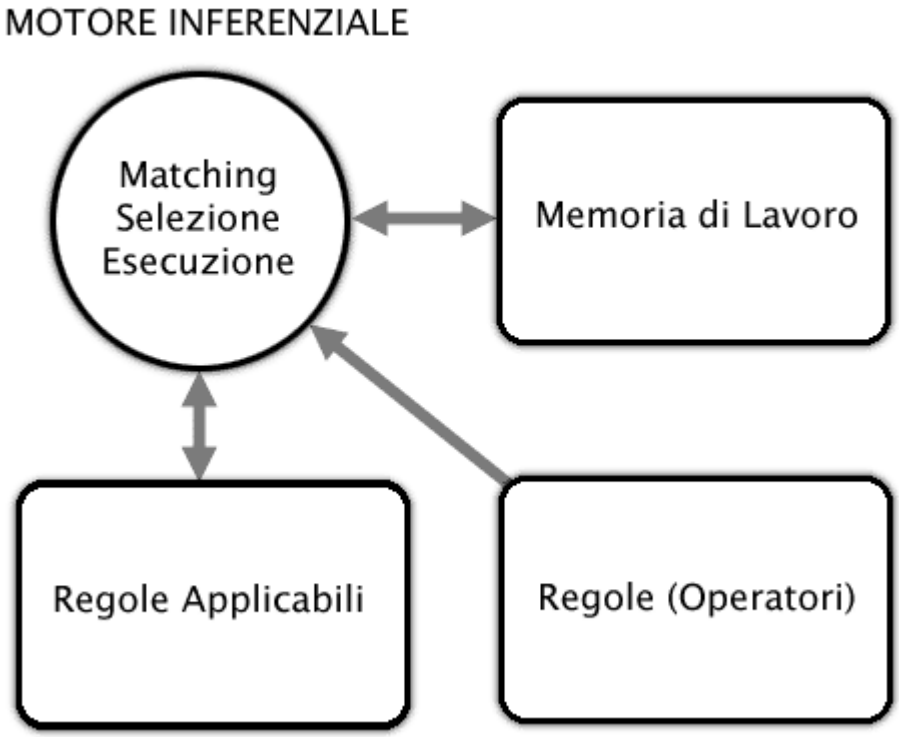


Figura 3 – Sistema a regole

Le regole sono organizzate secondo questa struttura:

```
WHEN <pattern> THEN <corpo>
```

Il motore inferenziale va a valutare, tra tutte le regole presenti nella base di conoscenza, grazie ad un meccanismo di pattern-matching, quali hanno un pattern verificato, ossia quali hanno la condizione (determinata dai fatti contenuti nella memoria di lavoro) verificata. Ogni regola il cui pattern è verificato produrrà delle conseguenze che generalmente consistono nella modifica dell'insieme dei fatti contenuti nella memoria di lavoro.

Esistono due metodologie con le quali è possibile concatenare le regole:

- Forward chaining: in questa tecnica i fatti contenuti nella memoria di lavoro vengono usati per far scattare le regole che fanno match con quei fatti. Uno schema di questo tipo di metodologia è rappresentato in *Figura 4.*
- Backward chaining: in questo caso invece si parte dal goal, ossia dal risultato che si vuole dimostrare, attivando quelle regole le cui azioni comprendono l'obiettivo; si prosegue in questo modo a ritroso fino a che nella memoria di lavoro abbiamo l'insieme dei fatti noti.
- Bidirezionale dove forward e backward chaining vengono utilizza insieme per ottenere soluzioni in modo più rapido

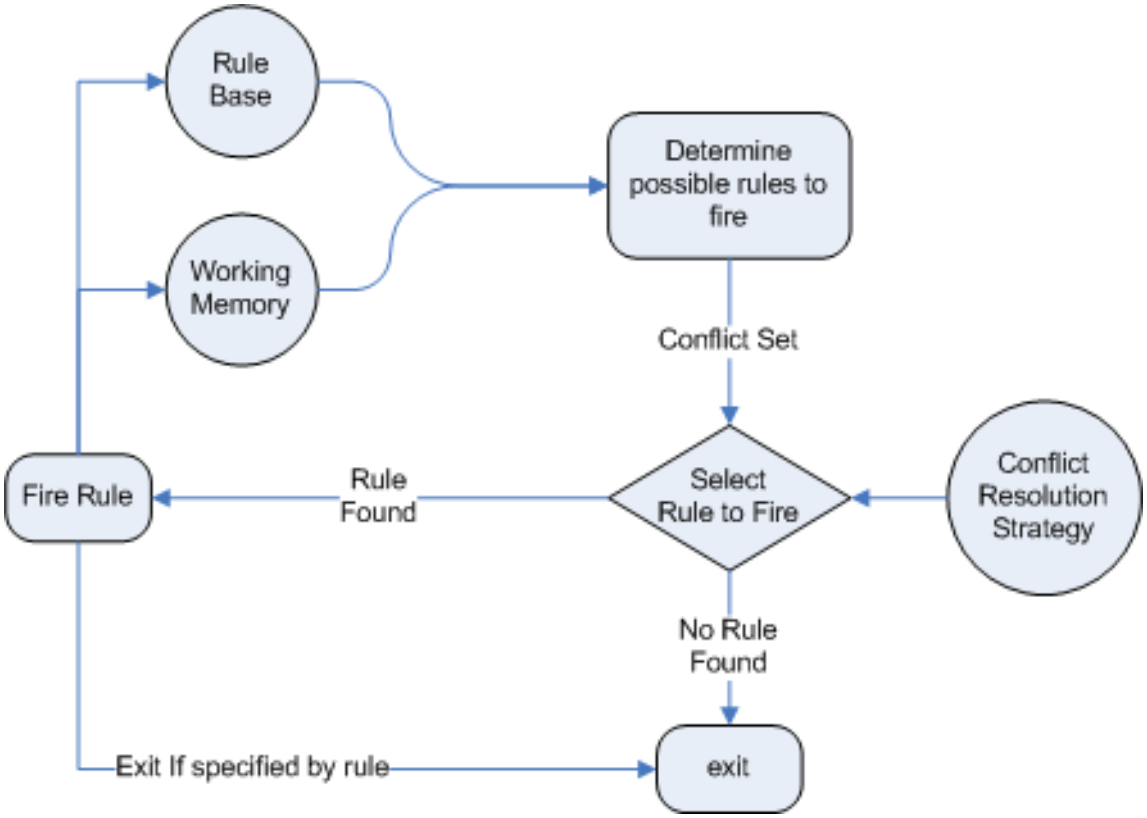


Figura 4 – Metodologia forward chaining per un sistema a regole

Le implementazioni dei sistemi a regole sono diverse; CLIPS (C Language Integrated Production System), sviluppato a partire dal 1985 da un progetto dell'agenzia spaziale NASA, che include un linguaggio object-oriented per la creazione di sistemi esperti, è scritto in C e nello stesso linguaggio sono scritte anche le sue estensioni e può essere utilizzato attraverso chiamate provenienti da applicazioni scritte in C. Si basa sull'algoritmo Rete ed è uno dei primi sistemi a regole con funzionalità avanzate.

Da CLIPS sono nati altri progetti simili come Jess nel 1995, che ne è un'estensione ed è sviluppato in linguaggio Java, e FuzzyCLIPS che aggiunge il concetto di rilevanza al linguaggio.
Altri sistemi a regole sono Microsoft BRE, scritto in C#, e Drools che è scritto in Java e si integra in modo naturale con tale linguaggio; tra le sue peculiarità possiede quella di poter gestire eventi e le loro proprietà temporali.
E' stato effettuato un confronto [13] tra Jess, Drools e Microsoft BRE

per evidenziare le loro performance sottoponendoli ad una serie di test. Viene evidenziato come i tre sistemi hanno performance simili e un aumento lineare del tempo di esecuzione all'aumentare del numero di regole nella base di conoscenza.

1.3.1 Drools

Dalla home page del progetto si può leggere:

"Drools 5 introduce una piattaforma per l'integrazione della logica di business che fornisce una piattaforma unificata per regole, flussi di lavoro e trattamento degli eventi"

Drools [14] è un sistema a regole che si basa su un'implementazione dell'algoritmo Rete [15]. Utilizza la metodologia forward chaining per estrarre nuovi fatti partendo da una serie di conoscenze. Il progetto è realizzato interamente in Java ed è Open Source. Prevede un linguaggio proprietario per la scrittura delle regole ma la sintassi ne facilita la comprensione da parte degli esperti del settore.

E' presente una profonda integrazione con l'IDE Eclipse. Come gli altri sistemi a regole, Drools spiega *cosa* dev'essere fatto e non *come* quel risultato dev'essere raggiunto; c'è una profonda separazione tra la logica, espressa attraverso le regole, e i dati (i fatti).

Nella Production Memory (PM) si trovano tutte le regole che descrivono la logica di business dell'applicazione, mentre nella Working Memory (WM) sono presenti i fatti che si riferiscono ad una

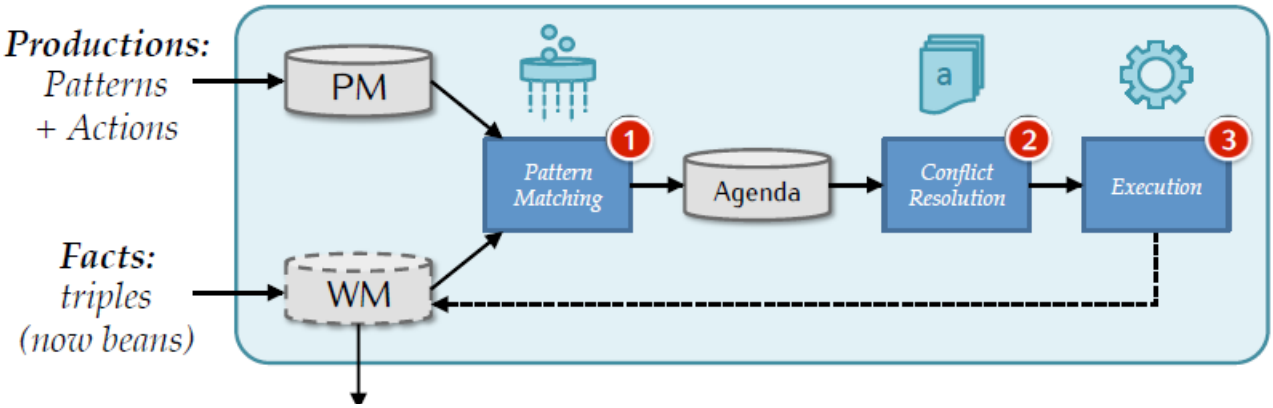


Figura 5 – Componenti Drools

particolare istanza, al problema attuale da risolvere.
Attraverso un meccanismo di pattern-matching, tra tutte le regole vengono selezionate quelle che devono essere eseguite, controllando che il pattern della condizione di attivazione contenga fatti che sono presenti in quel momento nella WM.
A questo punto l'insieme delle regole estratte vengono inserite nell'Agenda. Sarà ora compito di uno stadio di risoluzione dei conflitti andare a definire il corretto ordine di esecuzione delle regole, andando anche a prendere in considerazione la priorità che può essere espressa staticamente dall'utente per le regole.
Una volta che l'ordine è stato determinato è possibile passare alla fase di esecuzione dove le regole vengono azionate una dopo l'altra andando a ritrattare o modificare fatti nella WM.

In *Figura 5* si possono distinguere i componenti grazie ai quali Drools opera.
Essendo Drools legato profondamente al mondo Java, è possibile, tra le azioni che dovrà eseguire se una regola viene attivata, definire anche praticamente qualsiasi blocco di codice Java.

Drools lavora quindi con tre memorie diverse e in tre fasi di elaborazione in catena: oltre alla WM e PM è infatti necessaria l'Agenda per andare a memorizzare le regole da eseguire ma non ancora ordinate che potenzialmente potrebbero presentare dei conflitti; per quando riguarda le fasi di elaborazione, dopo un primo stadio dove con il meccanismo del pattern-matching vengono selezionate le regole da azionare, è necessario avere una fase dove avvenga la risoluzione dei conflitti con il riordino delle regole e una fase di esecuzione che andrà ad aggiornare la WM. In *Figura 6* sono evidenziate le memorie che Drools utilizza nel suo processo di esecuzione.

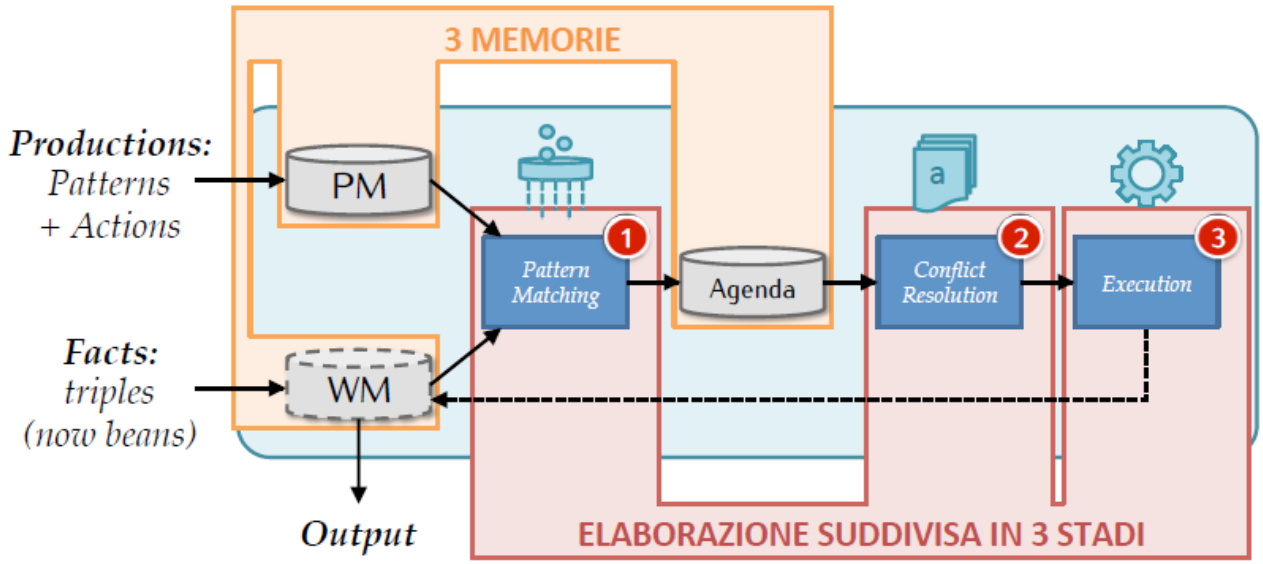


Figura 6 - Memorie di Drools

A Drools afferiscono 5 progetti che lavorando insieme riescono a creare un sistema potente e integrato:

- Drools Guvnor: è il repository centralizzato per la base di conoscenza; fornisce strumenti per la gestione di un gran numero di regole, gestendo un controllo degli accessi efficace; è infatti possibile bloccare o limitare l'accesso alle regole in base al tipo di privilegi posseduti. Guvnor è quindi utile quando vogliamo gestire un sistema di controllo degli accessi alle regole, quando vogliamo fornire la possibilità (attraverso interfacce grafiche, ad utenti con privilegi differenti) di modificare la base di conoscenza e se le regole che dobbiamo gestire sono numerose ed è necessario avere un meccanismo di versioning delle stesse.
- Drools Expert: è il cuore del sistema; implementa l'algoritmo Rete, effettua le operazioni di pattern-matching, risolve i conflitti ed esegue le regole, realizza dunque il motore di Drools vero e proprio.
- jBMP 5: è uno strumento per la gestione dei processi di business, fa da ponte tra l'analista dei processi e lo sviluppatore; è quindi

rivolto ad un utenza non tecnica e riesce a offrire una gestione delle funzionalità che l'analista può utilizzare e che piace allo sviluppatore.

- Drools Fusion: introduce in concetto di evento, ovvero di fatto che si verifica in un certo istante e che ha quindi una caratteristica di tipo transitorio. E' possibile associare agli eventi un valore puntuale oppure un intervallo temporale. Grazie a queste funzionalità è ad esempio possibile nelle regole, all'interno del pattern, andare a definire vincoli di tipo temporale tra i diversi eventi come condizione all'avvio di una regola.
- OptaPlanner: è uno strumento per ottimizzare il meccanismo di soluzione di problemi di ottimizzazione in situazioni dove è necessario fornire prodotti o servizi con una serie di vincoli per quanto riguarda le risorse a disposizione.

1.4 Calcolo degli Eventi

Il calcolo degli eventi (Event Calculus – EC) è un formalismo che consente di ragionare ed esprimere concetti riguardanti le azioni ed i loro effetti su un sistema.

Formalizzato per primo da Kowalsky e Sergot nel 1986 [16](**BEC** – Basic Event Calculus) ha ormai quasi 30 anni e viene utilizzato negli ambiti più disparati, dai ragionamenti sulle leggi e loro retroattività [17] alla robotica cognitiva [18], dalla gestione dei processi di business [19] alle linee guida in ambito clinico [20], dalla modellazione dei flussi di lavoro [21] a problemi di pianificazione [22] e altro ancora.

EC si basa si concetti fondamentali di Evento e Fluente; un Evento si riferisce sempre ad un istante di tempo e in genere provoca un cambiamento dello stato del sistema; il Fluente invece rappresenta una variabile del sistema stesso, che può assumere diversi valori e l'insieme dei fluenti in un dato istante può essere concepito come lo stato del sistema stesso. Ecco quindi che quando viene notificato un evento lo stato del sistema varia.

Il principio di funzionamento del primo formalismo di EC si basa su una serie di assiomi, ossia una serie di proposizioni, di frasi, che

costituiscono le fondamenta della formulazione, e che permettono di costruirvi sopra formulazioni differenti o che espandono quella primitiva.
Il meccanismo per il calcolo degli eventi comprende dei parametri che, a seconda degli obiettivi che ci prefiggiamo, risulteranno essere parametri di input piuttosto che parametri di output.
Nel calcolo degli eventi abbiamo "COSA SUCCEDE QUANDO", ossia la successione temporale degli eventi che vengono notificati al sistema, "COSA FANNO GLI EVENTI", che dà l'informazione sugli effetti degli eventi stessi, ossia su quali eventi vanno ad agire e in che modo e "QUANTO VALE QUANDO", che rappresenta lo stato del sistema in ogni istante temporale.

Se sappiamo cosa fanno gli eventi e quando essi avvengono allora possiamo voler predire in un dato istante quale sarà lo stato del sistema, ossia il valore dei fluenti; usiamo in questo caso un ragionamento *deduttivo* ed è utilizzato in genere quando si vogliono effettuare operazioni di monitoraggio.

Quando invece siamo a conoscenza dello stato del sistema, sappiamo cosa fanno i vari eventi, ma non conosciamo quando essi si verificano allora possiamo utilizzare un ragionamento *abduttivo* per appunto ricavare la sequenza temporale degli eventi; si utilizza quando vogliamo raggiungere un risultato voluto, e vogliamo ricavare i passi che devo rare per ottenerlo.

Abbiamo invece un ragionamento *induttivo* quando, come nel caso precedente, siamo a conoscenza dello stato del sistema nel tempo, della sequenza temporale degli eventi, ma non sappiamo in pratica che effetti i vari eventi producano; possiamo utilizzare questo tipo di ragionamento quando, avendo in mano dei dati osservati e una sequenza di eventi, vogliamo ottenere le leggi che regolano un dato ambiente, la teoria sulla quale si basa il funzionamento del sistema.

Possiamo quindi dire che EC è uno strumento che si può applicare a differenti applicazioni e tipologie di utilizzo, e, passando da una all'altra, gli obiettivi cambiano e con loro i parametri di input e di output del meccanismo di ragionamento.

Il formalismo introdotto da Kowalsky e Sergot non fu l'unica formulazione del calcolo degli eventi: Kowalsky stesso ed altri dopo di lui introdussero modifiche ed estensioni del modello di base, in modo da potenziarlo ed adattarlo a casistiche di utilizzo specifiche.

Una prima variazione rispetto al risultato raggiunto nel 1986 fu introdotta appunto da Kowalsky nel 1992 e viene chiamata **SEC** – Simple Eventi Calculus [23]; con SEC si passa da una formulazione sostanzialmente ad intervalli, ad una basata sugli istanti temporali. Formalizzato da Miller e Shanahan alla fine degli anni 90', **FEC** – Full Eventi Calculus [24] - è invece un'estensione del primo formalismo che introduce assiomi duali a quelli primitivi e ne introduce di completamente nuovi. Gli stessi autori introdussero **EEC** [24] che è contemporanea a FEC e ne è un estensione; aggiunge infatti degli assiomi che gestiscono eventi notificati concorrentemente.

Il problema di tutti i formalismi trattati fino ad ora è che all'arrivo di un nuovo evento, per derivare lo stato del sistema è necessario andare a rivalutare l'intera storia degli eventi notificati, con l'aggiunta del nuovo arrivato. Questo meccanismo produce overhead importanti, soprattutto quando il numero di eventi diventa elevato. Pur rimanendo opzioni robuste e funzionanti, BEC e le sue prime modifiche risentivano quindi di questo tipo di comportamento, e varianti successive cercarono di rimediare a tale problema. Queste variazioni si basano sul concetto che, all'arrivo di un evento, esso provoca una (limitata) variazione dello stato del sistema; non sarà più quindi necessario rivalutare tutti gli eventi da primo ma basterà aggiornare il valore dei fluenti che il nuovo evento avrà modificato.

CEC – Cached Event Calculus – [25] introduce il concetto di MVI (massimo intervallo di validità) e consente di specificare l'intervallo di tempo per il quale un fluente assume un determinato valore. Attraverso questa struttura dati si riesce di fatto a mantenere uno storico che mantiene una cache della storia del sistema. **REC** – Reactive Event Calculus – [26] utilizza anch'esso gli MVI, è si pone come un' alternativa più efficiente a CEC; nel caso in cui però un evento fosse notificato con ritardo, non mantenendo in cache lo stato in ogni istante ma solo la successione temporale degli eventi, sarà necessario effettuare una procedura di backtracking per eliminare le modifiche effettuate

dagli eventi arrivati prima di quello notificato con ritardo, andando a riapplicare gli effetti degli eventi nell'ordine cronologico corretto.

Essendo il backtracking un'operazione costosa, REC sarà da preferire in applicazioni nelle quali gli eventi vengono notificati senza ritardo, o comunque con delay ridotti tali da non compromettere l'ordinamento corretto degli stessi. Se invece i delay diventano non trascurabili, gli eventi potrebbero essere potenzialmente disordinati; in questi casi quindi viene preferito CEC che non richiede costosi backtracking in caso di notifiche con ritardo.

1.5 Riconoscimento dell'attività umana

Quello di cui sostanzialmente abbiamo bisogno è una tecnologia che ci consenta di andare a raccogliere dati che ci forniscano la posizione delle parti del corpo di un soggetto, in modo sufficientemente preciso da poter essere elaborata, e da estrarne le informazioni sulle pose assunte.

Ad oggi sono potenzialmente diverse le tecnologie che consentono il rilevamento dell'attività umana:

- Sensori di profondità: si tratta di sensore a infrarossi, ossia dispositivi che emettono raggi infrarossi e che possiedono una telecamera sensibile alla stessa banda. Queste soluzioni ha la caratteristica di non dipendere dall'illuminazione o da altre condizioni dell'ambiente dove avviene il riconoscimento ed inoltre riescono a fornire informazioni spaziali in tre dimensioni. Leader in questa tecnologia è l'azienda israeliana PrimeSense [27] che, oltre a proporre la propria gamma di sensori *Xtion*, ha progettato anche il più conosciuto Kinect, sensore per la console Xbox di Microsoft.
- Sensori basati sull'utilizzo di giroscopi e accelerometri: permettono anch'essi di ottenere informazioni spaziali. L'accelerometro individua le accelerazioni applicate lungo tre assi di riferimento che possono essere utilizzate per individuare dei movimenti. Il giroscopio invece misura la rotazione lungo gli assi di riferimento. Andando a combinare i dati provenienti dai

due sensori è possibile ottenere informazioni più accurate che però si limitano a rappresentare solamente un modo di tipo bi-dimensionale. Un esempio di questi tipi di dispositivi sono il WiiRemote [28], il controller per la console Wii di casa Nintendo.

- Telecamere con supporto al video-tracking, ossia dotate di algoritmi per riconoscere movimenti o gesture. Un esempio di tale tecnologia sono PlayStation Eye e PlayStation Move [29] di Sony. Il primo dispositivo è una telecamera che consente di riconoscere pose e movimenti del soggetto verso la quale è puntata; il secondo invece consiste in un controller con una sfera luminosa che consente di essere rilevata da Eye ed inoltre possiede una serie di sensori per ricavare ulteriori informazioni su movimenti e sulle pose.
- Tecnologie motion capture: sono soluzioni dove il soggetto indossa dei dispositivi (marker) in punti specifici del corpo, solitamente in corrispondenza delle giunture. Sono presenti diverse tecnologie (ottiche, inerziali, meccaniche o magnetiche) ma tutte hanno la caratteristica di misurare nel tempo la posizione dei marker che, essendo posizionati secondo una struttura precisa permettono di rappresentare uno scheletro e le posizioni che assume. Maggiore è il numero di marker e la loro densità, migliori e più precise saranno le informazioni ricavate. Di contro, rispetto ad altre soluzioni, sono generalmente più costose perché richiedono hardware e software specifico particolarmente avanzato e necessitano di spazi e condizioni di acquisizione con caratteristiche particolari.

C’è inoltre bisogno di uno strato software che ci consenta di riconoscere le pose rilevate dai vari sensori, una tecnologia che permetta di distinguere e quindi classificare i diversi tipi di movimenti e pose assunte.

1.5.1 Microsoft Kinect

Tra le diverse alternative che vi sono relativamente ai sensori di prossimità, focalizziamo la nostra attenzione su Kinect.
Kinect è un accessorio per la consolle Xbox di Microsoft; consente di interagire con il videogioco senza l'utilizzo di controller o altre periferiche ma semplicemente con il proprio corpo.
Kinect nasce con il nome in codice ProjectNatal nel 2009 e viene presentato ufficialmente all'E3 dell'anno successivo. L'hardware del dispositivo è sviluppato dall'azienda israeliana PrimeSense [27], leader nel riconoscimento dei movimenti tramite videocamere. Il software è invece realizzato internamente da Microsoft dalla società Microsoft Game Studios. Da febbraio 2012 sono stati rilasciati i driver del dispositivo per piattaforma Windows.

Nonostante sia rivolto principalmente ad un pubblico di videogiocatori, presto Kinect ha suscitato l'interesse di sviluppatori e personale esterno al mondo videoludico che hanno utilizzato lo strumento per progetti relativi al riconoscimento dell'attività umana, complici anche il prezzo relativamente basso della periferica e la possibilità di accedere alle sue funzionalità tramite le API rese disponibili; citiamo per esempio applicazioni per rilevare le espressioni del volto o ricreare avatar 3D personalizzati [30].

Kinect possiede una base sulla quale il dispositivo può ruotare grazie ad un perno che gli consente di ruotare lungo l'asse verticale per seguire i movimenti del soggetto. Possiede una telecamera RGB, un sensore di profondità realizzato con un emettitore di raggi infrarossi e una telecamera sensibile alla stessa banda, un altoparlante e un microfono dotato di un filtro per la soppressione del rumore ambientale. Grazie al dispositivo è possibile quindi interagire con Xbox anche attraverso comandi vocali. Il tipo di sensore consente rilevare correttamente i soggetti anche il condizioni di luce scarsa o illuminazione assente.

Figura 7 - Microsoft Kinect

Caratteristiche tecniche

La telecamera RGB ha una risoluzione di 640 x 480 pixel, mentre quella sensibile ai raggi infrarossi ha risoluzione minore, pari a 320 x 240 pixel.
Dovendo il dispositivo elaborare autonomamente le immagini, non delegando il lavoro alla consolle, è necessaria una piccola ventola che aiuti a dissipare il calore prodotto dai componenti interni.
Per alimentare la periferica Kinect richiede 12 watt, quindi originariamente non era possibile collegarlo direttamente alla Xbox tramite il cavo usb che riesce a fornire in media soltanto 2,5 watt, ed era quindi necessario un alimentatore esterno. Nell versioni più aggiornate della consolle Kinect vi può essere collegato direttamente tramite una porta apposita che fornisce la potenza necessaria.
Il campo visivo di Kinect è di 50° in orizzontale e 43° in verticale; il raggio visivo verticale può essere ulteriormente ampliato in quanto un motore riesce a inclinare lungo quest'asse le telecamere di 27°.
L'intervallo di funzionamento dichiarato va 0.8 m a 3.5 m ma si

riescono ancora ad ottenere rilevazioni soddisfacenti anche fino a 6 metri.
In Tabella 1 vengono riassunte le caratteristiche principali di Microsoft Kinect.

Tabella 1 – Caratteristiche tecniche Kinect

Campo Visivo (orizzontale-verticale-diagonale)	58°O – 43°V – 70°D
Risoluzione x/y (a 2 m dal sensore)	3 mm
Risoluzione z (a 2 m dal sensore)	1 cm
Range di lavoro	0.8 m - 3.5 m
Interfaccia	USB 2.0
Consumo di potenza	12 watt
Immagine di profondità	320 x 240 pixel
Immagine a colori RGB	640 x 480 pixel
Frame-rate	30 fps
Temperature di lavoro	0° C – 40° C

OpenNI e PrimeSense NITE

Se volgiamo interagire con Kinect avremo bisogno dei driver necessari. Le libreria di supporto per l'interfacciamento con l'hardware e il recupero dei dati sono diverse: vi sono soluzioni open-source come PCL (Point Cloud Library) [31] o Libfreenect [32], oppure ci si può affidare al Kinect SDK [33] rilasciato direttamente da Microsoft che ha il vantaggio di poter riconoscere un maggior numero di giunture per lo scheletro, di poter manovrare il motore per inclinare verticalmente il dispositivo e poter utilizzare l'interfaccia audio; di contro però il tipo di licenza con la quale è distribuito porta con se grosse limitazioni. Di seguito andremo però ad analizzare i driver il cui utilizzo è più diffuso: il framework OpenNI e il middleware PrimeSense NITE.

OpenNI (Open Natural Interaction) [34] è un framework che consente di interfacciarsi con l'hardware di un sensore e fornire una serie di API per l'utilizzo ai livelli superiori dei dati ricavati.
Il codice sorgente di OpenNI è liberamente distribuito e consultabile; ecco quindi che quando Kinect è stato commercializzato le librerie sono state implementate per quella specifica architettura. OpenNI infatti non è progettato con l'intento di funzionare solo con Microsoft Kinect ma è potenzialmente compatibile con qualsiasi dispositivo implementi le specifiche della libreria. OpenNI fornisce ai livelli superiori dati relativamente grezzi che devono essere poi elaborati dai middleware per poterne ricavare informazioni utilizzabili dalle applicazioni.

Il middleware PrimeSense NITE (Natural Interaction Technology for End-user) [35] utilizza la libreria OpenNI e trasforma i dati che il software fornisce in informazioni sull'ambiente più comprensibili e meglio utilizzabili per l'uomo e per le applicazioni superiori.
In particolare esso può agire attraverso degli algoritmi in diversi contesti tra i quali:

- Rilevazione e tracciamento delle mani
- Separazione della sagoma del corpo dallo sfondo
- Generazione dello scheletro utilizzando le giunture del corpo umano

In *Figura 8* viene mostrato uno screenshot che rappresenta la sagoma del corpo del soggetto riconosciuto e lo scheletro che la libreria PrimeSense NITE consente di costruire.

Figura 8 - Ricostruzione scheletro del soggetto

I Controlli NITE forniscono un ulteriore strato di separazione andando a fornire funzionalità per le applicazioni al livello superiore come il tracciamento e la rilevazione di gesture delle mani.
Gli applicativi possono quindi comunicare direttamente con il software OpenNI se necessitano di dati più grezzi mentre se hanno bisogno di informazioni più significative come lo scheletro dei soggetti si potranno servire del middleware NITE per ottenere ulteriori funzionalità.

In *Figura 9* viene mostrato come le librerie OpenNI e NITE operano e come si interfacciano tra di loro, con i sensori e le applicazioni a livello più alto. In *Figura 10* invece si vuole rappresentare il flusso di dati che dalla rilevazione delle immagini da parte dell'hardware porta alla ricostruzione di sagome virtuali che presentano informazioni di profondità.

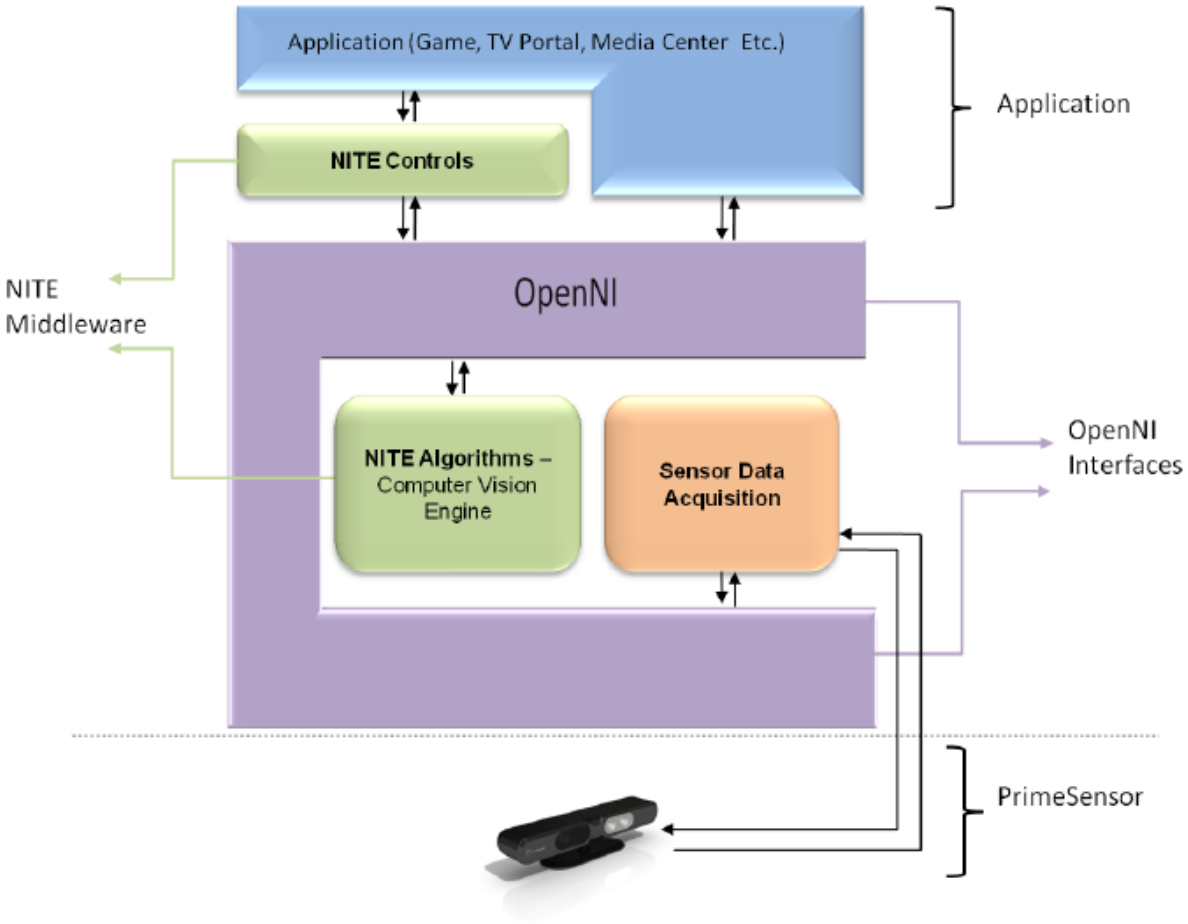


Figura 9 - Librerie per l'interfacciamento con il sensore

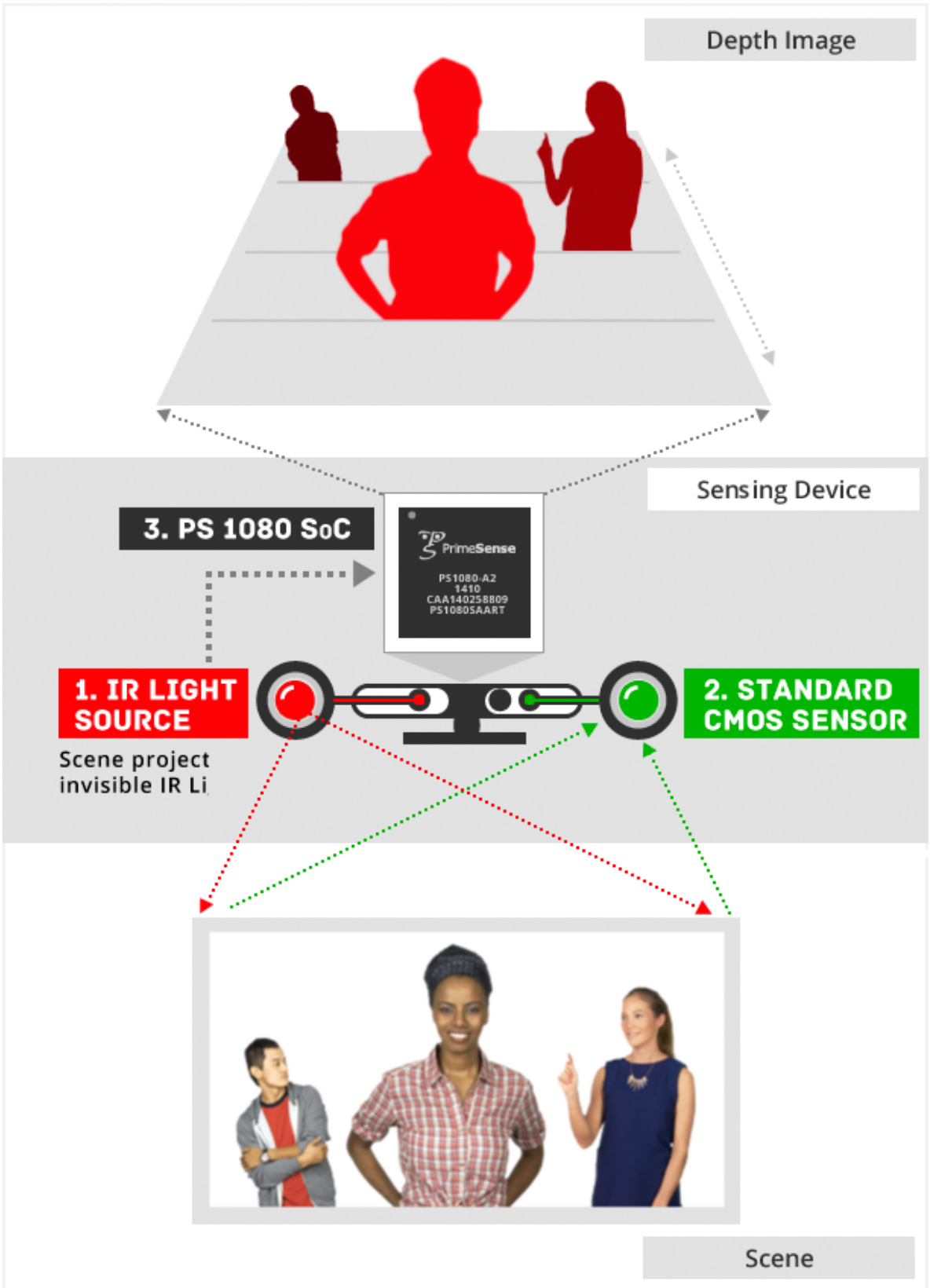


Figura 10 - Ricostruzione soggetti virtuali partendo dal sensore

1.5.2 Il problema della classificazione

Un altro aspetto da prendere in considerazione quando si tratta il problema del riconoscimento dell'attività umana è la tecnologia che può essere utilizzata per operare la classificazione di istanze in classi distinte. Con dispositivi hardware e uno strato software intermedio si riescono a recuperare dati anche di alto livello che, forniti ad un agente umano, riescono a fargli dedurre delle considerazioni significative. E' necessario però cercare di automatizzare questa fase, utilizzando una tecnologia in grado di effettuare valutazioni relativamente complesse. In particolare ci focalizziamo sul problema della *classificazione*.

In modo analogo a quanto avviene con il cervello umano, l'idea è di realizzare delle macchine ad Apprendimento Automatico (AA) che riescano, in seguito ad una opportuna fase di learning, a distinguere delle istanze che gli verranno sottoposte, andandole a catalogarle nelle classi delle quali ha imparato le caratteristiche.

La fase di insegnamento può essere di tre tipologie:

- Apprendimento supervisionato
- Apprendimento non supervisionato
- Apprendimento con rinforzo

In particolare l'*apprendimento supervisionato* consente di fornire dell'esperienza al sistema attraverso una serie di istanze, già opportunamente classificate. Maneggiando queste entità la macchina riesce a reagire ad ulteriori somministrazioni di problemi, non ancora classificati, andando ad attribuirvi la classe di appartenenza. La fase di somministrazione delle istanze nella fase di learning deve essere effettuata con cura. Bisogna infatti insegnare alla macchina a valutare in modo corretto le istanze, evitando sia casi di sovrallenamento che di sotto-allenamento: se infatti forniamo un allenamento troppo dettagliato potremmo riscontrare comportamenti non voluti perché la rete non riconosce come appartenenti ad una data classe istanze che si discostano anche leggermente dall'istanza obiettivo ideale che è stata assimilata; di contro, se l'allenamento fosse troppo blando, in fase di classificazione la macchina potrebbe entrare in confusione non riuscendo più a distinguere in maniera netta una classe dall'altra.

WEKA [36] è una libreria contenente una serie di algoritmi per l'AA che è stata sviluppata presso l'Università of Waikato in Nuova Zelanda. WEKA è open source e può essere richiamata facilmente all'interno di codice Java. Gli algoritmi per l'apprendimento supervisionato che la libreria implementa sono diversi, tra i quali citiamo il Percettrone MultiStrato, la Funzione su Base Radiale, gli Alberi Decisionali e gli Alberi su Modello Logistico; di seguito verrà però presentato l'algoritmo basato sulle Macchine a Supporto di Vettori che risultata essere lo strumento più adatto per risolvere problemi di classificazioni relativamente al nostro caso di utilizzo [37].

Macchine a Supporto di Vettori

La teoria delle Macchine a Supporto di Vettori (SVM) viene sviluppata negli anni 90' da Vladimir Vapnik presso i lavoratori Bell AT&T. Le SVM possono essere utilizzate in svariati ambiti tra i quali la classificazione dei testi e di oggetti, il riconoscimento di volti in fotografie, in ambito OCR (riconoscimento ottico dei caratteri) per estrarre contenuto testuale da dei documenti cartacei. Come tutti gli algoritmi ad apprendimento supervisionato, riceve inizialmente una serie di esempi grazie ai quale riesce ad imparare quali saranno le categorie che dovrà essere in grado di distinguere; gli esempi forniti possono essere mappati in uno spazio multi-dimensionale; se per semplicità ci limitiamo ad analizzare un classificatore binario, lo spazio può essere diviso in due parti attraverso un iperpiano che separa gli elementi appartenenti alle due tipologie.

Esistono due gamme di dati (*Figura 11*), quelli separabili linearmente e quelli non-separabili linearmente. Avendo posizionato i dati nello spazio multidimensionale possiamo infatti trovarci nella situazione in cui sia possibile attraverso un iperpiano effettuare facilmente la separazione e potrebbe invece succedere che i dati abbiano una natura tale che li porti ad essere non-separabili linearmente.

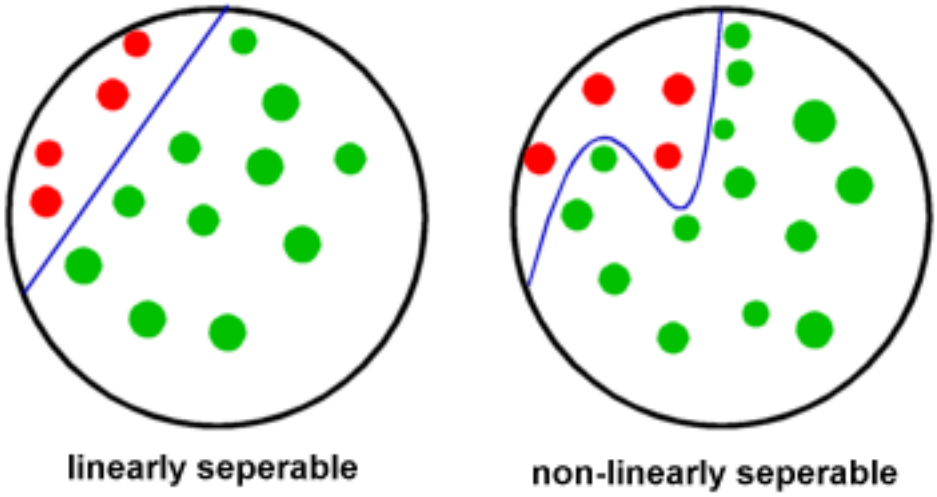


Figura 11 - Dati linearmente e non-linearmente separabili

I dati vengono quindi mappati attraverso una funzione φ detta *funzione kernel* in un nuovo spazio chiamato *spazio delle caratteristiche*, dove il problema è ricondotto al caso di dati linearmente separabili. Ci possono essere diverse funzioni kernel: lineare, polinomiale, funzione su base radiale (RBF) e sigmoide. Generalmente ci si affida ad RBF ma le altre funzioni possono ritornare utili per casi specifici. In *Figura 12* viene rappresentata graficamente una funzione kernel che porta i dati in uno spazio dove risulteranno essere separabili linearmente attraverso un iperpiano.

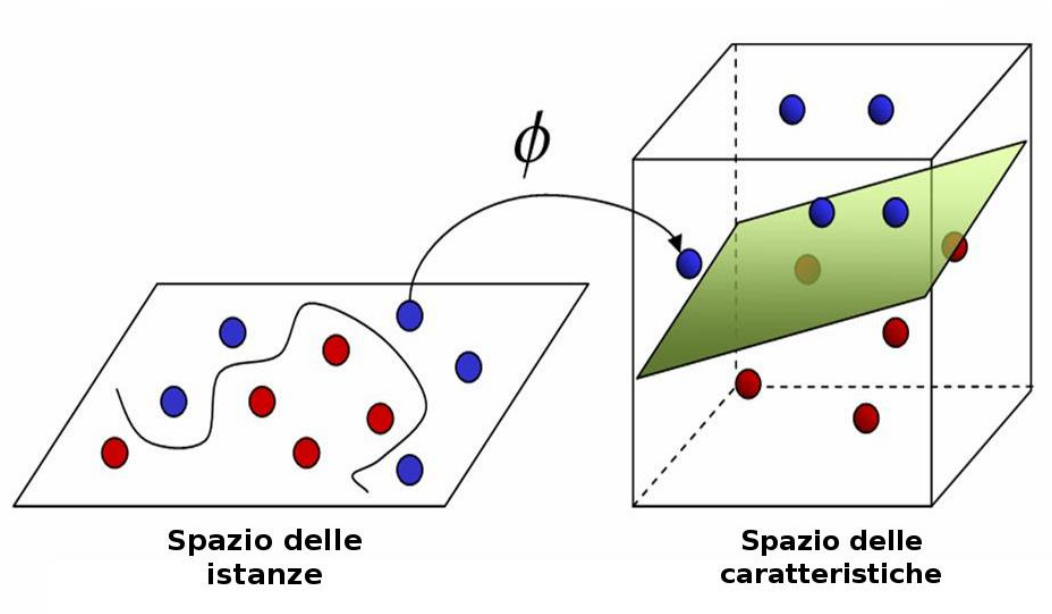


Figura 12 - Esempio di funzione kernel

Dopo esserci ricondotti al caso base di dati linearmente separabili è necessario, nello spazio delle caratteristiche, individuare tra tutti gli iperpiani che possono fungere da separatori, quello migliore. E' indispensabile scegliere infatti l'iperpiano che offre un il margine più elevato, ossia la massima distanza tra i vettori di supporto che rappresentano i limiti delle due categorie (*Figura 13*). Infatti, più elevata è la distanza tra i vettori di supporto, minore è l'errore di classificazione della SVM.

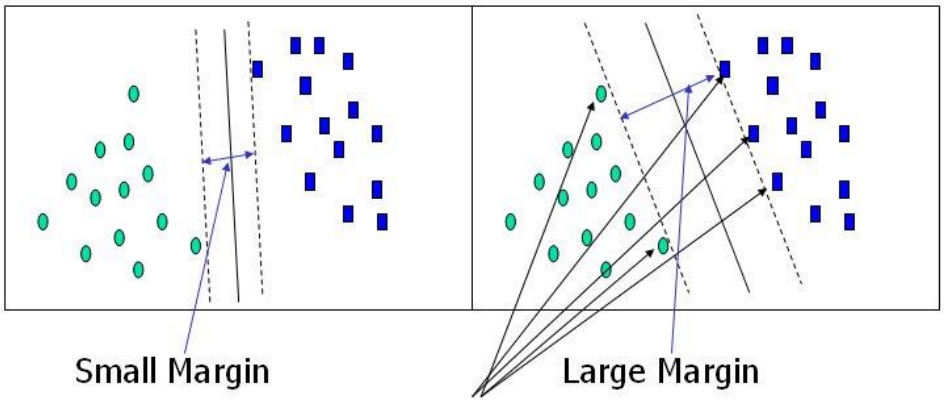


Figura 13 - Iperpiano di separazione con il minor margine

In questo capitolo sono state elencate le tipologie di strumenti che ci consentono di raggiungere gli obiettivi prefissati; avevamo bisogno di un linguaggio, di un editor per poter esprimere frasi in tale linguaggio, di un motore per poter lavorare sui dati e di un dispositivo per effettuar il riconoscimento delle pose dei soggetti da monitorare.

Per quanto riguarda il linguaggio sono state analizzate diverse alternative tra i generatori di parser più diffusi: Gold sembra essere la soluzione più adatta se si vuole avere in output codice in più linguaggi di programmazione, mentre tecnologie come JavaCC e Antlr risultano utili quando lavoriamo su piattaforma Java. Un caso particolare è Xtext che, oltre ad essere un generatore di parser, basato su Antlr, ed essere integrato nell'IDE Eclipse, consente di generare in modo semplice un editor per il linguaggio.

Si è poi passati alle tipologie di strumenti utilizzabili per la realizzazione del motore, concentrandoci sui sistemi a regole ed in particolare su Drools che è una soluzione potente ed integrata nell'ambiente di sviluppo Eclipse.

Si è descritta la teoria del Calcolo degli Eventi, filo conduttore del nostro lavoro e base sulla quale è stato sviluppato il linguaggio.

Il passo successivo è stato concentrarsi sulla ricerca delle tecnologie per il riconoscimento dell'attività umana; da una parte si sono elencate le alternative per quanto riguarda i sensori ad oggi presenti nel mercato, focalizzando l'attenzione su Microsoft Kinect che risulta essere un dispositivo supportato, facilmente reperibile e con un costo relativamente contenuto; dall'altra si è presentato il problema degli strumenti per classificare le informazioni estratte dai sensori, mostrando come soluzioni come la libreria Weka riescano, grazie ad algoritmi ad apprendimento supervisionato, a rispondere alle nostre necessità.

Capitolo 2

Implementazione

Di seguito verranno motivate le scelte fatte in ambito tecnologico e si entrerà in dettaglio nella descrizione del lavoro svolto.

2.1 Non solo fisioterapia

Sebbene lo scopo finale del nostro lavoro sia realizzare un sistema per consentire ad uno specialista di andare a definire esercizi composti da una serie di pose e mettere in atto un meccanismo per il quale sia possibile monitorare i movimenti e le posizioni assunte dal paziente e verificare la corretta esecuzione delle pose stesse, si è voluto generalizzare il più possibile la parte del progetto non strettamente legata al caso di studio specifico.
Sebbene infatti con la nostra soluzione sia possibile definire una serie di pose, il sistema può potenzialmente essere utilizzato per definire comportamenti che nulla hanno a che fare con il la fisioterapia o in generale con il riconoscimento dell'attività umana.
Al blocco generale che comprende l'editor del linguaggio e il motore è quindi agganciato il plug-in per il riconoscimento delle pose attraverso un'interfaccia per la notifica degli eventi; potremmo però prevedere di sostituire al plug-in per il rilevamento delle pose assunte dal corpo umano un plug-in che riconosce tramite per esempio dei sensori di pressione il passaggio dei pezzi nelle diverse parti di un nastro trasportatore di una catena di montaggio; questo plug-in invierà anch'esso delle notifiche al motore tramite un interfaccia ad eventi. In *Figura 14* è riportato schematicamente il concetto di plug-in: oltre al sistema per la rilevazione delle pose si potrebbero realizzare altri

blocchi che vadano ad interfacciarsi con il sistema generale inviando degli eventi.

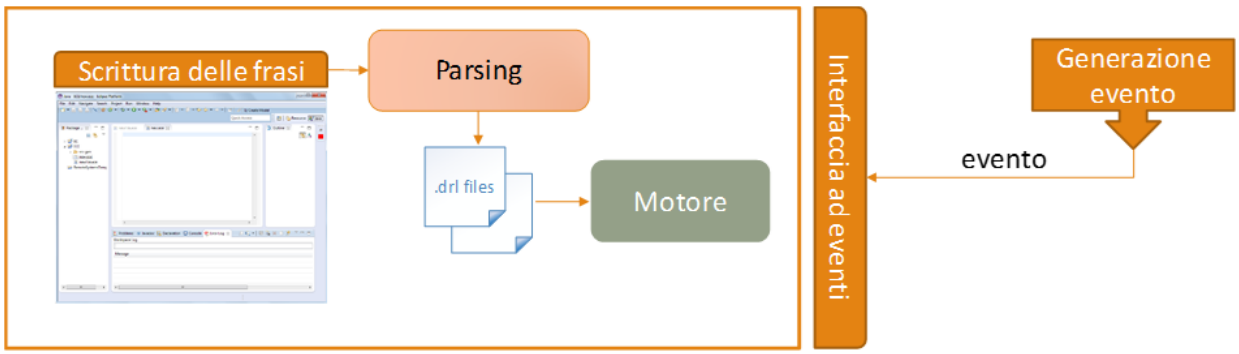


Figura 14 - Struttura a plug-in

Vengono alla luce quindi le potenzialità del sistema che può essere utilizzato nei più disparati ambiti applicativi. Di seguito verranno esplicitate le diverse tecnologie utilizzate che fanno parte del blocco generale.

2.2 Linguaggio ed Editor

2.2.1 Calcolo degli eventi e aspettativa

Prima di esplicitare le scelte fatte in ambito di linguaggio ed editor, risulta utile dire perché è stato scelto il formalismo del Calcolo degli Eventi (EC) per la rappresentazione dello stato del sistema e del comportamento che esso abbia. L'obiettivo era realizzare un sistema intelligente che permettesse di poter definire una serie di operazioni da poter svolgere se degli eventi vengono notificati. Il formalismo EC si adatta alla perfezione a rappresentare il comportamento del sistema; EC infatti permette di esprimere COSA avviene QUANDO si verifica un determinato evento.

Per i nostri scopi si rendeva necessario utilizzare un formalismo che consentisse anche di rappresentare le aspettative, ossia il concetto

astratto di attesa che si verifichi una condizione futura, aspettativa scatenata dal verificarsi di un determinato evento. ECE (Event Condition Expectation) [38] si presta bene agli scopi che volgiamo raggiungere, perché consente di poter esprimere COSA CI ASPETTIAMO avvenga QUANDO di verifica un determinato evento.

2.2.2 Il linguaggio

Per sviluppare il nostro linguaggio ci siamo affidati ad Xtext. Questo framework consente di andare a realizzarne la sintassi. Tra i vantaggi di Xtext vi sono quello di essere integrato nell'ambiente di sviluppo Eclipse e di essere fortemente compatibile con il mondo Java, requisiti fondamentali perché sono questi gli strumenti che sono stati utilizzati per realizzare l'intero progetto. Diversamente dai semplici parser che generano codice in uno specifico linguaggio, Xtext in aggiunta crea un Abstract Syntax Tree, ossia una struttura navigabile contenete il codice generato.
Lo strumento scelto inoltre permette di realizzare in automatico un Editor per il linguaggio definito: Xtext va infatti a creare un plug-in per Eclipse in modo automatico, un editor dove è possibile testare il linguaggio appena definito. La sua immediatezza, la piena compatibilità con le tecnologie utilizzate nel resto dell'applicazione, nonché la possibilità di generare in automatico e senza ulteriori sforzi un editor grafico per il linguaggio definito ci hanno fatto propendere per Xtext per questa parte del progetto.

Nel nostro linguaggio vogliamo poter scrivere frasi di tipo ECE ossia frasi EC con l'aggiunta della possibilità di poter esprimere un'aspettativa.

I mattoni fondamentali con i quali andremo a comporre una frase del linguaggio sono gli *Eventi*, gli *EcContext* (relativi al calcolo degli eventi puro) e/o gli *ExpContext* (relativi alle aspettative). Ogni frase inizia con la keyword “on”.

```
EceModel:
	statements+=Statement*;

Statement:
	'on' event=Event contextsList=ContextsList ';';

Event:
	eventName=ID
	('(' params+=EventFeature (',' params+=EventFeature)* ')')?;

EventFeature:
	name=ID;

ContextsList:
	(ecContextsList=EcContextsList)? (expContextsList=ExpContextsList)?;

EcContextsList:
	ecContexts+=EcContext (',' ecContexts+=EcContext)*;

ExpContextsList:
	expContexts+=ExpContext (',' expContexts+=ExpContext)*;
```

Un evento, oltre al proprio nome, può contenere dei parametri che potranno essere richiamati nella frase all’interno dei contesti.

A sua volta un contesto, dipendentemente dal tipo, contiene altri componenti interni.

Un *EcContext* inizia sempre con la keyword “set” e contiene un *Fluent*. Un *Fluent* è la rappresentazione di un fluente, con il valore che dovrà andare ad assumere. Ogni *Fluent* infatti comprende un campo che contiene il nome del fluente e un campo, preceduto da “to” che contiene un’espressione, valore al quale viene settato il fluente.

```
// EC CONTEXT
*********************************************************************
 EcContext:
       'set' fluent+=Fluent /*(',' fluent+=Fluent)**/;


Fluent:
       name=ID
       ('to' valuePart=ToRule)
       //('in' timePart=InRule)?
       //('if' condPart=ConditionRule)?;
```

```
on MyEventOne (myParam)

       set MyFluentYellow to myParam;
```

Un ExpContext invece inizia sempre con la keyword “expect”, seguita da una espressione che rappresenta la condizione che dovrà essere soddisfatta; è possibile inoltre definire se si vuole inserire una deadline temporale entro la quale si vuole che la condizione sia rispettata dopo che l’evento scatenante si è verificato.

```
// EXP CONTEXT
*********************************************************************
 ExpContext:
       'expect' finalCondition=ConditionRule
       (allenOp=AllenOp time=INT)?
       //('if' initialCondition=ConditionRule)?
       ('onFulf' actionF=ID)?
       ('onViol' actionV=ID)?;

AllenOp:
       {AllenOperator} value=('before');
```

```
on MyEventTwo

       expect MyFluentBlue==42 before 10;
```

Infine, preceduti dalle keyword “onFulf” o “onViol” si possono definire gli *eventi interni* da lanciare nel caso l’aspettativa venga rispettata o violata; questi tipi di eventi sono di tipo particolare: non vengono infatti notificati dai plug-in esterni ma il loro nome viene utilizzato in seguito nell’editor ogni qual volta l’utente voglia definire, attraverso una nuova frase, il comportamento che bisogna andare ad eseguire quando si verifica l’evento interno (quindi quando l’aspettativa definita in una frase precedente è rispettata o violata).

```
on MyEventTwo

       expect MyFluentRed==65 before 10  onFulf MyExpectationFulf;


on MyExpectationFulf

       set MyFluentGreen to 100;
```

Per quanto riguarda le espressioni che possono essere inserite in un EcContext, come il valore che deve assumere un fluente o la condizione da valutare in un ExpContext, esse dovranno essere definite come composizione di espressioni a loro volta, consentendo l’utilizzo di simboli grafici con il significato che hanno in tutti i principali linguaggi di programmazione; non essendo il core del nostro lavoro la sola definizione di un linguaggio che possa rappresentare tutte le possibili combinazioni di espressioni, si è deciso di implementare solo gli operandi utili alla trattazione e agli scopi che ci siamo posti. Il linguaggio, così come è strutturato, risulta comunque di facile estensione. In *Figura 15* è rappresentata schematicamente la struttura di una frase del linguaggio.

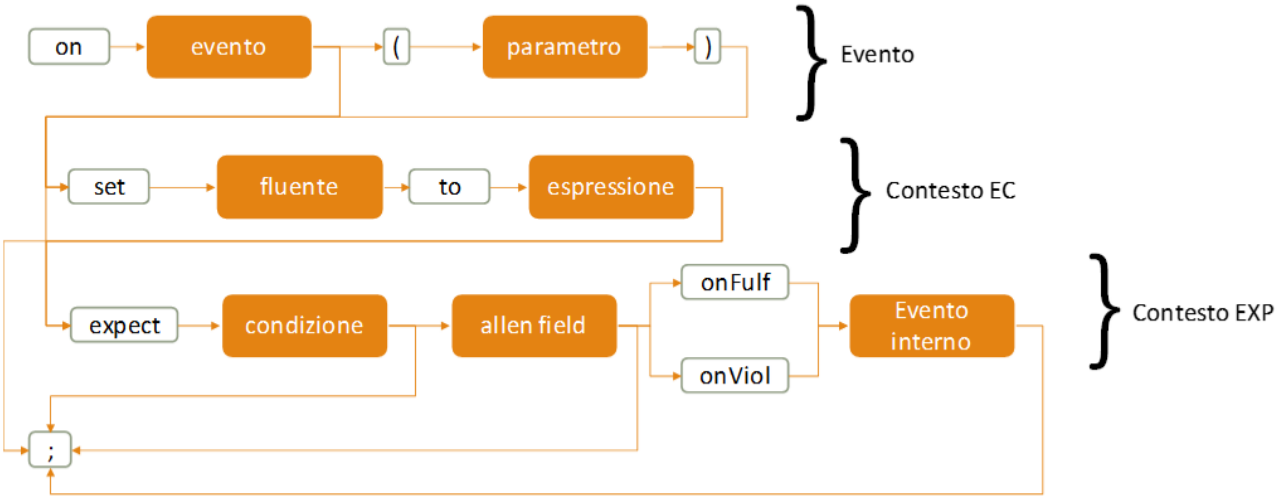


Figura 15 – Struttura di una frase del linguaggio

2.2.3 Generazione struttura intermedia Model

Una volta completata la definizione del linguaggio e configurato l'editor, si andranno a scrivere le frasi con la sintassi corretta. Xtext infatti consente di mettere in atto una serie di controlli real-time che agiscono nel momento stesso in cui l'utente va a scrivere le frasi, segnalando eventuali errori, sia di battitura che logici. Una volta che il file è stato editato correttamente Xtext genera automaticamente un Abstract Syntax Tree (AST), ossia una struttura dati organizzata ad albero che contiene tutti gli statement scritti, organizzati gerarchicamente.

Questa struttura può essere espansa e navigata in modo da generare ulteriore codice. Nel nostro caso AST risultava una struttura troppo legata alla sintassi del linguaggio e quindi è stato deciso di effettuare una traduzione di AST in un oggetto Java che, oltre ad ampliare la compatibilità con il resto dei componenti del progetto, risulta avere una struttura più congeniale alla manipolazione del suo contenuto che avviene nelle fasi successive.

Sempre con l'aiuto di Xtext è stato quindi possibile generare una classe MainECE con al suo interno definito un metodo statico getModel() che restituisce un oggetto di tipo Model che è la struttura dati che utilizzeremo delle fasi successive e che contiene di fatto le informazioni scritte nell'editor dall'utente, solo con una impostazione più consona alle operazioni che avremo intenzione di effettuare su tali dati. La struttura del model è riportata in *Figura 16* dove si può notare lo stretto legame che mantiene con la forma delle frasi del linguaggio.

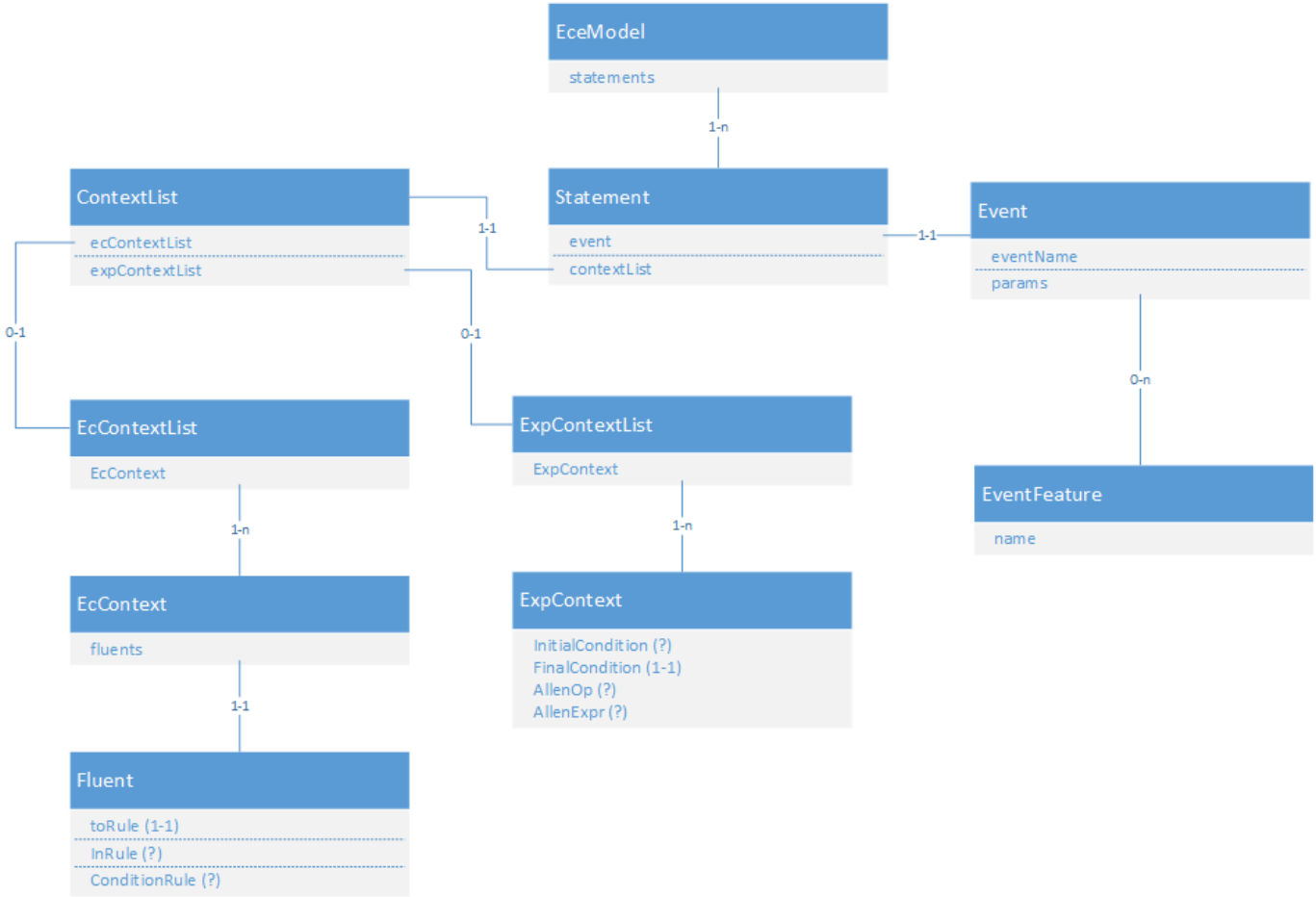


Figura 16 - Struttura del Model

2.3 Motore

Abbiamo bisogno a questo punto di un motore per elaborare l'informazione fornitaci dall'utente, facendola interagire con i segnali provenienti dal mondo esterno. In particolare, nel nostro caso specifico, dovremo codificare le informazioni descritte del Model in regole corrispondenti e generare ulteriore conoscenza all'arrivo nel sistema di eventi esterni, notificati dal plug-in agganciato al motore.

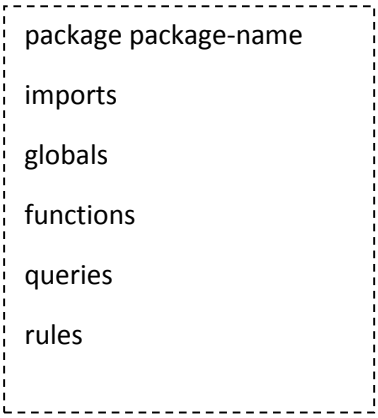
2.3.1 Drools

La nostra implementazione del Calcolo degli Eventi si basa sull'utilizzo di un sistema a regole di produzione (PRS) per costruire un sistema esperto che, sulla base della conoscenza espressa attraverso una serie di fatti e regole, sia in grado di generare ulteriore conoscenza. Vi sono differenti PRS a disposizione, ma noi abbiamo deciso di utilizzare Drools, sia per la sua spiccata propensione ad interfacciarsi con Java, sia perché è stata già utilizzata in passato nel nostro gruppo di lavoro.

Drools è un sistema a regole di produzione basato sul meccanismo del forward-chaining e consente di gestire sia la modellazione della conoscenza, sia la logica di business; sono presenti due memorie: una Memoria di Lavoro (WM – Working Memory) che contiene in ogni istante al proprio interno tutti e soli i fatti conosciuti, siano essi etichettati come eventi, fluenti o altro, e la Memoria di Produzione (PM – Production Memory) che contiene invece le regole che gestiscono i processi del dominio e vengono attivate a seconda dei fatti presenti di volta in volta nella WM.

Drools è suddiviso in moduli, tra i quali spicca Drools Fusion che consente di etichettare fatti come Eventi in modo da associarne un timestamp e poter quindi effettuare confronti in modo più semplice e naturale.

Le regole Drools, che come già detto sono contenute nella PM, sono inserite in specifici file con estensione .drl dentro le quali è possibile definire inoltre anche query, funzioni e altri elementi a supporto della logica di business che vogliamo implementare.



Le regole sono strutturate secondo un pattern specifico. Dopo il nome della regola, preceduto da "rule" viene inserita la keywork "when". A questo punto viene definita la cosiddetta LHS (Left Hand Side) che contiene le condizioni rispetto alle quali è vincolato lo scattare della regola. In LHS possiamo indicare quindi che certi fatti siano presenti nella WM, o che i loro parametri abbiano un determinato valore, o addirittura è possibile valutare delle espressioni. Segue quindi la RHS (Right Hand Side), preceduta dalla keyword "when", che di fatto contiene le operazioni da eseguire se le condizioni specificate in precedenza sono verificate. Qui in sostanza è possibile andare ad inserire, rimuovere o modificare fatti nella PM, oltre che andare a specificare direttamente del codice Java.

2.3.2 Struttura generale

Il motore realizzato contiene al suo interno due entità: un Model e una Session.

Il Model comprende tutte le informazioni sulla logica che sono state definite dall'utente attraverso l'editor. Essendo le frasi della sitassi del

linguaggio una serie di contesti contenenti le operazioni da svolgere nel momento in cui dei determinati eventi vengono notificati, il Model conterrà proprio questo tipo di istruzioni, ossia per ogni evento di interesse, le modifiche ai fluenti e/o il lancio di aspettative. La Session invece, come dice il nome, rappresenta una sessione Drools.

In *Figura 17* è presente una rappresentazione concettuale del motore realizzato, dove sono indicati i diversi stadi che attraversa quando è utilizzato.
In una prima fase, una volta recuperato il Model, andremo ad inizializzare il motore: questa operazione consiste nella generazione dei file .drl necessari alla sessione Drools per funzionare. Una volta che i file sono stati caricati nella sessione è possibile farla partire: le regole verranno caricate e lette e, dopo aver verificato la loro correttezza, la logica di business verrà inserita nella PM e i fatti che vogliamo siano presenti dopo l'inizializzazione immessi nella WM. Da questo istante in avanti la Session rimane in ascolto di eventi provenienti dall'esterno, eventi che una volta riconosciuti correttamente andranno ad interagire con la logica del motore e genereranno nuova conoscenza.

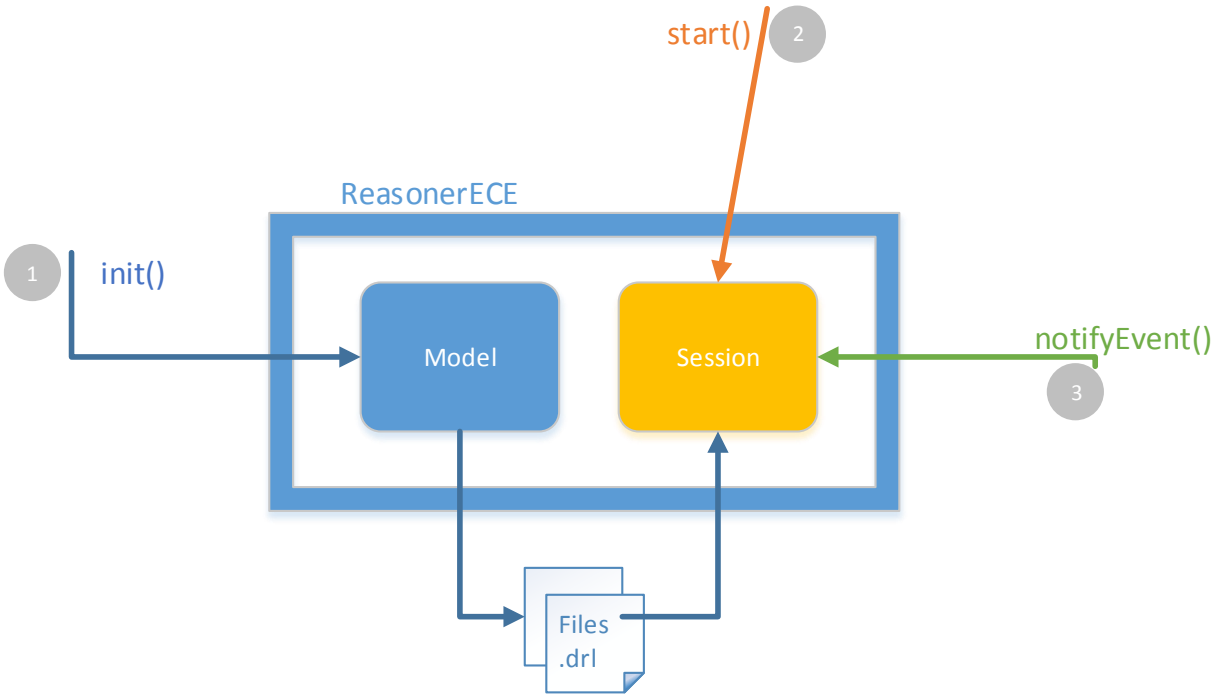


Figura 17 - Componenti del motore logico

Abbiamo quindi bisogno di un meccanismo che in fase di inizializzazione effettui una lettura del Model e vada a generare i file .drl utili al corretto funzionamento della Session Drools.

2.3.3 Generazione regole Drools

Dopo aver definito le frasi attraverso l'editor, abbiamo generato la classe Java dalla quale è possibile ottenere un oggetto di tipo Model, che possiamo immaginare come l'entità rappresentante il problema che l'utente ha deciso di manipolare. Ora ci si trova nella situazione di dover utilizzare queste informazioni e farle interagire tra di loro per generare nuova conoscenza, in particolare questo lavoro consiste nel riconoscere gli eventi, valutare condizioni ed agire di conseguenza.
Se vogliano andare ad utilizzare le informazioni del Model in un sistema a regole si renderà necessaria una ulteriore fase di traduzione, per consentire al nostro modello del problema di avere una forma e una sintassi tali da essere accettato e correttamente gestito dal sistema a regole scelto.

Drools, come già detto, richiede che la logica sia espressa attraverso dei file .drl, file che contengono al loro interno regole e definizioni di entità utili alle regole stesse. Dovremo quindi andare a generare questi file, scrivendo in essi delle regole (ECE-rules), per istruire Drools sul comportamento che dovrà assumere.
Sono state intraprese differenti strade per arrivare ad una soluzione, la prima è stata abbandonata, ma viene presentata per completezza e perché consente di comprendere meglio il funzionamento del sistema e gli scopi che ci si pone di raggiungere.

2.3.3.1 Traduzione con Drools

La prima tecnica per generare i file .drl era quella di inserire in una sessione Drools preventiva gli Statement contenuti nel Model scrivendo dei file .drl statici con la logica per far funzionare questa sessione. Essa avrebbe dovuto riconoscere il contenuto degli statement e generare di conseguenza nuovi .drl utili all'ultima sessione finale. A

loro volta i drl necessari al funzionamento della sessione preventiva erano specifici per gli statement presenti nel model e quindi la loro generazione doveva avvenire dinamicamente, attraverso un'ulteriore sessione Drools preventiva istruita questa volta con file drl statici scritti manualmente.
Avremmo avuto quindi la necessità di eseguire più sessioni in sequenza. Questa soluzione comportava una complessità di realizzazione non banale e presto è risultato chiaro che l'utilizzo di Drools in più fasi per la creazione dinamica di regole complesse non era la soluzione più adatta.

2.3.3.2 Traduzione con il pattern Visitor

Scartata l'opzione di creare regole attraverso ulteriori sessioni Drools si è deciso di applicare l'approccio tramite pattern Visitor. Questo tipo di pattern consente di andare a scandire qualsiasi elemento all'interno del Model, a qualsiasi profondità esso si trovi, ed è stato utilizzato per effettuare il parsing delle informazioni contenute nel Model per generare dinamicamente i file drl dei quali avevamo bisogno.

Esiste un visitor che scandisce statement e genera le dichiarazioni necessarie in un file .drl; vi sono visitor che vanno a generare file di regole relative alla logica di business per contesti EC e contesti EXP; altri Visitor servono per generare strutture dati intermedie, scandendo gli statement, utili ad altri visitor.

L'utilizzo del pattern visitor permette di scomporre il problema della generazione delle regole in più sotto-problemi, suddividendo quindi la complessità totale; ha inoltre il vantaggio di essere realizzata totalmente con il linguaggio Java, consentendo quindi una manipolazione più semplice del Model e dei suoi campi, cosa che non avveniva per la generazione con l'utilizzo di sessioni Drools, dove la risoluzione di un bug o l'aggiunta di una feature comportava un costo in termini di tempo estremamente più elevato.

Tipi di file .drl generati dinamicamente

La funzione che deve avere il motore nel nostro progetto è quella di rimanere in attesa di notifiche di eventi dal mondo esterno e, a seconda della logica determinata dalle informazioni estratte dal Model, andare a creare nuova conoscenza. In particolare nel nostro caso specifico Drools dovrà essere in grado di riconoscere un evento, lanciare le aspettative se previste, controllare se quest'ultime siano verificate o meno e andare ad agire di conseguenza nel caso di un successo o di una violazione.

La logica di business contenuta nei file drl è suddivisa in tre categorie: regole relative a dichiarazioni, operazioni e aspettative. Le regole di ciascuna categoria vengono raggruppate in un file .drl separato. I tipi primitivi sono fissi e vengono invece dichiarati in un file .drl ulteriore e non generato dinamicamente.

Gli elementi che il nostro motore dovrà andare a gestire saranno quindi di diversi tipi:

- Event: tutti i fatti che vogliamo gestire come eventi estendono il tipo Event, il quale è etichettato come *evento* in modo da essere gestito utilizzando le potenzialità di Drools Fusion. Possiede inoltre dei campi come *time* per memorizzare il timestamp e un campo per memorizzare un parametro.
- Fluent: ogni fluente estende questo tipo.
- Sample: viene inserito un sample per ogni fluente presente nel sistema; esso possiede dei campi come *fluent* e *value* che servono rispettivamente a specificare il fluente al quale il sample si riferisce e ad esplicitarne il valore.

Declarations.drl

Questo file contiene tutte le dichiarazioni utili all'interno del sistema, nonché una serie di regole utili in fase di inizializzazione. Vengono dichiarati tutti gli eventi che sono specificati nell'editor, i fluenti e i parametri degli eventi; vengono inoltre inseriti i fatti relativi ai fluenti dichiarati e un campione (sample) per ognuno di essi. Ciascuna specifica dichiarazione è espressa come estensione del proprio tipo di elemento generico al quale appartiene. Questi tipi

generici sono dichiarati staticamente in ulteriori file .drl e contengono la struttura generale di ciascun tipo di elemento. In questo modo è possibile andare a semplificare il lavoro di generazione dinamica di codice, utilizzando, senza doverlo specificare ogni volta, feature del tipo di elemento generico al quale l'elemento specifico appartiene. Una struttura di questo tipo inoltre consente un'organizzazione del codice migliore.
Un esempio di dichiarazione potrebbe essere quella di un evento espresso nel Model, dichiarazione dove si estende il tipo generico Event il quale al suo interno possiede annotazioni per l'assegnazione di un timestamp all'evento e campi specifici come una lista di parametri.

In particolare il Model viene scandito andando a dichiarare ciascun evento come estensione del tipo Event; Per ognuno dei parametri o fluenti presenti negli statement viene dichiarato il fluent corrispondente e inserito temporaneamente il sample relativo senza specificarne il valore.
Infine per ognuna delle aspettative degli statement viene dichiarata un'aspettativa, anch'essa estensione di Event, che aggiunge però un campo che servirà per rappresentare lo stato dell'aspettativa stessa.

Operations.drl

Qui abbiamo tutte quelle regole che determinano il comportamento del sistema e che non dipendono dal successo o meno di un'aspettativa; in altre parole in questo file viene rappresentata sotto forma di regole la logica di business dei contesti EC, ossia quelle regole che non hanno nulla a che fare con il concetto di aspettativa.
Viene generata una regola per ogni statement che contiene un contesto EC; si ricorda che un contesto EC non fa altro che, al verificarsi di un dato evento, settare un fluente ad uno specifico valore, sia esso direttamente un valore statico, un fluente già presente nel sistema, un parametro passato insieme all'evento o una combinazione di questi.
All'interno dell'LHS della regola si andrà a richiamare l'evento che dovrà scatenare la regola stessa e inoltre tutti i fluenti (e sample relativi) che rientreranno nel calcolo dell'espressione da associare ad un dato fluente output. In RHS invece andremo a modificare i sample dei fluent

che vogliamo settare, settando se necessario i sample a valori dei fluent recuperati in LHS. Se lo statement prevedeva anche un contesto EXP andiamo a lanciare un'aspettativa "pending" che verrà gestita dalle regole per la gestione delle aspettative.

In questo modo gestiamo separatamente i contesti EC e lasciamo le regole per la gestione delle aspettative in un file separato.

Expectations.drl

Questa raccolta di regole rappresenta il cuore della logica del sistema per quanto riguarda la trattazione delle aspettative.
In particolare per ogni statement del model che rappresenta un' aspettativa (che contiene quindi un contesto EXP) si va a generare una coppia di regole, una che ne valuta il successo, e che esegue le successive operazioni conseguenti, e una che ne valuta la violazione e applica le operazioni che devono essere eseguite in seguito a tale infrazione.
Essendo l'aspettativa una valutazione di una condizione, le due regole duali di ciascuna coppia dovranno andare a valutare la condizione di aspettativa (nonché altri vincoli di tipo temporale, come una deadline se prevista) e se tale condizione risulterà verificata(non-verificata) allora verranno eseguite le operazioni conseguenza del successo(violazione).
Nella LHS si verificherà, ad ogni nuovo evento notificato, che sia stata lanciata un'aspettativa e vi sarà il recupero dei fatti necessari alla valutazione della condizione, mentre in RHS, se sono previste delle operazioni a seguito di un rispetto o violazione dell'aspettativa, verrà lanciato un nuovo evento "interno". Tale evento è di tipo particolare in quando, a differenza di tutti gli altri eventi, che si riferiscono a notifiche provenienti da mondo esterno, esso è lanciato dal motore stesso al verificarsi di una situazione di violazione (o successo) ed è utile per l'utente riutilizzare il nome dell'evento nell'editor per specificare con un altro statement cosa andare a fare in quella data situazione di violazione (o successo).

In *Figura 18* è riportato uno schema che rappresenta la struttura delle classi che realizzano il sistema. Qui è possibile visualizzare il flusso di informazioni riguardanti le direttive espresse attraverso le frasi

dell'editor: Una volta che è stato generato il Model, esso è gestito da delle classi Visitor che si occupano di creare dinamicamente i tre file necessari al motore per operare.

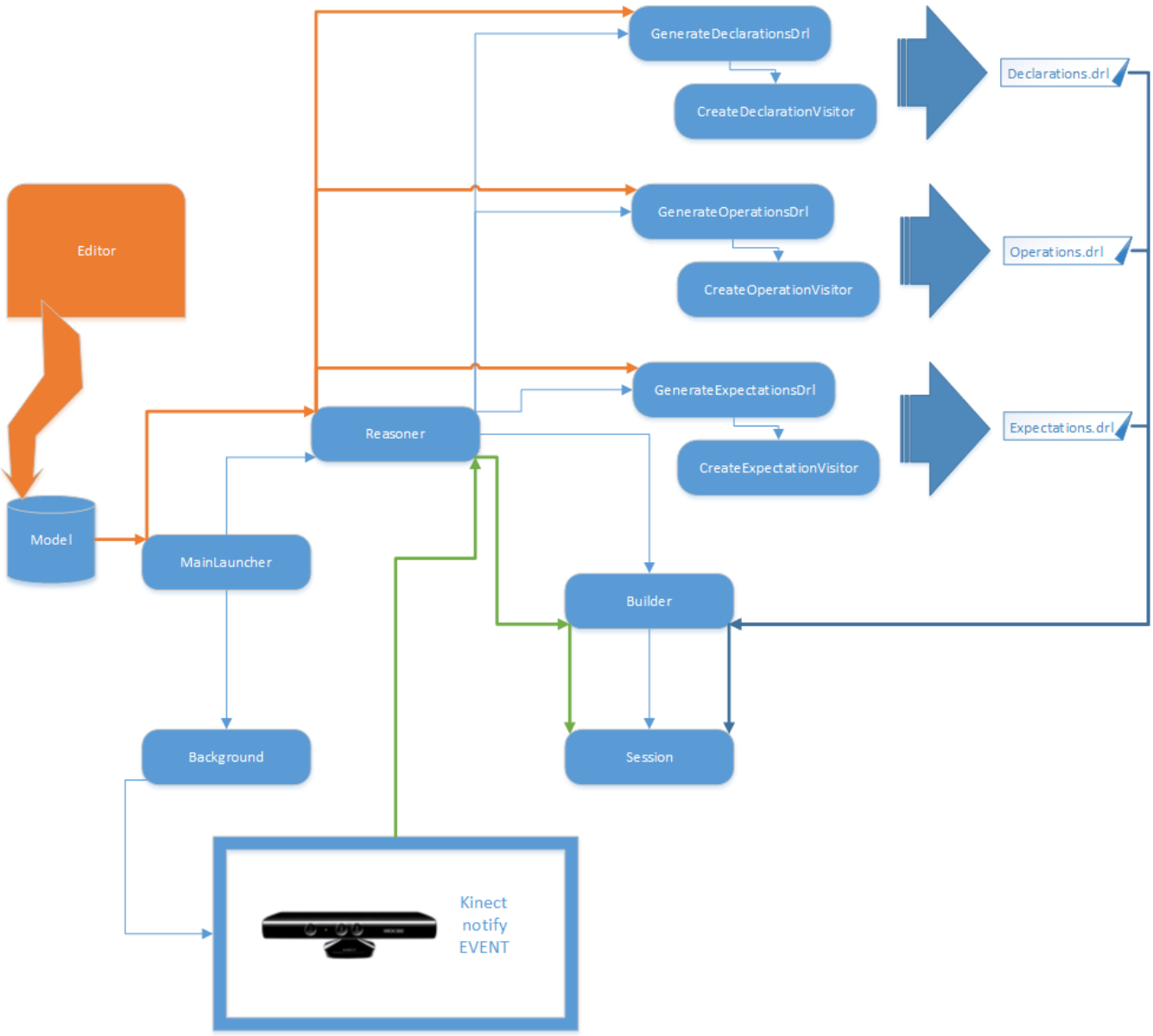


Figura 18 - Schema delle classi

In questo capitolo sono state descritte le scelte tecnologiche fatte e come si è deciso di implementare le funzionalità richieste per il sistema.

Per quel che riguarda la definizione del linguaggio e dell'editor ci siamo affidati a Xtext che, essendo un sistema integrato, basato su Java e che riunisce insieme la definizione del linguaggio e dell'editor, risulta essere lo strumento più adatto per il nostro lavoro, riducendo i tempi necessari per lo sviluppo. E' stata descritta nel particolari la grammatica scelta per il linguaggio e come vengono generate le strutture dati intermedie partendo dalle frasi del linguaggio, utili nei passi successivi.

Si è descritta l'implementazione del motore attraverso l'utilizzo di Drools, un sistema a regole potente e facile da utilizzare. E' stata descritta la fase di generazione dinamica delle regole partendo dalle frasi del linguaggio.

E' stato fatto notare anche come in realtà le funzionalità di editing del linguaggio e il motore del sistema siano totalmente indipendenti dal tipo di applicazione che si decide di "agganciare" e come, grazie alla realizzazione di un'interfaccia ad eventi per comunicare con il motore, sia quindi possibile andare ad utilizzare lo strumento realizzato anche per definire il funzionamento di applicazioni di qualsiasi genere, anche completamente slegate dall'ambito fisioterapico.

Capitolo 3

Caso di studio

Nel capitolo precedente è stato descritto nei particolari il blocco generale del progetto, ossia tutte quelle parti che sono indipendenti dal tipo di applicazione specifica con la quale ci si desidera interfacciare. In particolare è stata delineata la sintassi del linguaggio e illustrato come Xtext consenta di andare a generare una classe Java di supporto che ci possa fornire l'oggetto di tipo Model contenente la logica di funzionamento che l'utente vuole esprimere.
E' stata descritta la struttura generale del motore per il ragionamento logico ed è stato spiegato nei dettagli in cosa consiste e come avviene la generazione dei file .drl contenenti dichiarazioni e regole per il corretto funzionamenti del motore Drools.

Come è stato già anticipato, il blocco generale contenente il motore per il ragionamento mostra un interfaccia all'esterno, attraverso la quale può ricevere notifiche di eventi provenienti da plug-in differenti, ossia relativi a casi d'uso specifici. Il nostro obiettivo era di realizzare un'applicazione che ci consentisse di specificare esercizi in ambito fisioterapico e verificarne la corretta esecuzione.

In questo capitolo verrà quindi descritto nei dettagli il plug-in specifico che è stato realizzato e che si aggancia al motore di inferenza del blocco generale attraverso l'interfaccia a notifica di eventi definita.

In *Figura 19* sono raffigurate schematicamente le diverse parti dello strumento realizzato; si può notare che l'utilizzo del Kinect è garantito dall'interfaccia ad eventi ed è indipendente dal resto del sistema.

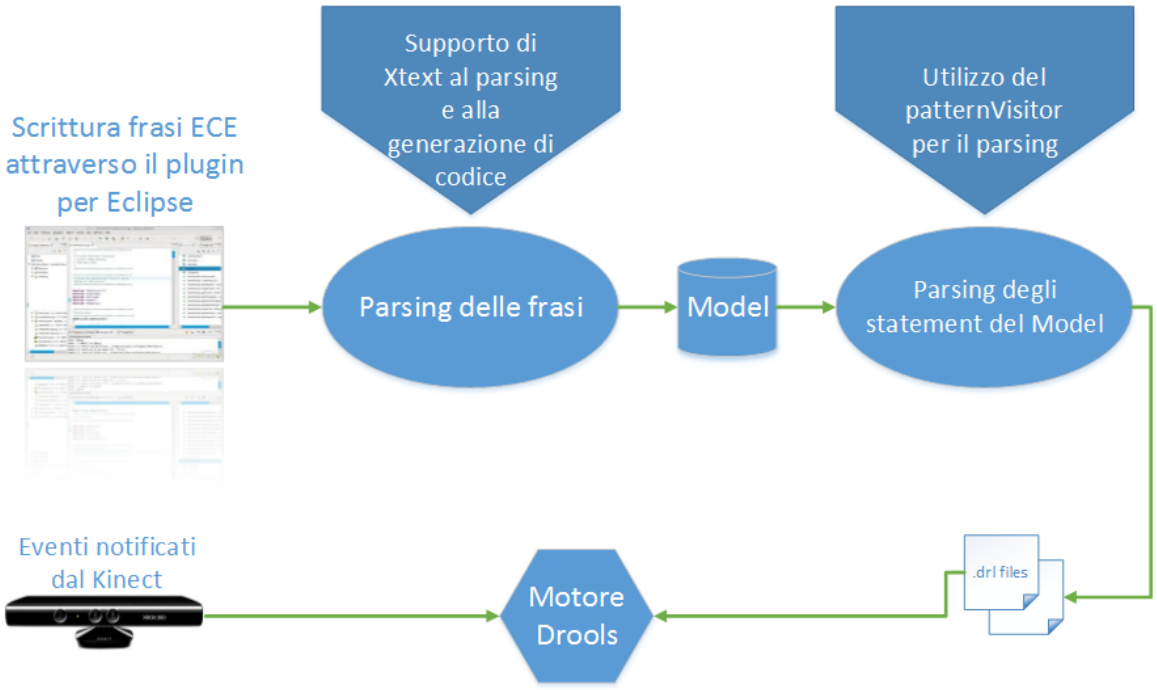


Figura 19 - Flusso dei dati

Per la realizzazione di questa parte ci si è appoggiati al lavoro svolto da Stefano Di Monte nella sua tesi [39], in particolare per quel che riguarda il rilevamento di dati tramite Kinect e il riconoscimento e stima delle pose tramite le librerie Weka. Verranno comunque qui di seguito ripresi i concetti fondamentali perché parte integrante del nostro lavoro.

3.1 Rilevamento dei dati

Per quanto riguarda la tecnologia hardware utilizzata per il recupero delle informazioni sulle pose assunte dai pazienti ci si è rivolti alla telecamera Kinect. La scelta verso questo tipo di dispositivo è stata dettata dal fatto di essere supportato sia dalla comunità open, sia dalla casa proprietaria Microsoft che forniscono driver e altri software per l'interfacciamento; inoltre vi sono numerosi casi di utilizzo, anche in ambito accademico, grazie anche al fatto che l'hardware ha una diponibilità elevata e un costo relativamente contenuto, e questo facilita il reperimento di supporto durante la fase di sviluppo.

Lato software sono stati utilizzati i driver proprietari Microsoft per la periferica, la libreria OpenNI per la manipolazione dei dati prodotti e il middleware PrimeSense NITE per estrarre la siluette del modello da riconoscere.

Dopo le elaborazioni effettuate dal software viene restituito uno scheletro del soggetto monitorato; la struttura è determinata dalle posizioni delle giunture, e per ciascuna di esse viene restituito un array contenente le tre coordinate spaziali della giuntura rispetto al Kinect e un valore (0.0 – 0.5 – 1.0) che rappresenta il grado di affidabilità con la quale la posizione della giuntura è stata determinata.

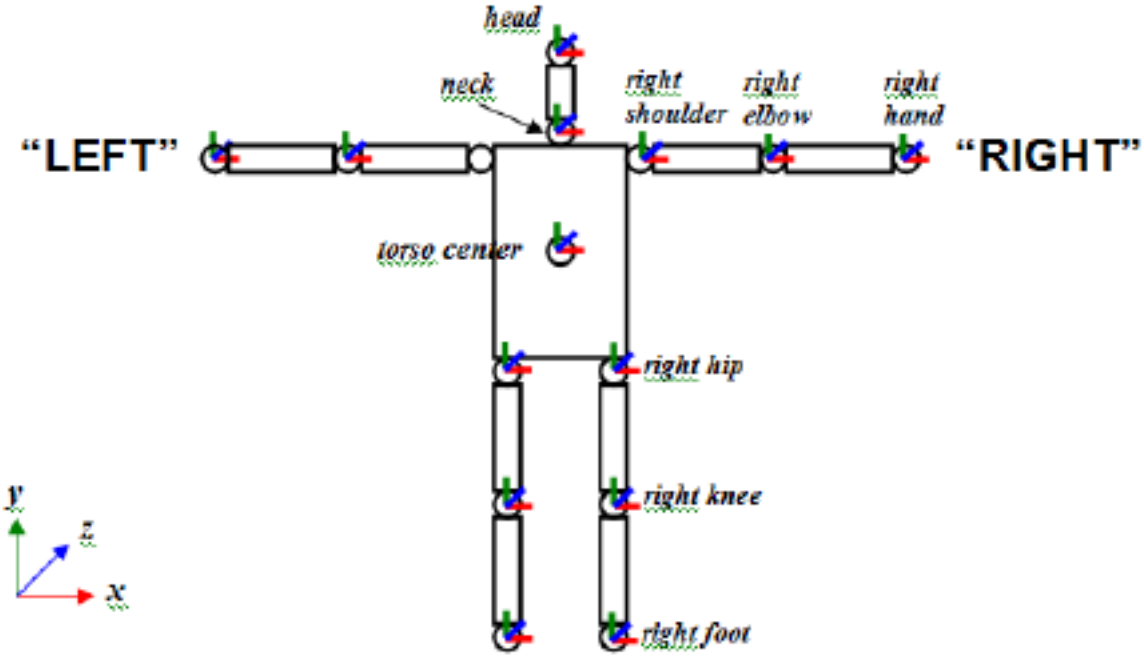


Figura 20 - Giunture monitorate da Kinect

In *Figura 20* sono riportate le giunture che sono riconosciute e che consentono di ricostruire lo scheletro del soggetto.

E' bene notare che se ci limitassimo a utilizzare tali informazioni direttamente per istruire una rete neurale potremmo incorrere in problemi sia in fase di learning sia durante il riconoscimento: infatti, essendo le coordinate spaziali riferite al Kinect, due pose di fatto identiche e corrette ma ottenute a distanze differenti rispetto alla telecamera anche di pochi centimetri, porterebbero all'assimilazione di

due pose completamente diverse se stiamo istruendo la rete, ed a una classificazione scorretta durante la fase di riconoscimento.

Si è optato quindi per uno scambio di spazio vettoriale, da quello con centro il Kinect ad uno con centro il torso. In questo modo ci portiamo a riferire tutte le altre coordinate rispetto alla giuntura torso, la quale risulterà essere la più stabile e la meno soggetta a variazioni, andando inoltre a rendere il riconoscimento indipendente dalla distanza del soggetto dalla telecamera.

3.2 Stima delle pose

Una volta ottenuti i dati relativi alle giunture e convertiti nel nuovo sistema di riferimento vettoriale è necessario utilizzare la libreria Weka per andare a classificare le varie pose assunte dal paziente. E' stato utilizzato un apprendimento di tipo supervisionato e in particolare si è scelto di utilizzare la tecnica delle Macchine a Supporto di Vettori (SVM) per classificare correttamente le istanze che è risultata essere la scelta più appropriata per la nostra applicazione [37] .

Invece di avere una SVM che riceva dati relativi a tutto il corpo si è scelto di avere più SVM, una per ogni parte del corpo per la quale è nostro interesse classificarne la posizione.
Nella *Figura 21* viene riassunto come le giunture delle diverse zone della persona riconosciuta prendono parte alla definizione delle parti del corpo.

In questo modo è stato possibile definire degli esercizi da svolgere che prevedono il movimento per esempio solo di un arto senza andare a specificare la posizione assunta da tutto il resto del corpo.

Nel lavoro di Di Monte [39] questo tipo di scelta era dettata dal fatto che nel suo caso di studio era possibile specificare una posa di un esercizio come composizione delle diverse parti del corpo.

Tabella 2 – Giunture che compongono le parti del corpo

Parte del corpo	**Composizione**
Testa	Testa + Collo
Torso Eretto Seduto/In Piedi	Torso
Torso Inclinato Seduto	Torso + Gamba Sinistra + Gamba Destra
Torso Inclinato In Piedi	Torso + Gamba Sinistra + Gamba Destra
Braccio Sinistro	Spalla Sinistra + Gomito Sinistro + Mano Sinistra
Braccio Destro	Spalla Destra + Gomito Destro + Mano Destra
Gamba Sinistra	Anca Sinistra + Ginocchio Sinistro + Piede Sinistro
Gamba Destra	Anca Destra + Ginocchio Destro + Piede Destro

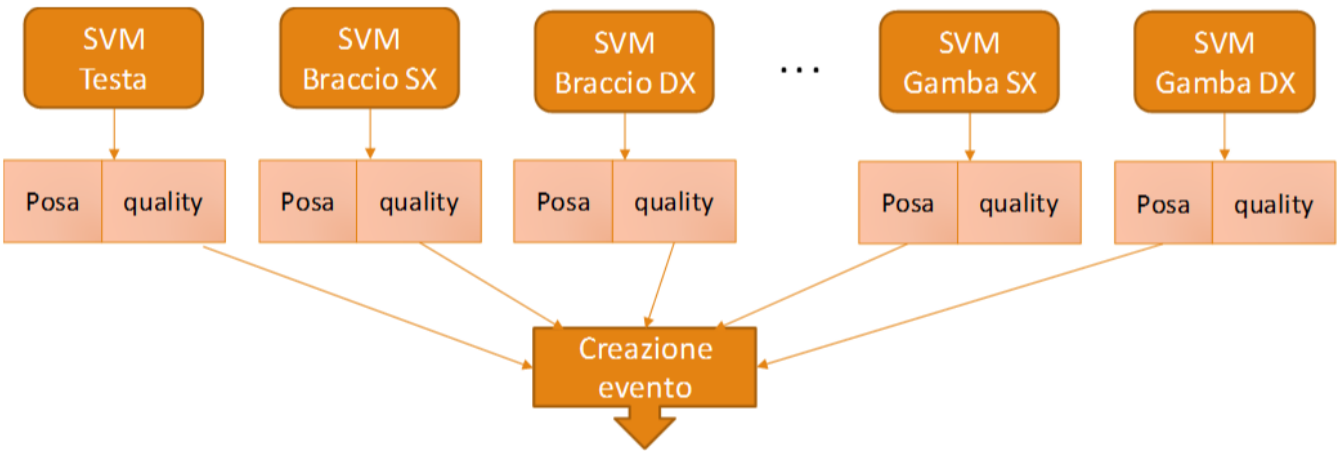


Figura 21 - Insieme delle SVM

L'idea di base era che un gruppo di SVM possa imparare su un insieme di pose meno complesso di quello in cui siano presenti tutte le giunture. L'insieme di tutte le possibili soluzioni descritto dalle giunture che costituiscono le varie parti del corpo, è infatti sicuramente inferiore all'insieme delle soluzioni che produrrebbe una posa comprendente tutte le giunture rilevate dal software del Kinect.

Nella nostra applicazione invece non è prevista la descrizione di una posa comprendente l'intero corpo ma gli esercizi vengono descritti

sempre in funzione delle singole parti del corpo, come tra l'altro era possibile fare anche in [39].
Facendo un semplice esempio, se ci fosse una sola SVM che accetta tutte le giunture del corpo, se volessimo riconoscere la posa "braccio alzato" dovremmo nella fase di learning insegnare alla rete neurale a riconoscere come "braccio alzato" tutte le pose del corpo che effettivamente hanno l'arto in quella posizione ma che presentano il resto del corpo in tutte le altre combinazioni possibili.
Una strada del genere non è certamente percorribile e quindi la scelta di avere più reti neurali, una per ogni parte del corpo risulta un'ottima soluzione per il nostro caso di studio.

3.3 Notifica degli eventi

Durante la fase di riconoscimento ognuna delle reti neurali riporta come output la posa della specifica parte del corpo ed un valore di confidenza che rappresenta quanto la posa rilevata si avvicina alla posa ideale (*Figura 21*); questo valore può quindi dare un'informazione sulla qualità con la quale il paziente sta svolgendo l'esercizio, relativamente alla singola parte del corpo. Questo dato è importante perché esso verrà passato al motore del sistema come parametro dell'evento che rappresenta la posa rilevata.
Mano a mano quindi che le pose per le parti del corpo verranno riconosciute, esse verranno passate subito, una per una, sotto forma di eventi, al motore del blocco generale. La confidenza delle pose entra qui in gioco in quanto risulterà particolarmente utile per lo specialista nel momento in cui vorrà specificare la modalità di agire del sistema durante la fase operativa, vincolando certe scelte o comportamenti in base alla qualità con cui sono state eseguite le pose prescritte.

3.4 Esempio di funzionamento

Ora andremo ad utilizzare gli strumenti descritti in questo capitolo e nel precedente per mostrare un semplice ma significativo esempio di funzionamento del sistema.

Aprendo l'editor per la prima volta ci si trova di fronte ad un file vuoto. E' possibile iniziale subito a scrivere le frasi utilizzando il linguaggio che è stato definito.

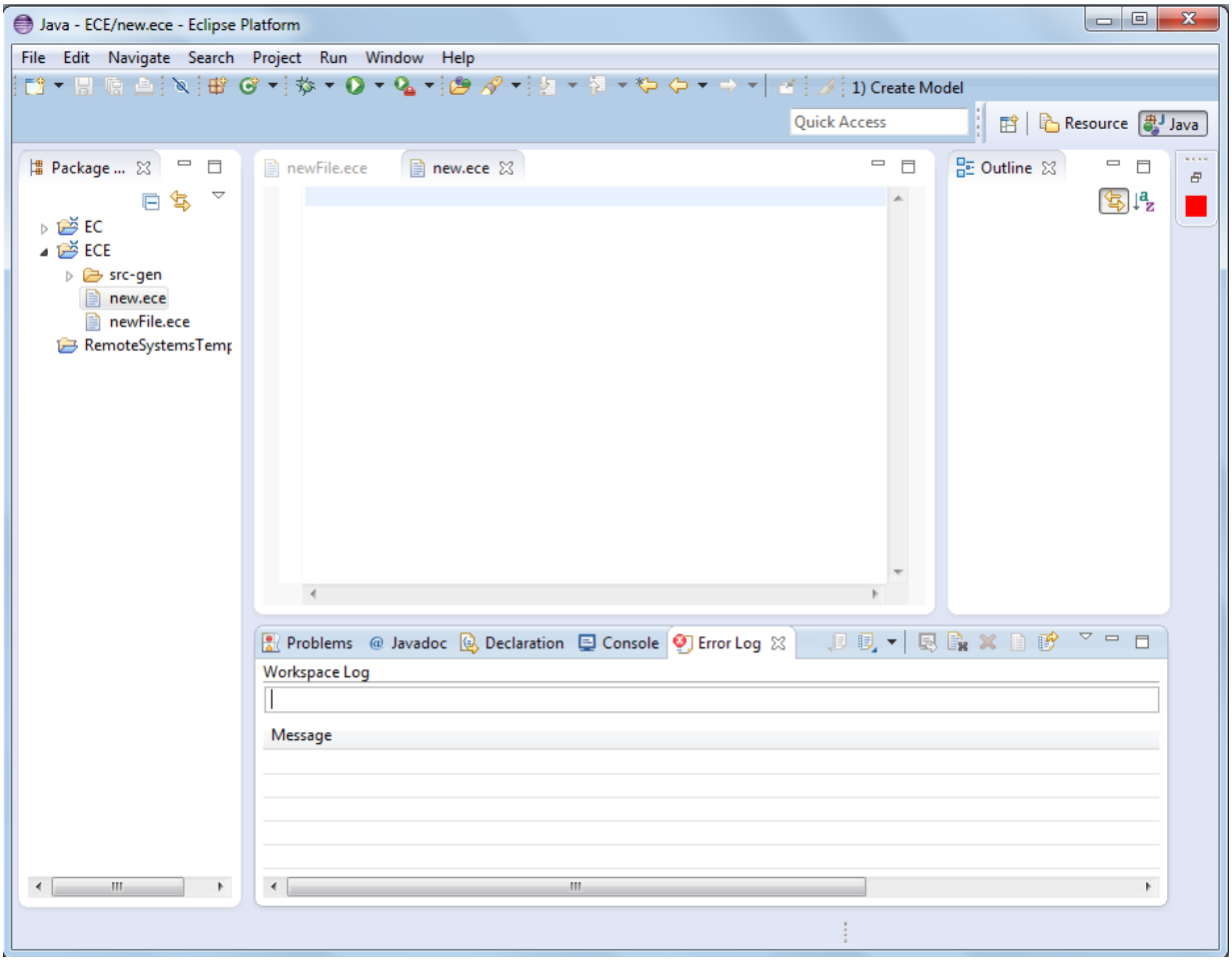


Figura 22 - Editor testuale realizzato

Attraverso lo shortcut Ctrl+Space è possibile, in ogni istante della fase di scrittura delle frasi, farci suggerire dall'editor i possibili elementi che è possibile inserire.

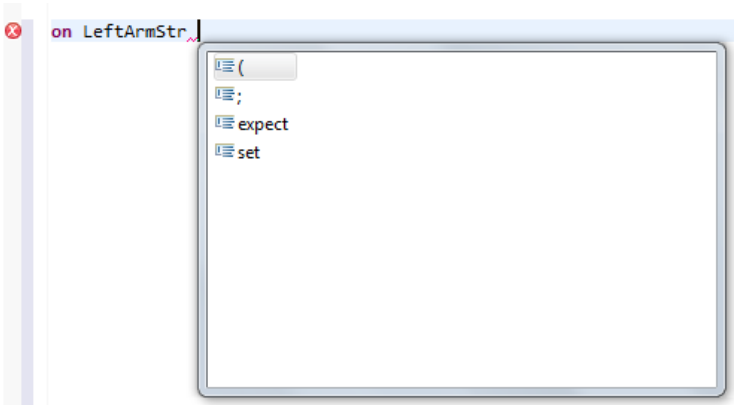


Figura 23 - Supporto alla scrittura

E' possibile richiamare segni grafici, parole chiave, oltre a fluenti utilizzati in precedenza e ai parametri degli eventi. Inoltre in tempo reale l'editor controlla ciò che viene scritto ed in caso di errori di sintassi suggerisce all'utente le possibili soluzioni al problema.

Iniziamo con lo scrivere la prima frase:

```
on Start
        set LeftArm to 999,
        set LeftArmStrConf to 999,
        set LeftArmLowConf to 999,
        set Score to 999;
```

All'evento "Start" (evento che verrà sempre lanciato automaticamente all'inizio della fase di riconoscimento) vogliamo vengano settati ad un valore qualsiasi quattro fluenti.

```
on LeftArmLow(poseConf)
        set LeftArm to 0,
        set LeftArmLowConf to poseConf;
```

All'arrivo dell'evento "LeftArmLow", ossia quando il paziente viene riconosciuto con il braccio sinistro abbassato, vogliamo che il fluente "LeftArm" che rappresenta la posizione in un dato istante del braccio venga settato a 0, mentre nel fluente "LeftArmLowConf" che rappresenta la qualità con la quale la posa è stata eseguita vogliamo venga memorizzato il valore del parametro "poseConf" passato insieme all'evento e che sappiamo appunto corrispondere alla confidenza con la quale la posa è stata riconosciuta.

```
on LeftArmStr(poseConf)
	set LeftArm to 1,
	set LeftArmStrConf to poseConf
	expect LeftArm==0 before 10 onFulf MyExpectationFulf;
```

All'arrivo dell'evento "LeftArmStr", ossia quando il paziente viene riconosciuto con il braccio sinistro orizzontale, vogliamo come per la frase precedente settare il fluente "LeftArm", questa volta al valore 1, e il fluente "LeftArmStrConf" al valore di confidenza per quella posa, passato come parametro dell'evento.
In aggiunta però, in questa frase esprimiamo il concetto di *aspettativa*: vogliamo che quando l'evento si verificherà, entro 10 secondi il fluente "LeftArm" valga 0, ovvero si sia ritornati alla posa con il braccio abbassato.
Volendo anche andare ad eseguire ulteriori operazioni in caso di successo, nel caso l'aspettativa sia rispettata vogliamo venga lanciato un evento di nome "MyExpectationFulf" il quale potrà essere utilizzato successivamente nell'editor per esprimere il comportamento se tale condizione si verificasse.

Con quest'ultima semplice frase vogliamo che quando l'evento "MyExpectationFulf" si verifica, ovvero nel momento in cui l'aspettativa alla quale si riferisce è stata verificata (quindi quando le

```
on MyExpectationFulf
	set Score to LeftArmStrConf * LeftArmLowConf;
```

due pose per il braccio sono state eseguite correttamente) al fluente "Score" sia assegnato il valore corrispondente al prodotto tra i valori di confidenza delle due pose effettuate.

La fase di editing è a questo punto conclusa e non rimane altro da fare che salvare il file e lanciare l'applicazione per il riconoscimento delle pose.

La finestra che ci viene presentata, riportata in *Figura 24*, presenta un pulsante "Kinect" con il quale ci si collega alla telecamera; L'applicazione visualizzerà una mappa di profondità in bianco e nero dell'area di fronte al sensore e inizierà a cercare di rilevare delle figure umane.
Posizionandosi davanti al Kinect la sagoma del corpo verrà evidenziata con un colore e verrà mostrato lo scheletro. Si noti che, se volessimo, come nel nostro caso, valutare solamente pose che si riferiscono al movimento di un arto superiore, non sarà necessario che nel campo visivo siano inclusa anche la parte inferiore del corpo, in quanto abbiamo una rete neurale per ogni zona e le reti neurali sono indipendenti tra loro e l'importante sarà solo avere il busto come riferimento.

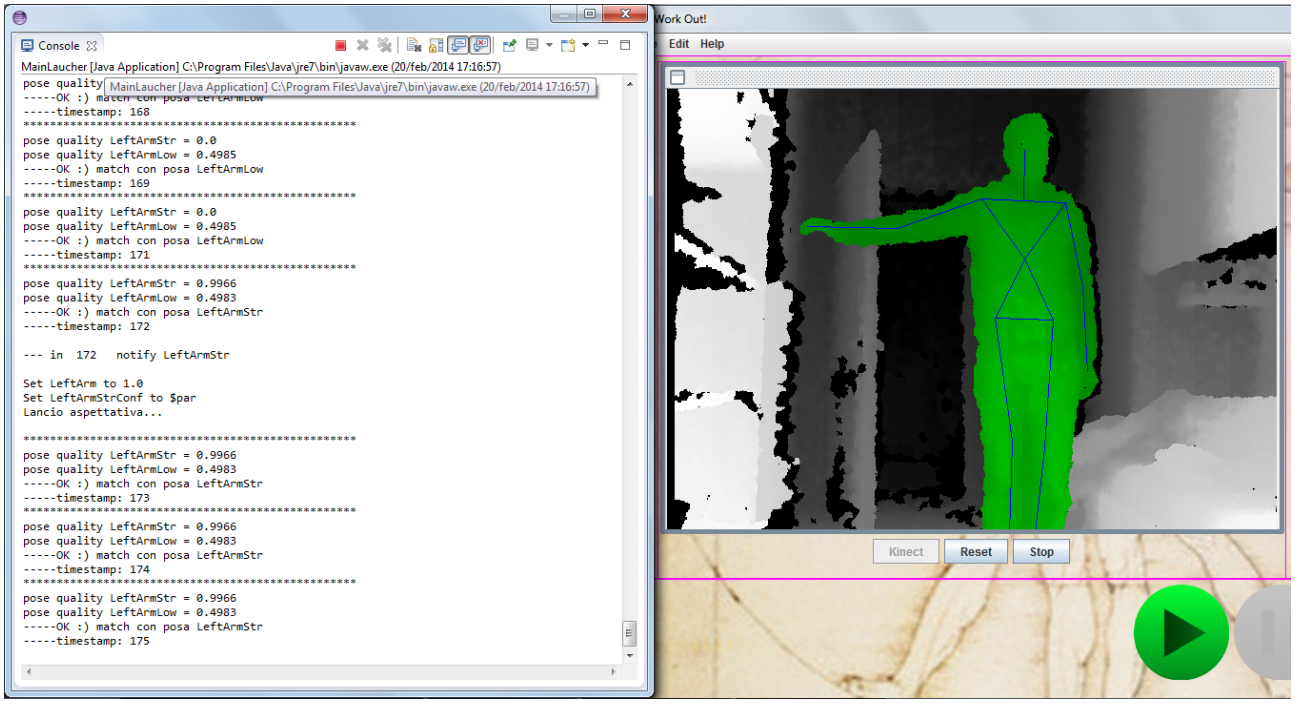


Figura 24 - Scheletro ricostruito e output testuale

A questo punto, mano a mano che le pose vengono riconosciute, vengono lanciati degli eventi che vanno ad interagire con il motore del sistema generando dell'output.

Proviamo quindi ad eseguire l'esercizio che è stato definito attraverso l'editor: braccio alzato a "T" e successivamente, entro 10 secondi, braccio abbassato.

Osserviamo il comportamento in caso di esercizio completato correttamente, analizzando l'output di *Figura 27*.
All'istante iniziale 4 viene riconosciuto il braccio alzato, il valore di confidenza per braccio alzato viene settato al valore passato come parametro dell'evento e viene lanciata un' aspettativa, ossia *è atteso un evento braccio abbassato entro 10 secondi.*
Dopo due secondi, all'istante 6, viene riconosciuto il braccio abbassato, il valore di confidenza per braccio abbassato viene settato al valore del parametro dell'evento e *l'aspettativa risulta rispettata*. Il fluente Score viene settato al prodotto delle confidenze salvate nei fluenti.

Ipotizziamo invece il comportamento in caso di esercizio non completato correttamente, analizzando l'output in *Figura 28*.
All'istante 4 viene riconosciuto il braccio alzato e *viene lanciata l'aspettativa* come nel caso precedente. Il braccio abbassato non viene però riconosciuto prima di 13 secondi e quindi *l'aspettativa in questo caso è violata*.

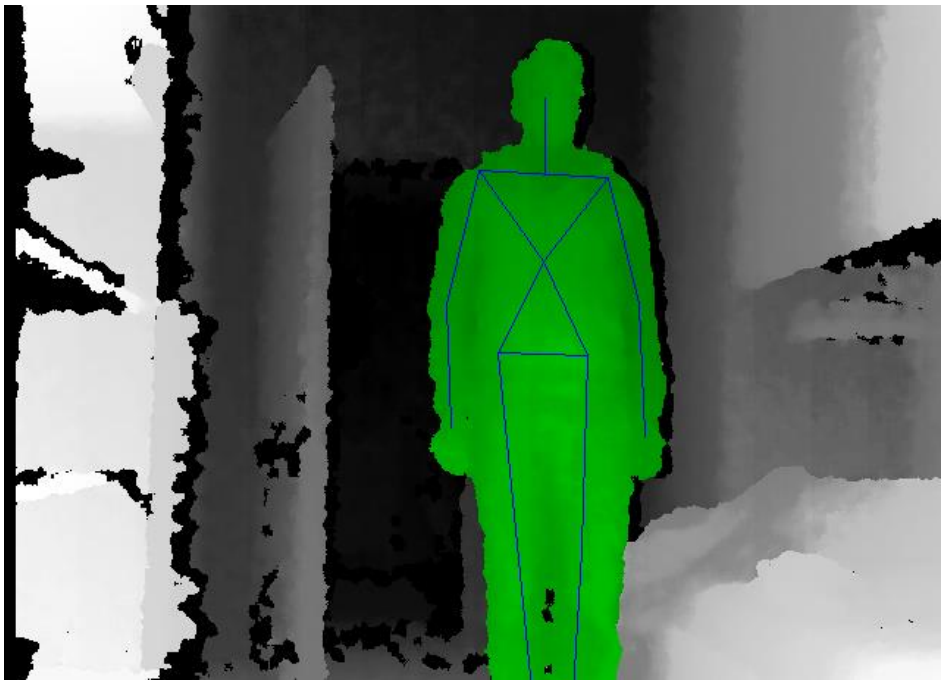


Figura 25 – scheletro prima posa

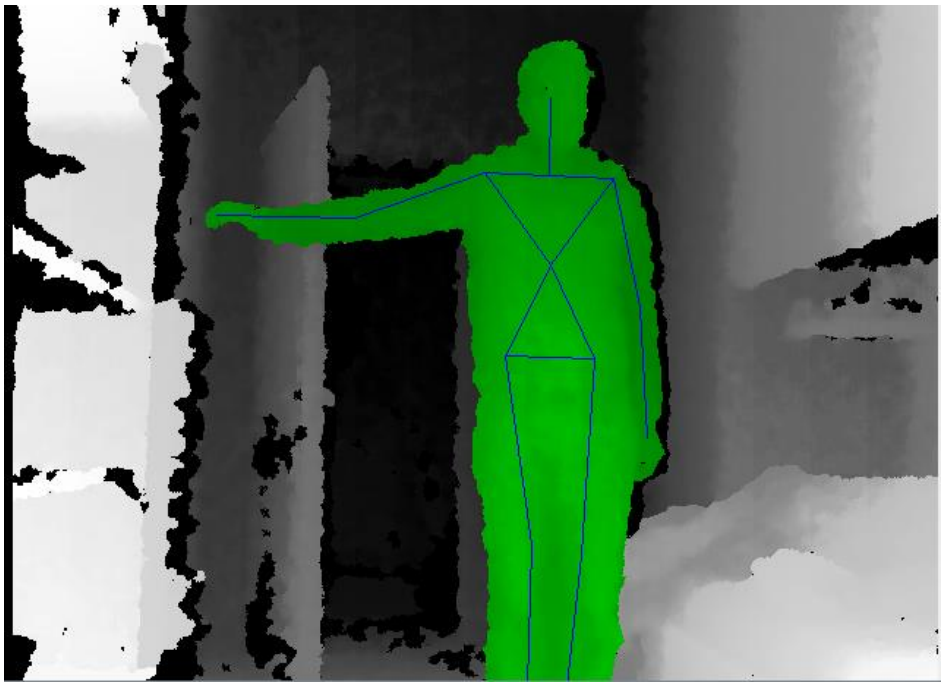


Figura 26 – scheletro seconda posa

```
device USB\VID_0409&PID_005A\5&39FC87BE&0&1 OPENED!

**********************************************              //braccio abbassato
pose quality LeftArmStr = 0.0
pose quality LeftArmLow = 0.9984
-----OK :) match con posa LeftArmLow
-----timestamp: 1

--- in  1   notify LeftArmLow

Set LeftArm to 0.0
Set LeftArmLowConf to $par
**********************************************
pose quality LeftArmStr = 0.0
pose quality LeftArmLow = 0.9985
-----OK :) match con posa LeftArmLow
-----timestamp: 2
**********************************************
pose quality LeftArmStr = 0.0
pose quality LeftArmLow = 0.9985
-----OK :) match con posa LeftArmLow
-----timestamp: 3
**********************************************              //braccio alzato
pose quality LeftArmStr = 0.9966                            //entro 10 secondi mi aspetto un
pose quality LeftArmLow = 0.0                               //braccio abbassato
-----OK :) match con posa LeftArmStr
-----timestamp: 4

--- in  4   notify LeftArmStr

Set LeftArm to 1.0
Set LeftArmStrConf to $par   //fluente settato a 0.9966
Lancio aspettativa...        //lancio un'aspettativa
**********************************************
pose quality LeftArmStr = 0.9968
pose quality LeftArmLow = 0.0
-----OK :) match con posa LeftArmStr
-----timestamp: 5
**********************************************              //OK! dopo 2 secondi il braccio
pose quality LeftArmStr = 0.0                               //è abbassato
pose quality LeftArmLow = 0.9985
-----OK :) match con posa LeftArmLow
-----timestamp: 6

--- in  6   notify LeftArmLow

Set LeftArm to 0.0
Set LeftArmLowConf to $par  //fluente settato a 0.9985
Expectation FULFILLED       //aspettativa rispettata
Set Score to fluent/parameter value (or their composition) //setto lo Score(prodotto
                                                           //delle due confidence
**********************************************         // ==0.9985*0.9966
pose quality LeftArmStr = 0.0
pose quality LeftArmLow = 0.9985
-----OK :) match con posa LeftArmLow
-----timestamp: 7
**********************************************
pose quality LeftArmStr = 0.0
pose quality LeftArmLow = 0.9986
-----OK :) match con posa LeftArmLow
-----timestamp: 8
```

Figura 27 - Output se aspettativa verificata

```
device USB\VID_0409&PID_005A\5&39FC87BE&0&1 OPENED!

**********************************************              //braccio abbassato
pose quality LeftArmStr = 0.0
pose quality LeftArmLow = 0.4985
-----OK :) match con posa LeftArmLow
-----timestamp: 1

--- in  1   notify LeftArmLow

Set LeftArm to 0.0
Set LeftArmLowConf to $par
**********************************************
pose quality LeftArmStr = 0.0
pose quality LeftArmLow = 0.4985
-----OK :) match con posa LeftArmLow
-----timestamp: 2
**********************************************
pose quality LeftArmStr = 0.0
pose quality LeftArmLow = 0.4985
-----OK :) match con posa LeftArmLow
-----timestamp: 3
**********************************************              //braccio alzato, mi aspetto
pose quality LeftArmStr = 0.9972                            //braccio abbassato entro 10 sec
pose quality LeftArmLow = 0.4994
-----OK :) match con posa LeftArmStr
-----timestamp: 4

--- in  4  notify LeftArmStr

Set LeftArm to 1.0
Set LeftArmStrConf to $par
Lancio aspettativa...
**********************************************
pose quality LeftArmStr = 0.9968
pose quality LeftArmLow = 0.4984
-----OK :) match con posa LeftArmStr
-----timestamp: 5
**********************************************

. . . . . .


**********************************************              //dopo 13 secondi arriva braccio
pose quality LeftArmStr = 0.0                               //abbassato...troppo tardi.. ☹
pose quality LeftArmLow = 0.4982
-----OK :) match con posa LeftArmLow
-----timestamp: 17

--- in  17   notify LeftArmLow

Set LeftArm to 0.0
Set LeftArmLowConf to $par
Expectation VIOLATED      //aspettativa violata
```

Figura 28 - Output se aspettativa violata

In questo capitolo si è definito il plug-in per il riconoscimento dell'attività umana in ambito fisioterapico.

Come tipo di sensore si è scelto di utilizzare Microsoft Kinect per andare a rilevare i movimenti del soggetto.
Inoltre è stata utilizzata la libreria Weka e gli algoritmi basati sulle Macchine a Supporto di Vettori per garantire la classificazione delle pose riconosciute grazie a un sistema ad apprendimento supervisionato.

Infine si è passati alla presentazione di un caso esemplificativo semplice ma significativo che permetta di apprezzare le potenzialità dello strumento realizzato nel suo insieme.

Conclusioni e sviluppi futuri

La riabilitazione fisica, oltre ad essere utile per il paziente, in quanto gli permette di riacquistare la condizione fisica che aveva prima di un incidente o di una patologia, porta con se una serie di svantaggi più o meno gravi che risultano evidenti quando ad aver bisogno di cure sono le persone anziane.

Gli individui di una certa età sono spesso soggetti a cadute ed infortuni e dopo uno stato di relativa immobilità forzata o di una operazione potrebbero necessitate di cure specialistiche. Generalmente il paziente viene indirizzato verso l'ASL di competenza dove inizierà il percorso di riabilitazione. Spesso il fisioterapista, allo scopo di velocizzare e migliorare il processo di guarigione, potrebbe voler suggerire una serie di esercizi che dovranno essere svolti in autonomia da casa, in assenza quindi dello specialista che può aiutare e correggere il soggetto durante la fase di esecuzione. Qui nascono una serie di problemi: la persona potrebbe non svolgere gli esercizi che gli sono stati indicati, a causa di un dolore residuo per esempio, eseguirli non con la frequenza sufficiente o addirittura svolgerli in maniera scorretta impedendo il corretto riacquisto delle capacità motorie ed in alcuni casi facendo insorgere altre patologie. Inoltre il fisioterapista avrà bisogno di visitare con una certa frequenza il paziente per controllare i progressi raggiunti, con quindi tutta una serie di inconvenienti dovuti alla necessita di doversi incontrare ogni volta presso la stessa struttura.

Questo lavoro di tesi si è concentrato sulla soluzione di questo tipo di problematiche andando ad agire sugli elementi di base che sono causa delle inefficienze descritte sopra.

Abbiamo voluto sviluppare uno strumento che consentisse allo specialista di definire gli esercizi che vuole siano svolti, e un sistema che permettesse al paziente di svolgerli in autonomia, ricevendo un feedback costante sulla corretta esecuzione degli stessi. Volevamo

inoltre che il medico avesse la possibilità di monitorare la buona esecuzione del lavoro assegnato al malato, potendo andare a modificare se necessario le pose.

Avevamo quindi bisogno di tre componenti fondamentali:

- Un linguaggio, sufficientemente potente da poter rappresentare tutti gli esercizi che lo specialista vuole assegnare
- Un editor, ossia uno strumento che, assistendo il fisioterapista, utilizzando il linguaggio, permetta di definire gli esercizi nel dettaglio
- Un motore, ovvero una tecnologie che riesca a valutare gli esercizi svolti e li vada a confrontare con quelli proposti dallo specialista.

Inoltre avevamo bisogno di un qualche tipo di sensore che permettesse di monitorare i movimenti del paziente, in modo da poter fornire i dati al motore.

In questo lavoro di tesi abbiamo iniziato con l'analizzare il settore della fisioterapia e con il capire quali fossero le problematiche e le esigenze del settore. Sono successivamente state descritte le tecnologie che ad oggi risultano utili ai nostri scopi: In particolare abbiamo analizzato strumenti per la realizzazione del linguaggio e dell'editor, del motore, ed è stato esplorato il campo di sensori atti al riconoscimento dell'attività umana, i software a supporto di queste tecnologie e gli algoritmi che permettono una classificazione delle pose.

Nei capitoli successivi sono state descritte le scelte fatte per la definizione del linguaggio che consentisse di esprimere i concetti dei quali avevamo bisogno ed è stato esposto il lavoro svolto per realizzare il sistema.

Il filo conduttore in questa fase è stata la teoria del Calcolo degli Eventi, estesa però con l'introduzione del concetto di aspettativa che consentisse di rappresentare un comportamento futuro atteso e che permettesse di esprimere le conseguenze in caso di aspettativa verificata o meno.

Il linguaggio è risultato essere sufficientemente potente per poter definire esercizi base ma allo stesso tempo di facile estensione per ampliarne le capacita. E' stato descritto lo strumento per la definizione

dell'editor e sono stati enunciati i vantaggi che una tecnologia integrata in uno strumento di sviluppo già esistente porta con se.

Per quanto riguarda il motore è stato scelto Drools che consente di realizzare un sistema esperto che riesce a gestire la conoscenza acquisita tramite le frasi scritte nel linguaggio dal fisioterapista, e, utilizzando le informazioni relative alle pose assunte dal paziente, generare nuova conoscenza attuando azioni conseguenti. E' stata analizzata la fase di traduzione delle frasi in regole compatibili con il motore Drools, sempre avendo chiaro il legame stretto con la teoria del Calcolo Degli Eventi.

Infine è stato affrontato il problema di come riconoscere le pose degli esercizi. Ci siamo affidati al Kinect, tecnologie di ampia diffusione e di facile utilizzo. E' stata descritta l'apparecchiatura e il corredo software che la accompagna e che è necessario per il suo funzionamento. E' stata poi esposta la libreria Weka e l'algoritmo SVM utilizzato per la classificazione delle pose.

Infine è stata mostrata l'applicazione in funzione al lavoro su un caso semplice ma significativo dove è presente il concetto di aspettativa.

Per quanto riguarda gli sviluppi futuri, saprebbe interessante ampliare il linguaggio, potenziandolo, e integrare un'architettura distribuita per la verifica in remoto degli esercizi [40].

In questo lavoro di tesi si è inoltre posto l'accento sul fatto che le fasi e le tecnologie che riguardano il linguaggio, l'editor e il motore sono totalmente indipendenti dal tipo di applicazione considerata. Infatti l'utilizzo del sensore Kinect e il software per il riconoscimento dell'attività umana si inseriscono nel progetto come un plug-in che si aggancia al motore attraverso un'interfaccia ad eventi. Si possono comprendere quindi le potenzialità del sistema sviluppato che, se da un lato riesce a risolvere i problemi relativi all'ambito fisioterapico, dall'altra si presta ad essere utilizzato nei settori più disparati.

Ringraziamenti

Ora che sono giunto al traguardo mi guardo indietro e mi chiedo dove ho trovato le forze e l'energia per raggiungere, passo dopo passo, la meta, consapevole però che senza l'aiuto delle persone che hanno creduto in me e che sempre mi hanno supportato non sarei riuscito a raggiungere gli stessi obiettivi.

Un ringraziamento alla Prof.ssa Mello che mi ha permesso per la mia tesi di lavorare in un settore, quello dell'intelligenza artificiale, così affascinante e stimolante. Ringrazio Stefano Bragaglia che mi ha seguito durante questi mesi e che mi ha aiutato e spronato a dare il meglio sempre e a Federico Chesani che mi ha accompagnato nell'ultimo periodo per la rifinitura del lavoro.

Grazie a mamma e papà che, non senza sacrifici, mi hanno sostenuto in questo mio percorso credendo in me aiutandomi a mettere a fuoco un obiettivo dopo l'altro. Di una cosa sono sicuro: senza di voi ora non sarei qui.

Un grazie ai miei compagni di studi, sempre disponibili al confronto e agli amici Luca, Pier e Riccardo che durante il mio percorso universitario mi hanno sopportato e che con una birra in compagnia e due risate mi hanno fatto staccare di tanto in tanto la spina.

Un grazie speciale a Federica che in questi cinque anni di studi con amore e sterminata pazienza ha sempre creduto in me.

Bibliografia

[1] J. BACKUS, «The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference,» in *Proceedings of the International Conference on Information Processing. UNESCO.*, 1959, pp. 125-132.

[2] «Java Compiler Compiler,» [Online]. Available: https://javacc.java.net//.

[3] «GOLD Parsing System,» [Online]. Available: http://goldparser.org/.

[4] «ANTLR,» [Online]. Available: http://www.antlr.org/.

[5] «Xtext - Language Development made Easy!,» [Online]. Available: http://www.eclipse.org/Xtext/.

[6] «JUnit,» [Online]. Available: http://junit.org/.

[7] «NetBeans,» [Online]. Available: https://netbeans.org/.

[8] «JetBrains MPS,» [Online]. Available: http://www.jetbrains.com/mps/.

[9] M. BOARI, E. LAMMA, P. MELLO, S. STORARI e S. MONESI, «An Expert System Approach for Clinical Analysis Result Validation,» *Proceedings ICAI,* 2000.

[10] E. LAMMA, P. MELLO, A. NANETTI, G. POLI, F. RIGUZZI e S. STORARI, «An Expert System for Microbiological Data Validation and Surveillance,» *Proceedings of ISMDA,* 2001.

[11] B. BARBIERI, G. GAMBERONI, E. LAMMA, P. MELLO, P. PAVESI e S. STORARI, «An Expert System for the Oral Anticoagulation Treatment,» *Innovations in Applied Artificial Intelligence,* vol. 3533, pp. 773-782, 2005.

[12] E. FEIGENBAUM, B. BUCHANAN e G. SUTHERLAND, «HEURISTIC DENDRAL : a Program for Generating Explanatory Hypotheses in Organic Chemistry,» *Computer Science Department, Stanford University.*

[13] «Microsoft's Rule Engine Scalability Results - A comparison with Jess and Drools,» [Online]. Available: http://geekswithblogs.net/cyoung/articles/54022.aspx.

[14] «Drools,» [Online]. Available: https://www.jboss.org/drools/.

[15] C. L. FORGY, «RETE: A Fast Algorithm for the Many Patter / Many Object Match Problem,» *Artificial Intelligence,* pp. 17-37, 1982.

[16] R. KOWALSKI e M. SERGOT, «A logic-based calculus of events,» in *Foundations of knowledge base management*, Springer, 1989, pp. 23-55.

[17] R. KOWALSKY, «Legislation as logic programs,» in *Informatics and the Foundations of Legal Reasoning*, Springer, 1995, p. 325–356.

[18] M. SHANAHAN, «Robotics and the common sense informatic situation,» in *ECAI*, 684–688, 1996, p. PITMAN.

[19] M. WESKE, «Business process management: concepts, languages, architectures,» Springer, 2010.

[20] A. TEN TEIJE, S. MIKSCH e P. LUCAS, «Computer-based medical guidelines and protocols: a primer and current,» IOS Press, 2008, p. volume 139.

[21] N. CICEKLI e I. CICEKLI, «Formalizing the specification and execution of workflows using the event calculus,» in *Information Sciences*, 2006, p. 176(15):2227–2267.

[22] M. SHANAHAN, «Solving the frame problem: a mathematical investigation,» The MIT Press, 1997.

[23] R. KOWALSKY, «A logic-based approach to conflict resolution,» in *Technical report, Department of Computing, Imperial College*, London, 2003.

[24] M. SHANNON, «The Event Calculus explained,» in *Artificial intelligence TODAY*, 1999, p. 409–430.

[25] L. CHITTARO e A. MONTANARI, «Efficient Handling of Context-Dependency in the Cached Event Calculus,» in *Proceedings of TIME'94 - International Workshop on Temporal Representation and Reasoning*, 1994, p. 103–112.

[26] F. CHESANI, P. MELLO, M. MONTALI e P. TORRONI, «A logic-based, reactive calculus of events,» in *Fundamenta Informaticae*, 2010, p. 105(1):135–161.

[27] «PrimeSense,» [Online]. Available: http://www.primesense.com/.

[28] WiiRemote. [Online]. Available: https://www.nintendo.com/wii/what-is-wii/#/controls.

[29] «PlayStation Move,» [Online]. Available: http://it.playstation.com/psmove/.

[30] F. LOPERFIDO, «Uso della Kinect per la generazione di avatar 3D personalizzati,» *Università di Bologna, Corso di Studio in Scienze e tecnologie informatiche,* 2013.

[31] «Point Cloud Library (PCL),» [Online]. Available: http://pointclouds.org/.

[32] «OpenKinect,» [Online]. Available: http://openkinect.org/wiki/Main_Page.

[33] «Kinect SDK,» [Online]. Available: http://www.microsoft.com/en-us/kinectforwindowsdev/start.aspx.

[34] «Open-Source SDK for 3D sensors - OpenNI,» [Online]. Available: http://www.openni.org/.

[35] «PrimeSense NiTE,» [Online]. Available: http://www.primesense.com/wp-content/uploads/2012/10/PrimeSense-NiTE-Middleware-A4-Lo.pdf.

[36] «University of Waikato,» [Online]. Available: https://weka.waikato.ac.nz/explorer.

[37] S. DI MONTE, «Monitoraggio in spazio controllato mediante Kinect e Calcolo degli Eventi,» pp. 72-74, 2013.

[38] S. BRAGAGLIA, F. CHESANI, E. FRY, P. MELLO, M. MONTALI e D. SOTTARA, «Event condition expectation (ECE-) rules for monitoring observable systems,» *DEIS, University of Bologna,* 2011.

[39] S. DI MONTE, «Monitoraggio in spazio controllato mediante Kinect e Calcolo degli Eventi,» 2013, pp. 91-100.

[40] M. SAVOJARDO, «Architettura distribuita per l'esecuzione e verifica in remoto di esercizi fisioterapici mediante Kinect,» 2013.

Elenco delle figure

Elenco delle tabelle