



Consiglio Nazionale delle Ricerche

Istituto di Calcolo e Reti ad Alte Prestazioni

Un assistente robotico per “AMICO”

Modellazione di uno scenario di assistenza verbale

domestico-sanitaria per il Progetto

“Assistenza Medica In COntextual awareness”

Prototipi di componenti software

ed ambiente di simulazione

A. Machì,

Primo Ricercatore ICAR-Palermo

RT-ICAR-PA-19-01

Data ultimo aggiornamento

20 Settembre 2019



Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) –

Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: www.icar.cnr.it

– Sede di Napoli, Via P. Castellino 111, 80131 Napoli, URL: www.na.icar.cnr.it

- Sede di Palermo, Viale U. La Malfa 153, 90146 Palermo, URL: www.pa.icar.cnr.it

Sommario

Un assistente robotico per “AMICO” Modellazione di uno scenario di assistenza verbale domestico-sanitaria per il Progetto “Assistenza Medica In COntextual awareness”	1
Prototipi di componenti software ed ambiente di simulazione	1
1 Premessa	3
2 Il Progetto Amico	3
3 Scenario di riferimento	3
4 Analisi dei requisiti	5
4.1 Vincoli	5
4.2 Modalità di conduzione dell’interazione	5
5 Modellazione di attori e skills. Casi d’uso e Funzionalità	6
5.1 Attori e ruoli :.....	6
5.1.1 Servizi e skills	6
5.2 Casi d’uso e funzionalità	7
5.2.1 Casi d’uso	7
5.2.2 Funzionalità.....	7
6 Modellazione della conoscenza di dominio ed operativa	9
7 Software Sviluppato	11
7.1 Ambiente di simulazione ad eventi basato su socket.io.	11
7.2 Componenti dell’ambiente di simulazione.	13
7.2.1 Componenti esecutive (server e servizi)	13
7.2.2 Servizi Watson Dialog Assistant	19
7.3 ws2naoQi(bridge).....	24
7.4 actionLibServer (controller).....	24
7.5 Componenti grafiche.....	25
8 Policies	33
8.1.1 User profiling.....	33
8.1.2 Preemptive priority scheduling.....	33
8.1.3 Robot attitude.....	34
8.1.4 Animated speech	35
9 Dettagli implementativi	38
9.1 Servizi Watson	38
9.1.1 Skill HomeAssistant	38
9.1.2 Skill HealthAssistant2	38
9.2 Sintassi dei comandi	39
9.3 Eventi del dialogo	40
9.4 IDE, inizializzazioni operative, bundle del software,bugs	41
9.4.1 Ambienti di sviluppo	41
9.4.2 Bugs.....	41
9.4.3 Software Bundle	42
10 Stato Rapporto Tecnico	53
11 Ringraziamenti	58
12 BIBLIOGRAFIA	59

1 Premessa

Il presente rapporto tecnico descrive lo stato dell'arte, alla data di copertina, di una attività di modellazione della conoscenza e di modellazione e simulazione software propedeutica allo sviluppo del prototipo di un sistema assistivo robotico per il Progetto "Assistenza Medica In COntextual awareness" [1]

Senza pretesa di completezza, viene descritto lo stato di avanzamento della modellazione, e dello sviluppo delle componenti realizzate. Sono indicati i limiti riscontrati ed indicati possibili attività di correzione o espansione.

2 Il Progetto Amico

Il Progetto di Ricerca Industriale e non preponderante Sviluppo Sperimentale, area di specializzazione "Tecnologie per gli Ambienti di Vita", dal titolo "Assistenza Medica In COntextual awareness" [2] prevede all'obiettivo realizzativo OR3 la Progettazione e realizzazione dei servizi di base, di analisi, decisione e comunicazione relativa al monitoraggio della terapia di riabilitazione cardiaca post operatoria domestica.

Nell'ambito di tale obiettivo viene sperimentato l'utilizzo di un intrattenitore robotico con fattezze umanoidi quale intermediario della interazione sanitaria col paziente con funzioni di "*motivatore*" all'aderenza dei comportamenti del paziente al protocollo sanitario riabilitativo.

3 Scenario di riferimento

In attesa della definizione dettagliata, delle esatte funzionalità richieste per il monitoraggio della terapia cardiaca post operatoria domestica, caso di studio clinico target del progetto, la modellazione prende in considerazione uno scenario generico di interazione verbale fra un robot domestico ed un paziente in terapia domestica.

Ciò al fine di modellare e prototipare un insieme di processi controllori e di servizi di supporto generico a sessioni di dialogo complesse, con mantenimento del contesto semantico, e di sperimentare design e behavioural patterns per la espressione nel dialogo, da parte dell'assistente robotico, di comportamenti antropomorfici quali autocoscienza, esplicazione delle motivazioni ed atteggiamenti affettivi di empatia.

Riferendosi al modello astratto di programmazione OSI (Open Standard Interconnection) [3] lo studio si focalizza sulla definizione di archetipi di processi controllori, di protocolli di cooperazione, di policies del livello OSI 5 della applicazione, responsabili del mantenimento della coerenza semantica ed operativa a livello di sessione di scambio di messaggi robot-paziente.

Per la gestione dell'infrastruttura di comunicazione a messaggi/eventi a livello OSI 4 si presuppone l'utilizzo di framework e servizi forniti da framework quali webSocket [4] ed i meccanismi di livello OSI4 di ROS (Robot Operating System).[5]

Nello scenario considerato, il robot assistente ha la funzione principale di intrattenere e motivare il paziente a mantenere il suo comportamento puntualmente compliant con le prescrizioni del protocollo terapeutico, mentre il monitoraggio di parametri fisiologici è affidato ad una infrastruttura di sensori indossabili dal paziente. Il robot quindi opera da assistente personale e domotico per la maggior parte del tempo. La funzione di monitoraggio dello stato di salute è al di fuori delle sue funzioni e del suo livello di coscienza, se non per la esecuzione di task sanitari verbali o mimico specificatamente previsti nel piano terapeutico e per l'attivazione di opportuni alert.

Sulla base di un piano terapeutico dinamicamente modificabile (livello OSI 7) il robot attiva quindi conoscenze e funzioni sanitarie previste dal protocollo terapeutico (notifiche, solleciti, interviste, log di espressioni verbali e di mood), sia in base alla temporizzazione prevista dalla pianificazione terapeutica, che a seguito di richiesta di assistenza espressa verbalmente (o attraverso atteggiamento espressivo posturale) da parte del paziente.

Nel dialogo l'assistente esprime autocoscienza del suo stato operativo interno, della situazione ambientale ed assume un atteggiamento emotivo variabile nei confronti dell'interlocutore in funzione della policy prevista di accondiscendenza empatica col paziente (es. in caso di ragionevole aderenza dello stesso al protocollo o dell'aggravarsi del suo livello di ansia) o di stimolo deciso (es. in caso di cattiva aderenza al protocollo o di criticità dello stato di salute)..

Le funzioni di assistente domotico considerate nel presente studio sono ispirate alle funzioni di assistente personale di AMY-robot [6] . Le **funzionalità** di assistente sanitario sono ispirate a quelle del Robot assistente MAX del Progetto SERROGA [7] (con esclusione delle funzioni di navigazione attiva) e riguardano *l'intrattenimento, la ricerca e fornitura di informazioni, la attivazione di controlli ambientali e di servizi telematici, il monitoraggio di parametri biomedici, il monitoraggio dello stile di vita, la somministrazione di questionari, l'assistenza all'esecuzione di attività in agenda.*

L'interazione avviene sia attraverso sessioni di scambio esplicito (end-to-end utente-robot), di frasi in linguaggio naturale (italiano) che attraverso lo scambio con e fra i processi di controllo, di messaggi di gestione. I messaggi di gestione utilizzano termini di un vocabolario controllato ed una sintassi a frasi stereotipate da linguaggio di comando.

Durante il dialogo il paziente e l'assistente negoziano il controllo della sessione. Una sessione di intrattenimento può essere interrotta per l'esecuzione di un task sanitario indifferibile, ed eventuali notifiche di sistema sono posposte alla fine di una sessione di dialogo.

4 Analisi dei requisiti

4.1 Vincoli

Per semplificare lo scenario vengono imposti i seguenti vincoli alle possibili interazioni:

- La sessione di dialogo è attiva quando l'interlocutore ed il robot sono in vista diretta ed hanno concordato di parlarsi (engagement). L'interruzione temporanea dell'engagement attiva una sospensione della sessione con conservazione dello stato semantico del discorso, e l'attivazione di una procedura di richiamo per il ripristino della sessione. La ripetuta mancanza di re-engagement provoca la fine prematura della sessione.
- Nella semplificazione iniziale il dialogo è limitato a *sessioni* di scambio di *messaggi* verbali fra UNA entità sorgente alla volta (*paziente, robot, o agente/servizio esterno*) ed UNA entità destinataria alla volta (*paziente, robot, agente o servizio esterno*). E' attiva una sola sessione di comunicazione alla volta.
- Perché il dialogo sia possibile è necessario che non ci sia ambiguità sulla sorgente e sulla destinazione dei messaggi.

4.2 Modalità di conduzione dell'interazione

- In funzione dello scenario la sessione può essere attivata dal robot (*Monitoraggio, Terapia*) o dall'entità esterna (*Intrattenimento, Richiesta informazioni*), o indifferentemente da uno dei due (*Promemoria*).
- La comunicazione del paziente viene interpretata e disambiguata anche richiedendo chiarimenti o completamenti della espressione verbale utilizzata.
- La risposta ad un messaggio umano può essere attesa immediatamente (*Intrattenimento*) o differita a dopo il recupero delle informazioni pertinenti (*Query*), od il completamento della dell'azione richiesta (*Task*) mantenendo nel frattempo attivo l'intrattenimento.
- Durante le esecuzioni di procedure sanitarie di checkup, sono ammesse solo divagazioni rivolte a chiarire il significato dei termini in uso.
- I task sanitari facenti parte del protocollo di cura hanno la precedenza sui task richiesti dal paziente.
- Stile verbale e contenuti possono essere adattati al profilo dell'utente ed al suo stato emotivo o di salute modificando sia vocabolario utilizzato che il livello di proattività e di emotività espressa verso il paziente.
- L'atteggiamento del robot nei confronti del paziente può variare da un coinvolgimento emotivo di simpatia (SYMPATIC) e permissività ad un atteggiamento di neutralità (EMPATIC), ad un atteggiamento di eventuale disapprovazione e deciso sollecito di attenersi al protocollo terapeutico (IMPERATIVE).

5 Modellazione di attori e skills. Casi d'uso e Funzionalità

5.1 Attori e ruoli :

Nello scenario operano alcuni attori, fra i quali, vengono considerati e modellati:

- **Paziente:** si intrattiene con l'assistente, gli chiede sia informazioni che l'esecuzione di task immediati (con sospensione del dialogo di intrattenimento) quali "ballare" o di task differiti (con notifica finale asincrona) quali "andare ad accendere la luce in bagno". I task immediati richiedono il mantenimento dell'engagement, che si perde invece durante la esecuzione di quelli differiti, per ripristinarsi al concludersi degli stessi, quando si ritorna alla modalità intrattenimento. Il paziente assume il ruolo di interlocutore o generatore di task durante il dialogo e di intervistato durante l'esecuzione dei test sanitari.
- **Controllore dell'assistente di dialogo:** Intermedia la comunicazione del paziente con gli assistenti di dialogo robotico (intrattenitore domestico generico e specialista sanitario) e controlla il flusso della sessione di dialogo. Di norma dirige la comunicazione sull'assistente domestico. L'assistente sanitario è attivato da richieste di task sanitari e mantiene la sessione come non interrompibile da richieste utente.
- **Gestore delle attività del robot:** coordina i task di attività propria del robot, quelli di interazione e quelli di esecuzione di attività richieste dal paziente o dal protocollo sanitario. Mantiene una coda di task utente e sanitari e ne manda in esecuzione uno alla volta. I task utente sono sospendibili da parte di task sanitari temporizzati.
- **Gestore del piano sanitario**
- **Monitor** delle risorse di sistema e dello stato generale

5.1.1 Servizi e skills

Sono considerati disponibili agli attori due differenti servizi Assistenti di Dialogo Watson dedicati alla analisi del parlato del paziente (NLP) uno con dominio di conoscenze generico (conversazione) ed uno specialistico (interpretazione dei termini sanitari e conduzione di interviste).

- **Assistente di dialogo domestico** per: intrattenimento, ricerca di informazioni su DB, notifica di messaggi, richiesta di attivazione task di controllo ambientale o esecuzione di performance
- **Assistente di dialogo sanitario** per : motivazione del paziente all'aderenza al protocollo di recupero, checkup periodico, notifica di scadenze, messaggi , registrazione di dati sanitari su DB.

E' considerato disponibile, inoltre, un servizio di gestione dei dati della conoscenza di dominio, es. in forma di DB (lettura di informazioni sanitarie, lettura/aggiornamento della cartella clinica del paziente).

5.2 Casi d'uso e funzionalità

5.2.1 Casi d'uso

I casi d'uso individuati corrispondono ai vari momenti/modalità con cui si effettua la interazione paziente-robot:

- *_CHAT : intrattenimento attivato dal robot o dal paziente e supportato dal controllore dall'assistente di dialogo (servizio assistente domestico)*
- *_MOTIVATION: motivazione all'aderenza al protocollo sanitario durante la esecuzione dei task sanitari programmati dal gestore del piano sanitario, attivati dal gestore delle attività, ed eseguiti dal controllore di dialogo)*
- *_ASK_FOR: richiesta di attività di servizio da parte del paziente, eseguita (assistente domestico)*
- *_EXECUTE esecuzione di task utente o di sistema (richiesti dal paziente ed eseguiti dal sistema da parte del controllore di dialogo o richiesti dal controllore dell'attività del robot ed eseguiti dal controllore di dialogo)*
- *_DIALOG_CONTROL: controllo di flusso del dialogo (assistente domestico)*

5.2.2 Funzionalità

In attesa della definizione dettagliata, delle esatte funzionalità richieste per il monitoraggio della terapia cardiaca post operatoria domestica, caso di studio clinico target del progetto, la modellazione prende in considerazione un insieme minimale di funzionalità ipotizzabili in base alla esperienza di laboratorio o a funzionalità previste in altri progetti di assistenza medica similari ad AMICO.

Task di gestione

- *Startup e raggiungimento dalla postazione (ad attivazione il robot lascia la postazione di carica e si nella postazione base, da dove cerca il contatto col paziente)*
- *Ritorno alla dockstation (a fine attività, o a fine carica, il robot si porta nella postazione di carica e si pone in uno stato di consumo minimale)*
- *Engagement (all'entrata del paziente nel campo di vista, o a seguito di richiamo, il robot sollecita il paziente ad un contatto visuale diretto ed all'inizio di una sessione di dialogo).*
- *Reset del dialogo (al verificarsi di un errore non recuperabile, non fatale per la attività, quale una incongruenza nella interpretazione della semantica del dialogo) si effettua un reset della conversazione riportando lo stato del contesto semantico ai valori di default e riattivando il caso d'uso _CHAT*

- *Notifica (al verificarsi di una transizione nello stato di esecuzione (es attivazione o fine di un task autonomo differito) il messaggio di aggiornamento asincrono è gestito come una notifica. La notifica è immediata durante il caso _CHAT o differita nel caso _EXECUTE*

Task utente (domestici)

- *Convenevoli, gestione di frasi inopportune*
- *Recupero informazioni dal db (es. cos'è (sintomo), a che serve (farmaco))*
- *Richiesta di performance sincrona (es. ballare)*
- *Richiesta di attività asincrona autonoma quale "attivare qualcosa in qualche luogo" (es. "accendere la luce in bagno").*

Task di sistema (sanitari)

- *Reminder (lista attività giornaliere)*
- *Solicit (sollecito attività prevista dal protocollo, es. peso)*
- *Questionario (es. Checkup sintomi e log sul db)*
- *Test Checkup (checkup breve estemporaneo a seguito di espressioni del paziente che suggeriscono la presenza di sintomi da monitorare)*
- *Alert orario di esecuzione attività terapeutica (es. assumere un farmaco)*
- *Registrazione su FSE dati sanitari*
- *Check mood (dialogo stimolante espressione di stati d'animo)*

6 Modellazione della conoscenza di dominio ed operativa

La modellazione dello scenario prevede la modellazione di conoscenza relativa a:

- *Composizione e stato delle componenti HW/Sw del sistema (nome, parametri di configurazione ed accesso, stato)*
- *Servizi (nome, parametri di configurazione ed accesso ,stato)*
- *Attività (identificatori e proprietà dei task utente e di sistema)*
- *Ambiente fisico (luoghi, oggetti, persone, variabili ambientali)*
- *Semantica del dialogo (caso d'uso, soggetto, predicato, oggetto, luogo, animazione, task utente, task di sistema)*
- *Modalità di interazione (attitudine del robot, mood del paziente, engagement)*
- *Conoscenza del dominio terapeutico (sintomi, farmaci, patologie ...)*

Nel loro insieme le entità e le proprietà degli elementi di conoscenza costituiscono una ontologia non necessariamente formalizzata, ma almeno espressa da un vocabolario controllato di termini, laddove i termini sono oggetto di riconoscimento e decisione e non solo di espressione verbale nel dialogo.

Sulla base della modellazione proposta, l'aggiornamento dei fatti di conoscenza è attribuito alle varie componenti del sistema secondo lo schema riportato nella seguente tabella:

Classe	Gestore	Interfaccia di Lettura/Notifica	Aggregatore
Componenti Sistema <i>dialogManager Interactor planner</i>	<i>singola Componente manager</i>	<i>http REST API dei managers /getStatus (POST)</i>	<i>Monitor</i>
Servizi e dialogo <i>homeAssistant, healthAssistant mongoDBserver</i>	<i>dialog Manager</i>	<i>dialogManager http REST API /getStatus</i>	<i>Monitor</i>
Task	<i>planner/scheduler</i>	<i>planner http REST AP /getStatus</i>	<i>Monitor</i>
Ambiente fisico	<i>Non assegnato</i>	<i>Non definito</i>	<i>Monitor</i>
Semantica del dialogo	<i>dialogManager</i>	<i>dialogManager http REST API /getStatus</i>	<i>Monitor</i>

Modalità interazione	<i>interactor</i>	<i>interactor http REST API /getStatus</i>	<i>Monitor</i>
Conoscenza dominio terapeutico	<i>DB manager & Planner</i>	<i>DB API</i>	<i>Health Manager</i>

E' responsabilità della componente Monitor la raccolta periodica ed il mantenimento di una espressione strutturata (es serializzazione json o base di conoscenza) dei fatti relativi alle sezioni 1-6, mentre la conoscenza del dominio terapeutico è più opportunamente mantenibile in una base di dati (es. Mongo DB [10]) dalla componente HealthManager.



7 Software Sviluppato

Nell'affrontare la prototipizzazione di compone Approccio *breadth-first*, preferendo lo sviluppo prototipale di scheletri minimali di tutti i moduli esecutivi die controllori degli attori, dei servizi e delle interfacce grafiche di controllo, piuttosto che la realizzazione dettagliata di alcune di esse.

Si è provato l'utilizzo di framework diversi (socket.io , e ROS) , di linguaggi diversi (javascript e python) e meccanismi diversi di intercomunicazione e sincronizzazione (webSocket, ros-topics, http REST).

Parallelamente allo sviluppo delle componenti e dei servizi esecutivi si sono sviluppati componenti di interazione e monitoraggio di una applicazione web (dashboard) di simulazione e coordinamento supervisionato della applicazione.

L'attuale realizzazione del software è tesa alla verifica della integrazione di differenti tecnologie e servizi esistenti in una piattaforma di agenti (debolmente) accoppiati attraverso scambi di messaggi ed utilizzando servizi esterni. Gli agenti costituiscono un nucleo di sistema operativo di gestione della attenzione, della autocoscienza e della operatività del robot.

Scarsa attenzione è stata posta ad esigenze efficienza, risposta in real-time e consumo energetico.

Proattività ed intelligenza sono distribuite fra le vari componenti in forma di auto-adattività al contesto espresso nelle variabili di stato.

Una moderata attenzione è posta al riconoscimento ed alla gestione di errori di sistema e logici , inglobati nella logica di gestione del dialogo (Watson), nello skeleton dei processi di sistema.

7.1 Ambiente di simulazione ad eventi basato su socket.io.

L'analisi della interazione nella modellazione dello scenario sopra esposta indica la compresenza nello scenario di processi sincroni ed asincroni concorrenti, la potenziale sovrapposizione di flussi di comunicazione fra paziente e assistente robotico e della attività dialogica ed esecutiva, nonché la necessità di monitoraggio delle condizioni di inizio e fine dei vari task e dell'engagement fra paziente e robot. Ulteriore livello di concorrenza nasce dalla espressività multimodale (parlato e linguaggio gestuale).

Per facilitare la programmazione di tali comportamenti le è pensato di sviluppare un prototipo di ambiente di simulazione dei processi di controllo e di visualizzazione delle degli eventi e delle variabili di stato atte a descrivere il sistema di interazione nei suoi aspetti sistemici e comunicativi nonché la semantica del dialogo.

A tal fine si è scelto di far ricorso a tecnologie web per la realizzazione uniforme di processi , porte di comunicazione ed interfacce web, ed in particolare al frame-work "socket.io" [8],

sviluppato per applicazioni web real-time pilotate ad eventi e sviluppate in linguaggio javascript.

Il frame-work implementa nativamente un sistema di scambio di messaggi su canali web-socket, con sessione stabilmente mantenuta aperta. Permette la tramite sottoscrizione e notifica di eventi nominati, e dichiarazione ed esecuzione asincrona di call-back ad evento ricevuto, nonché la gestione di errori.

Una serie di librerie per mettono la realizzazione di server e client su http ed altri protocolli .

L'interfacciamento con processi, già pre-sviluppati sul robot in ambiente ROS è realizzato attraverso un bridge websoscket

Le componenti fanno uso di servizi esterni quali un gestore di DB no-sql MongoDB ed assistenti di dialogo IBM.

Un classificatore neurale analizza la frase e riconosce la presenza di termini associabili all'intenzione della frase (Intents) o alla categoria di soggetti, oggetti o proprietà citate (Entities).

Al messaggio è quindi applicato un workflow di processamento a regole.

Il messaggio in input è analizzato applicando una sequenza di gruppi di regole (if then else). Se viene riconosciuta la condizione di ingresso (Intent e/o Entity) il gruppo di regole è attivato ed applicato.

Ogni regola può assegnare valori a variabili logiche e consumare il messaggio, fornire una risposta e rimandare all'attesa di nuovo input o passarlo ad una lista di regole subordinate da verificare in sequenza fino a trovarne una per cui è valida la condizione di accensione.

Ogni regola può consumare il messaggio e rimandare a nuovo input, o , a seguito del riconoscimento di una combinazione logica di Intents, Entities e valori di variabili logiche, rimandare a regole subordinate ovvero saltare esplicitamente ad altro gruppo di regole.

E' possibile far riconoscere messaggi strutturati, attraverso la definizione di slots (proprietà). Una funzionalità interna all'analizzatore linguistico permette di definire richieste di completamento dei valori delle proprietà definite, se originariamente non presenti nel messaggio di input formulato dall'umano.

7.2 Componenti dell'ambiente di simulazione.

Per l'ambiente sono stati sviluppati gli stereotipi di differenti tipi di processo che realizzano in maniera basilare le funzionalità dei principali attori software modellati nello scenario, GUI che permettono il monitoraggio ed il debug delle componenti software e l'interazione web con le stesse, una interfacci grafica integrata, denominata dashboard e delle applicazioni di servizio di utilità

7.2.1 Componenti esecutive (server e servizi)

La Fig. 1 mostra le componenti esecutive della piattaforma. Segue una descrizione sintetica della realizzazione. In Appendice i dettagli.

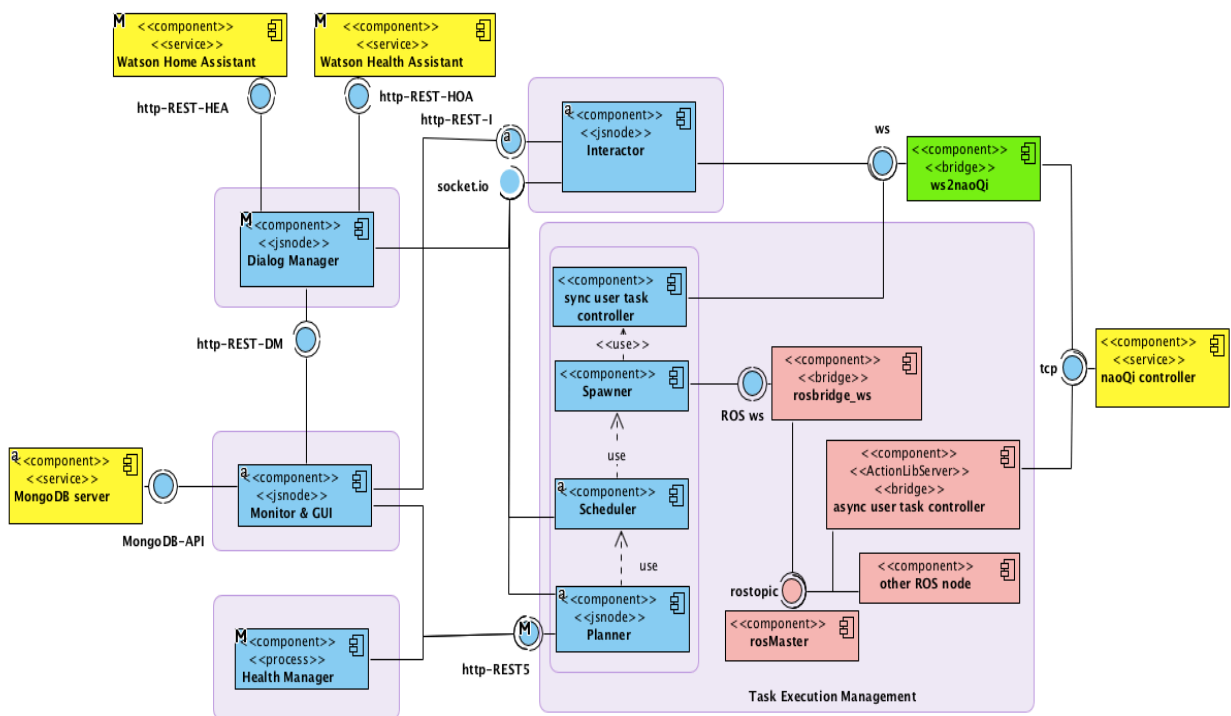


Fig. 1 Componenti esecutivi dell'ambiente di simulazione


Dialog Manager

Descrizione	Intermedia la comunicazione fra <i>interactor</i> ed i servizi assistenti di dialogo robotico (intrattenitore domestico generico e specialista sanitario). Di norma dirige la comunicazione sull'assistente domestico. L'assistente sanitario è attivato al riconoscimento di richieste di task sanitari . Estrae dai metadati delle risposte fornita degli assistenti, la descrizione della semantica del discorso e la trasmette all' <i>interactor</i> come metadati dei messaggi di risposta.
Realizzazione	Applicazione server <i>nodejs</i> . event-driven. Server socket.io per <i>monitor(GUI),interactor(GUI),planner(GUI)</i> , Client del bridge <i>websocket</i> verso <i>naoQi</i> Server http per <i>monitorGUI,interactorGUI,plannerGUI</i> Server <i>http REST</i> per <i>monitor(GUI)</i>
TO DO	Gestione del controllo di flusso sul dialogo per permettere la sospensione ed il ripristino del contesto semantico del dialogo di intrattenimento in caso di sopravvenienza di task sanitario o altri casi di digressione.
Note	I messaggi trasferiti contengono sia dati (testo di domanda/risposta o comando/notifica), che metadati (situazione del dialogo dal punto di vista watson che dal punto di vista della piattaforma AMICO . il manager il mantenimento della sezione stato dei servizi di cui è cliente e la semantica del dialogo. Le richieste di task utente del paziente ed i comandi di sistema del planner sono codificati in stringhe di comando

Health Manager

Descrizione	Gestore dinamico del piano sanitario. Definisce le procedure giornaliere del protocollo sanitario, e la loro temporizzazione. Configura il Planner attraverso la sua interfaccia di servizio REST del Planner. Monitora la cartella sanitaria del paziente sul DB ed eventualmente riconfigura il Planner.
Realizzazione	Nessun prototipo è stato attualmente realizzato. Il ruolo è attualmente attribuito allo utente utilizzatore della dashboard che attraverso un menu a tendina della Planner GUI sceglie task o insiemi di task predefiniti (procedure) da accordare manualmente allo scheduler.
Configurazione	
TODO	Progettazione e sviluppo di servizio componente di pianificazione del piano terapeutico giornaliero.

Monitor

Descrizione	Configuratore delle risorse dashboard (luogo e tipo di robot) e visualizzatore della della situazione della applicazione (valori del contesto)
Realizzazione	webApp ad eventi in HTML/javascript servita dall' Interactor Un timer fornisce un evento interno di <i>hearth beat</i> che attiva un polling periodico delle interfacce di introspezione REST dei manager e dei servizi esterni (dialogManager,introspection,planner,health e del DB) sezioni della situazione dai vari manager e servizi
Limiti	Manca attualmente il polling dei bridge (ws2naoQi e rosbridge_ws) ed un meccanismo di lettura della situazione integrata.
TODO	Trasformazione da GUI a nodo (con interfaccia REST) + GUI Lettura di stati operativi da topic ROS attraverso collegamento client a rosbridge_ws tramite libreria js http://static.robotwebtools.org/roslibjs Gestione appropriata delle inizializzazioni connesse con la selezione di location e tipo di robot, attualmente disabilitate e fissate alla configurazione {"location":"crsslab","robot":"pepper"} Manca la gestione dello stato del robot (indenticato dallo stato combinato del bridge verso naoQi), attualmente aggiornato dalla GUI interactor, da trasferire al server interactor e da leggersi da parte del minito tramite chiamata http POST
Snapshot	 <p>The screenshot shows a 'Status Monitor' interface. At the top, there are dropdown menus for 'Location:' (set to 'crsslab') and 'Robot:' (set to 'no robot'). Below these are three columns of status indicators:</p> <ul style="list-style-type: none"> Servizi: homeAss (green), healthAss (green), mongoDB (green), dialogManag (blue). Nodi: monitor: (blue), interactor: (green), planner: (green), robot: (black), ROS: (blue). Attività Dialogo: userTask: (green), userTaskStatus: (green), systemTask: (green), systemTaskStatus: (green). <p>At the bottom, there are several status messages: 'gui disconnected from Interactor', 'gui re-connected to Interactor', 'gui disconnected from Interactor', and 'gui Monitor connected to node Interactor'.</p>

Interactor

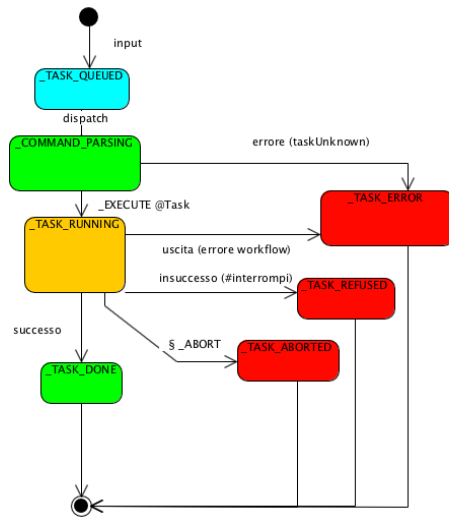
Descrizione	Gestore della modalità di interazione. Definisce attraverso guardie le finestre di ricezione attiva di input esterno e di espressione di testo animato o altro output. gestisce la policy di engagement (necessità della presenza e contatto fisico dell'interlocutore umano del dialogo). Gestisce la temporizzazione dell'I/O attraverso code e segnali di inizio/fine di i/o
--------------------	---

Realizzazione	Applicazione server <i>nodejs</i> . event-driven. Server socket.io per <i>monitor</i> e <i>dialogManager</i> Server http per le GUI Server http REST per <i>monitor(GUI)</i>
Limiti	L'output vocale è gestito nella GUI
TODO	Codifica della politica e del task di engagement. Portare la gestione dell'output <i>animatedSpeech</i> dalla GUI al nodo

Planner/Scheduler

Descrizione	<p>Gestore di coda di tasks con priorità. Ogni task è una istanza di un tipo di task definito da caratteristiche quali tempo di durata tipico e massimo, orario di inizio minimo e massimo, stringa di comando, messaggi di inizio task, feedback e fine task. I task di sistema sono pianificati ad orario, quelli richiesti dall'utente inseriti secondo disponibilità ed eventualmente interrotto in caso di conflitto con i task di sistema.</p> <p>Il planner pianifica e ripianifica i task di sistema. Lo scheduler mantiene una coda di task aggiornandone lo stato. Inserisce i task nella prima finestra utile compatibile con la definizione del tipo di task, eventualmente interrompendo eventuali task utente incompatibili. E' assegnata una finestra temporale di esecuzione, scaduta la quale il task è interrotto.</p> <p>Alla fine di ogni task avviene un resheduling. L'avanzamento dello stato dei task è notificato con messaggi di stato all'interactor che li inoltra al <i>dialogManager</i></p>
Realizzazione	<p>Applicazione <i>nodejs</i>.<i>event-driven</i> client dell'interactor (codice javascript inserito nella GUI del Planner)</p> <p>Il primo task in attesa è eseguito direttamente lanciato da un dispatcher o affidato asincronamente ad un server</p> <p>Per la esecuzione dei task ROS un dispatcher genera un <i>ActionLibServer</i> con callback asincrone per la notifica di messaggi di avanzamento e fine task.</p>
Limiti	Stati del task: <i>PENDING,REFUSED,RUNNING,ABORTED,TIMED_OUT,SUCCEDED</i>
Configurazione	
TODO	

Note



Stati del task in esecuzione

Spawner/ActionController

Descrizione	
Realizzazione	<p>Procedura esecutiva del Planner</p> <p>I task di notifica inviati all' <i>interactor</i> con evento <i>planner-to-interactor</i></p> <p>I task di dialogo di sistema (health tasks) sono inviati all'<i>interactor</i> con evento <i>planner-to-interactor</i> con attivazione di temporizzatore di timeout .</p> <p>Task esecutivi sono passati ad una procedura esecutiva <i>actionServer</i> con attivazione di temporizzatore.</p> <p>Alla ricezione di un evento di timeout questo è notificato all' <i>interactor</i> come task di notifica di stato <i>_TIMED_OUT</i></p>
Configurazione	
Limiti	Non è implementato il meccanismo di interruzione prematura dei task in caso di conflitto di priorità previsto dalla logica di gestione dell' <i>actionLib</i> ROS.
TODO	
Note	

7.2.2 Servizi Watson Dialog Assistant

Per la comprensione di frasi in linguaggio naturale espresse dal paziente durante il dialogo spontaneo (domande) o indotto (risposte) vengono sfruttati servizi Dialog Assistant della piattaforma di conoscenza IBM Watson.

IN questi servizi, un classificatore neurale analizza la frase e riconosce la presenza di termini associabili all'intenzione della frase (Intents) o alla categoria di soggetti, oggetti o proprietà citate (Entities).

Al messaggio è quindi applicato un workflow di processamento a regole.

Il messaggio in input è analizzato applicando una sequenza di gruppi di regole (if then else). Se viene riconosciuta la condizione di ingresso (Intent e/o Entity) il gruppo di regole è attivato ed applicato.

Ogni regola può assegnare valori a variabili logiche e consumare il messaggio, fornire una risposta e rimandare all'attesa di nuovo input o passarlo ad una lista di regole subordinate da verificare in sequenza fino a trovarne una per cui è valida la condizione di accensione.

Ogni regola può consumare il messaggio e rimandare a nuovo input, o , a seguito del riconoscimento di una combinazione logica di Intents, Entities e valori di variabili logiche, rimandare a regole subordinate ovvero saltare esplicitamente ad altro gruppo di regole.

E' possibile far riconoscere messaggi strutturati, attraverso la definizione di slots (proprietà). Una funzionalità interna all'analizzatore linguistico permette di definire richieste di completamento dei valori delle proprietà definite, se originariamente non presenti nel messaggio di input formulato dall'umano.

I servizi vengono richiamati indirettamente attraverso nodi server wrapper

Sono configurati gli scheletri di due assistenti di dialogo, un assistente domotico generico ed un assistente sanitario specializzato che sfruttano la conoscenza del difefrente contesto di esecuzione per attribuire significato ai utilizzano differentemente i termini negli intents e nelle entities

Le figure 2 e 3 mostrano due istantanea di parte della schermata della interfaccia di programmazione del workflow del dialogo. E' visibile la organizzazione in sequenza di gruppi di regole che processano l'ingresso. In fig xx è e espanso il modulo di riconoscimento di richiesta di *Azione Immediata* dell'HomeAssistant con una esemplificazione della possibilità di uscita dalla sequenzialità del processamento. In fig yy è e espanso il modulo di gestione dell'*attitude* a chiusura del dialogo relativo ad un task sanitario nell'HealthAssistant.

Segue una descrizione sintetica della realizzazione. In Appendice i dettagli.

IBM Watson Assistant

[Skills /](#)

HomeAssistant
Intrattenimento come home assistant

Intents Entities **Dialog** Options Analytics Versions Content Catalog

HomeAssistant

- Benvenuto
welcome
1 Responses / 20 Context Set / Does not return
- Controllo Conversazione (_DIALOG_CONTROL)
#SISTEMA && @ControlloConversazione
1 Responses / 4 Context Set / Skip user input / Does not return
- Comando di Sistema
#SISTEMA
1 Responses / 1 Context Set / Skip user input / Does not return
- Richiesta verbale di Azione (_ASK_FOR)
#Azione
1 Responses / 5 Context Set / Skip user input / Does not return
 - Skip user input and evaluate child nodes
 - Occupato
\$userTask=null || \$systemTask=null
2 Responses / 0 Context Set / Return allowed
 - Azione Immediata
@ComandoImmediato
1 Responses / 5 Context Set / Skip user input / Return allowed
 - Skip user input and evaluate child nodes
 - alzare
@ComandoImmediato=="alzare"
1 Responses / 3 Context Set / 2 Slots / Jump to / Return allowed
[Jump to Lancio di task immediato \(Evaluate condition\)](#)
 - ballare
@ComandoImmediato=="ballare"
0 Responses / 4 Context Set / 1 Slots / Jump to / Return allowed
[Jump to Lancio di task immediato \(Evaluate condition\)](#)
 - altrimenti
anything else
1 Responses / 0 Context Set / Return allowed
 - Task Differito
@ComandoDifferito
1 Responses / 1 Context Set / Skip user input / Return allowed
 - TaskNonEseguitibile
anything_else
1 Responses / 1 Context Set / Return allowed

Fig. 2 Istantanea della interfaccia di programmazione del dialogo di Watson HomeAssistant
Espansione del modulo di regole per il trattamento di Richiesta verbale di Azione (caso _ASK_FOR)

HealthAssistant2

Intents Entities **Dialog** Options Analytics Versions Content Catalog



Fig. 3 Istantanea della interfaccia di programmazione del dialogo di Watson HomeAssistant
Espansione del modulo di regole per il trattamento di Richiesta verbale di Azione (caso _ASK_FOR)

Assistente di conversazione domestico Watson HomeAssistant (servizio)

<p>Descrizione</p>	<p>Servizio Watson Dialog Assistant dedicato al processamento di frasi in linguaggio naturale configurato per il supporto di dialogo domestico nei casi d'uso: CHAT, ASKFOR, SYSTEM e parzialmente DIALOG_CONTROL</p>
<p>Realizzazione</p>	<p>Programmazione di una skill di Watson Dialog Assistant con licenza base. Un termine può essere assegnato ad un solo intent. Il riconoscimento di un intent prevale sul riconoscimento di una entità. La figura mostra la macchina a stati a stati abbozzata nella attuale realizzazione nelle regole di dialogo</p>
<p>Configurazione</p>	<p>Interfaccia web di programmazione IBM Watson Dialog Assistant Account IBM username: "machi@pa.icar.cnr.it", password: "pepperChat2019"</p>
<p>Limiti</p>	<p>Realizzazione attualmente limitata alla dimostrazione dei casi base. Vari Intents e Entities inutilizzati. Dialog Flow embrionale. Le API non permettono l'inserimento di variabili di contesto nel payload del messaggio di chiamata.</p>
<p>TODO</p>	<p>Esplorare la funzione di disambiguazione presente nella versione a pagamento del servizio. Modifica delle API di chiamata per permettere la trasmissione di variabili del contesto AMICO insieme al messaggio (es serializzazione/deserializzazione di oggetto stato in Definizione di una policy di controllo di flusso del dialogo e realizzazione del controllo di flusso pilotato da comandi dell 'Interactor.</p>

Assistente di conversazione sanitario Watson HealthAssistant (servizio)

<p>Descrizione</p>	<p>Servizio Watson Dialog Assistant dedicato al processamento di frasi in linguaggio naturale configurato per il supporto di dialogo domestico nei casi d'uso: <i>_MOTIVATION, _EXECUTE_SYSTEM</i> e parzialmente <i>DIALOG_CONTROL</i></p>
<p>Realizzazione</p>	<p>Programmazione di una skill di Watson Dialog Assistant con licenza base. Un termine può essere assegnato ad un solo intent. Il riconoscimento di un intent prevale sul riconoscimento di una entità. La figura mostra la macchina a stati abbozzata nella attuale realizzazione nelle regole di dialogo</p>
<p>Configurazione</p>	<p>Interfaccia web di programmazione IBM Watson Dialog Assistant Account IBM username: "machi@pa.icar.cnr.it", password: "pepperChat2019"</p>
<p>Limiti</p>	<p>Realizzazione attualmente limitata alla dimostrazione dell'utilizzo di slots nella disambiguazione di casi base ed alla gestione dell'attitude attraverso inserimento di prologhi e epiloghi differenziati nelle risposte. Vari Intents e Entities inutilizzati. Dialog Flow embrionale. manca la definizione e gestione di comandi di flusso per il set di parametri di contesto quali attitude ed engagement,</p> <p>Le API non permettono l'inserimento di variabili di contesto nel payload del messaggio di chiamata.</p>
<p>TODO</p>	<p>Esplorare la funzione di disambiguazione presente nella versione a pagamento del servizio. Modifica delle API di chiamata per permettere la trasmissione di variabili del contesto AMICO insieme al messaggio (es serializzazione/deserializzazione di oggetto stato) Definizione di una policy di controllo di flusso del dialogo e realizzazione del controllo di flusso pilotato da comandi dell 'Interactor.</p>

7.3 ws2naoQi(bridge)

Descrizione	<i>WebSocket server , ponte verso il server naoQi su pepper. Gestisce la sessione di collegamento , attivando il robot e spegnendolo in caso di inutilizzo prolungato. Riceve messaggi contenenti parametri di interazione ed attiva i relativi servizi sul robot. A fine azione risponde al client con un messaggio di stato</i>
Realizzazione	Websocket server tornado in python . Alla attivazione apre una sessione naoQi sul robot ed ottiene proxies sui servizi (aspeech animato, movimento etc) controllati da un temporizzatore. Ogni nuovo messaggio reinizializza il temporizzatore. I parametri di azione sono trasferiti tramite un oggetto serializzato in json e passato come messaggio testuale nel campo data (vedi § 9.4.4)
Configurazione	
Limiti	
TODO	.

7.4 actionLibServer (controller)

Descrizione	<i>Nodo ROS Controller di task in esecuzione entro il framework . Comunica con il client tramite un topic di goal , uno di feedback, uno di stato definiti</i>
Realizzazione	Nodo esecutivo ROS I parametri di azione sono trasferiti tramite un oggetto serializzato in json e passato come messaggio testuale nel campo data (vedi § 9.4.4)
Configurazione	
Limiti	
TODO	.

7.5 Componenti grafiche

Lo strumento di simulazione e controllo della applicazione è costituito da una webapp denominata *checkboxboard* che permette di interagire direttamente con varie componenti inviando esplicitamente messaggi (a testo libero o predefiniti per esercitare i vari casi d'uso), selezionando parametri, task e comandi di controllo di flusso, o monitorarne la mutua interazione. Ogni interfaccia grafica è sviluppata in html+javascript come una applicazione web indipendente ed inserita un *frame* della applicazione web html di coordinamento visuale.

La figura 4 mostra una istantanea della visualizzazione della situazione del sistema sulla *checkboxboard*.

The screenshot displays a web browser window with the URL `localhost:3000/dashboard.html`. The interface is divided into several panels:

- Home Assistant (Watson):** A code editor showing JSON configuration for various services like `homeAss`, `healthAss`, `mongoDB`, and `dialogManav`.
- Health Assistant (Watson):** A chat interface with a prompt: "Do informazioni ed esegue compiti sanitari."
- MongoDB:** A panel for database management.
- Interactor:** A chat interface showing a conversation between `planner` and `patient`. The `planner` says "RESET" and "NOTIFY_PENDING". The `patient` says "ciao, cosa posso fare per te?". The `homeAss` responds with "ciao, cosa posso fare per te?".
- Status Monitor:** A panel showing the status of various components: `homeAss`, `healthAss`, `mongoDB`, `dialogManav`, `Interactor`, `planner`, and `robot`. It also shows the `Attività Dialogo` and `Semantica Dialogo`.
- Planner:** A panel with a "RESET" button and a "QUEUE" button.
- Scheduler:** A panel with buttons for "START", "PAUSE", "STEP", "NEXT", "RESUME", and "RESET". It shows a "Task Queue" table with columns: Task, Plan, Schedule, Exp.Dur, Timeout, Status.
- Dispatcher:** A panel with "no message" text.
- Docs:** A panel with a navigation menu: Scenario, Simulator, Home, Health, Dialog, Bridge, Interactor, Planner.

At the bottom of the interface, there is a "Chat" section with a text input field and buttons for "Flusso", "Attitudine", "Mood", and "Engagement".

Fig. 4 Componenti di interazione grafica dell'ambiente di simulazione

La figura 5 mostra le relazioni fra le interfacce grafiche, i servizi ed i server esecutivi.

Segue una descrizione sintetica delle funzionalità delle singole interfacce grafiche.

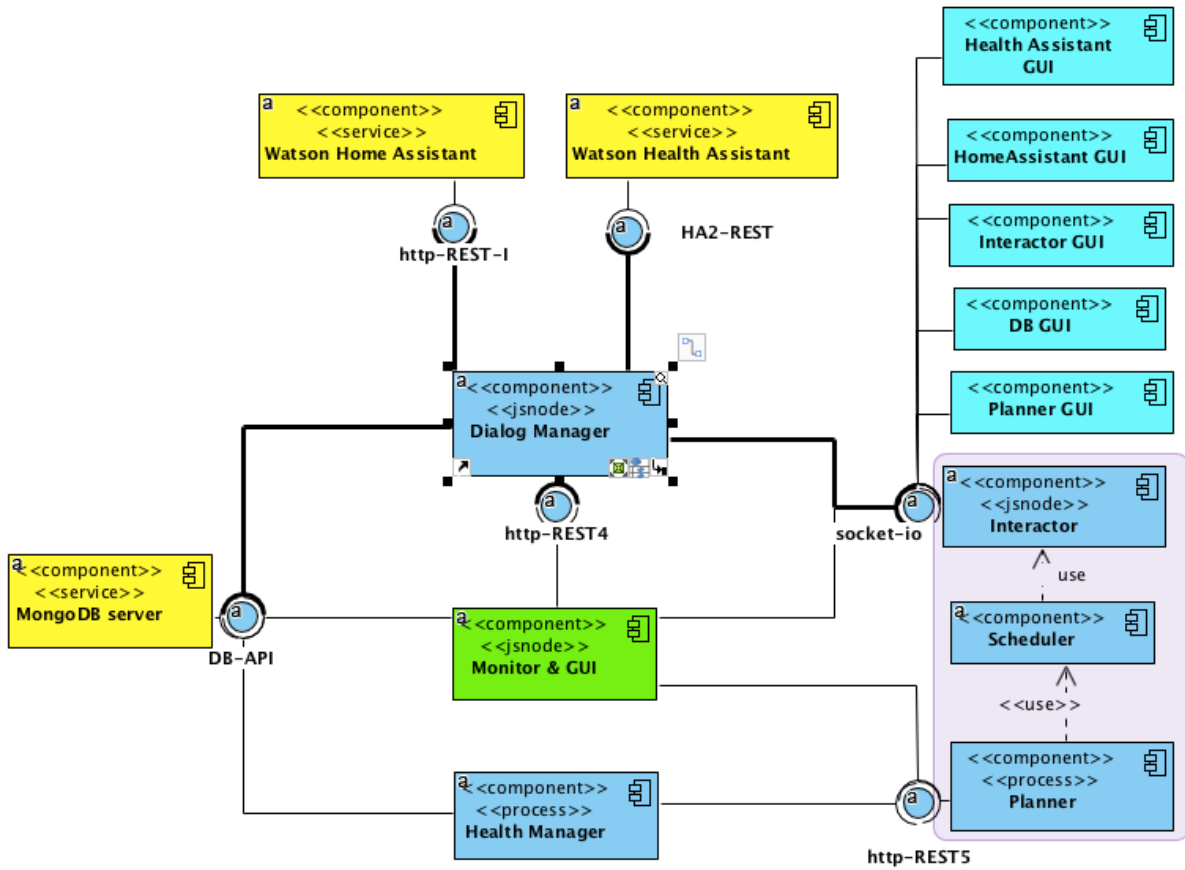


Fig. 5 Componenti di interazione grafica dell'ambiente di simulazione

Gui dell'Assistente domestico Watson

Descrizione	<p><i>GUI di visualizzazione del testo dei messaggi contenuti negli eventi di dialogo scambiati fra dialogManager, e HomeAssistant WatsonI messaggi sono mostrati in formato chat con nome del mittente e contenuto.</i></p> <p><i>Una finestra ancillare permette di esplorare i valori dei metadati di input ed output scambiati attraverso la porta di chiamata insieme al testo di domanda o risposta</i></p>
Realizzazione	<p><i>Adattamento del codice contenuto nel client della applicazione chat di Dialog Assistant in javascript fornita come esempio in tutorial IBM Watson</i></p> <p><i>Una area di input permette di esercitare il servizio senza la intermediazione filtrante del DialogManager</i></p>

Snap

Home Assistant (Watson)

Home Assistant (Watson)



Cosa posso fare per te?

ciao

ciao, cosa posso fare per te?

accendi

accendere cosa? la luce, la tv o il condizionatore?

la luce

accendere interruttore dove? Posso raggiungere la cucina, il soggiorno o il bagno

in bagno

mi organizzo per accendere interruttore in bagno

Inserisci una frase per l'assistente domestico ...

```
57   "node_1_1552207786029": {
58     "0": {
59       0
60     }
61   },
62 },
63   "branch_exited": true,
64   "branch_exited_reason": "completed"
65 },
66 "case": "ASK_FOR",
67 "role": "HOMEGAS",
68 "error": null,
69 "input": "accendi",
70 "place": "soggiorno",
71 "query": null,
72 "topic": "DEFERRED_TASK",
73 "subject": "pepper",
74 "userTask": "accendere",
75 "animation": "Waiting/Think_3",
76 "predicate": "accendere",
77 "complement": null,
78 "systemTask": null,
79 "errorMessage": "",
80 "prevPredicate": null,
81 "prevComplement": null,
82 "userTaskStatus": "NEW",
83 "assistantStatus": "READY",
84 "systemTaskStatus": null,
85 "pendingNotification": null,
86 "where": "in bagno",
87 "object": "interruttore"
```

GUI dell'Assistente sanitario Watson

Descrizione	<p>GUI di visualizzazione del testo dei messaggi contenuti negli eventi di dialogo scambiati fra dialogManager, e Health Assistant</p> <p>Una finestra ancillare permette di esplorare i valori dei metadati di input ed output scambiati attraverso la porta di chiamata insieme al testo di domanda o risposta</p> <p>Una area di input permette di esercitare il servizio senza la intermediazione filtrante del DialogManager</p>
Realizzazione	<p>Adattamento del codice contenuto nel client della applicazione chat di Dialog Assistant in javascript fornita come esempio in tutorial IBM Watson.</p> <p>I messaggi sono mostrati in formato chat con nome del mittente e contenuti</p>
Snap	

GUI del Mongo DB server

Descrizione	GUI di visualizzazione delle query e al DB e delle risposte
Realizzazione	webApp ad eventi in HTML/javascript servita dall' Interactor
Snap	<p>MongoDB</p> <pre> assistant "{ topic:sintomi subject:astenia predicate:definizione }" mongo mi dispiace, conosco il termine astenia ma non la sua definizione assistant "{ topic:farmaci subject:tavor predicate:indicazioni }" mongo ansia, tensione e insonnia assistant "{ topic:sintomi subject:ansia predicate:definizione }" mongo L'ansia è uno stato psichico di un individuo, prevalentemente cosciente, caratterizzato sensazione di intensa preoccupazione o paura, spesso infondata, relativa a uno stimolo ambiental associato a una mancata risposta di adattamento da parte dell'organismo in una determinata situ esprime sotto forma di stress per l'individuo stesso. </pre>

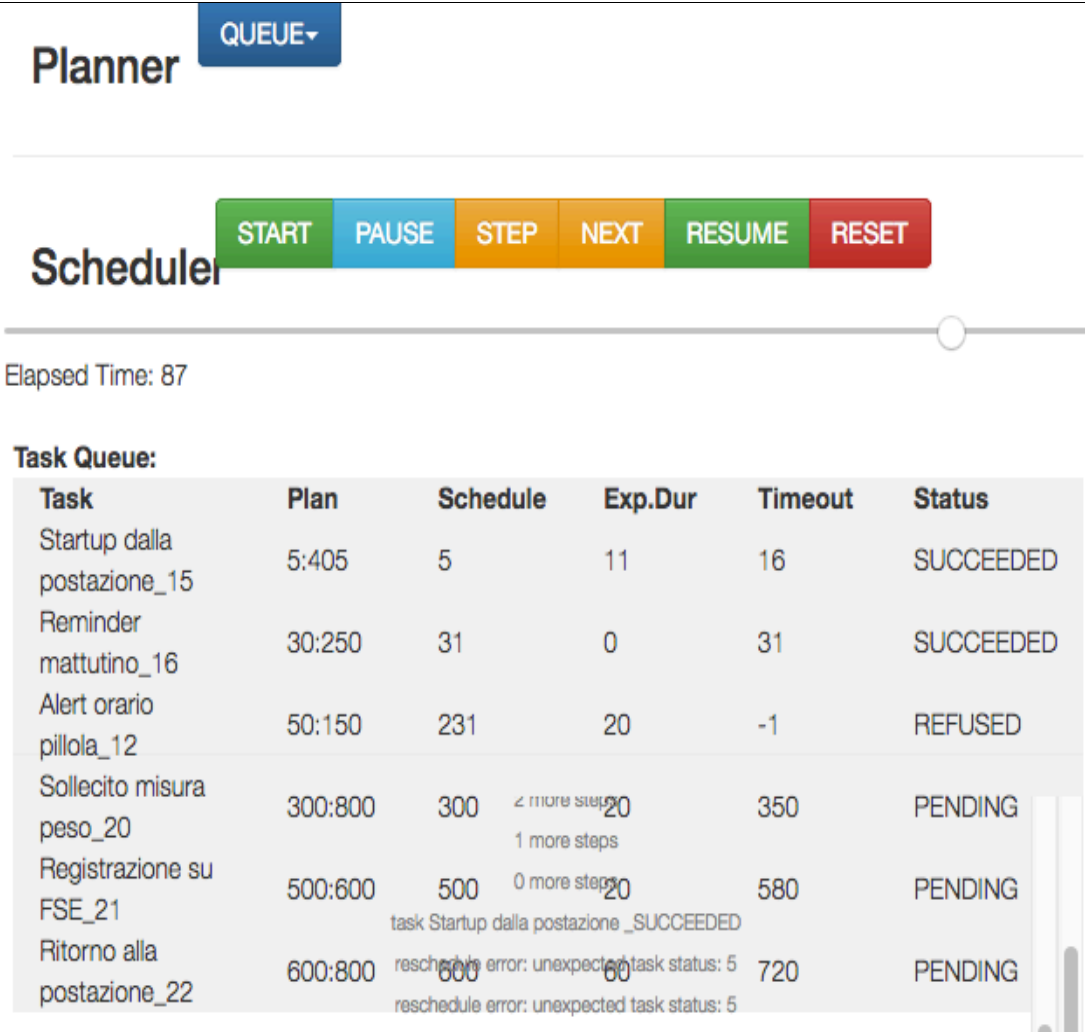
GUI del Monitor

Vedi la descrizione del *monitor* \$7.1 che è realizzato in codice javascript integrato entro la GUI ed attivato da un timer periodico.

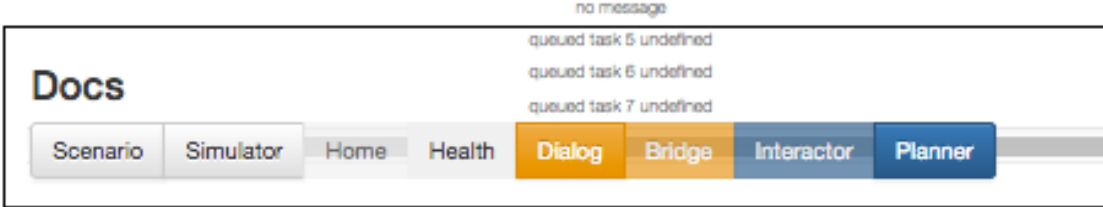
GUI dell'Interactor

Descrizione	<p><i>GUI di visualizzazione del testo dei messaggi contenuti negli eventi di dialogo scambiati fra dialogManager, Interactor, e</i></p> <p><i>Menu di selezione messaggi vocali del paziente,,comandi di controllo flussosso del dialogo, assegnazione di valori di mood e attitude</i></p>
Realizzazione	<p><i>Estensione ed adattamentio del codice contenuto nel client della applicazione chat di Dialog Assistant in javascript fornita come esempio in tutorial IBM Watson.</i></p> <p><i>I messaggi sono mostrati in formato tipico chat con nome del mittente e contenuto.</i></p> <p><i>Attualmente realizza la logica esecutiva dell'interactor: buffering del parlato e e, sincronizzazione del ll'accettazione di comunicazione utente tramite guardia blocacnte la gui. Redirezione del testo da pronunciare al bridge naoQiws o a servizio esterno google ResponsiveVoice.</i></p>
Eventi	<p>interactor_client: <i>connenct,disconnect,reconnect,error, config, interactor_ho_human,</i></p> <p>qiws_client: <i>open,close,message</i></p>
TODO	<p><i>Realizzazione del controllo di flusso del dialogo e dell policy di attitute e configurazione conseguente del dialogManager.</i></p>
Snap	

GUI del Planner/scheduler.

Descrizione	<i>GUI di gestione della coda di task programmati e di controllo della timeline di esecuzione</i>																																										
Realizzazione	webApp ad eventi in HTML/javascript servita dall' Interactor Menu dropdown permettono la istanziare dei task da inviare allo scheduler che li accoda secondo la policy attiva. Pulsanti di azione controllano la timeline di esecuzione. Un'aretestuale visualizza i valori di stato del task attuale forniti dal dispatcher, quando attivato dello scheduler o richiamato dagli eventi di stato del controller di azione attivo.																																										
TODO																																											
Snapshot	 <p>Task Queue:</p> <table border="1"> <thead> <tr> <th>Task</th> <th>Plan</th> <th>Schedule</th> <th>Exp.Dur</th> <th>Timeout</th> <th>Status</th> </tr> </thead> <tbody> <tr> <td>Startup dalla postazione_15</td> <td>5:405</td> <td>5</td> <td>11</td> <td>16</td> <td>SUCCEEDED</td> </tr> <tr> <td>Reminder mattutino_16</td> <td>30:250</td> <td>31</td> <td>0</td> <td>31</td> <td>SUCCEEDED</td> </tr> <tr> <td>Alert orario pillola_12</td> <td>50:150</td> <td>231</td> <td>20</td> <td>-1</td> <td>REFUSED</td> </tr> <tr> <td>Sollecito misura peso_20</td> <td>300:800</td> <td>300</td> <td>20</td> <td>350</td> <td>PENDING</td> </tr> <tr> <td>Registrazione su FSE_21</td> <td>500:600</td> <td>500</td> <td>20</td> <td>580</td> <td>PENDING</td> </tr> <tr> <td>Ritorno alla postazione_22</td> <td>600:800</td> <td>600</td> <td>60</td> <td>720</td> <td>PENDING</td> </tr> </tbody> </table>	Task	Plan	Schedule	Exp.Dur	Timeout	Status	Startup dalla postazione_15	5:405	5	11	16	SUCCEEDED	Reminder mattutino_16	30:250	31	0	31	SUCCEEDED	Alert orario pillola_12	50:150	231	20	-1	REFUSED	Sollecito misura peso_20	300:800	300	20	350	PENDING	Registrazione su FSE_21	500:600	500	20	580	PENDING	Ritorno alla postazione_22	600:800	600	60	720	PENDING
Task	Plan	Schedule	Exp.Dur	Timeout	Status																																						
Startup dalla postazione_15	5:405	5	11	16	SUCCEEDED																																						
Reminder mattutino_16	30:250	31	0	31	SUCCEEDED																																						
Alert orario pillola_12	50:150	231	20	-1	REFUSED																																						
Sollecito misura peso_20	300:800	300	20	350	PENDING																																						
Registrazione su FSE_21	500:600	500	20	580	PENDING																																						
Ritorno alla postazione_22	600:800	600	60	720	PENDING																																						

GUI Documentazione

Descrizione	<i>Area con pulsanti di azione Attiva la visualizzazione in altra finestra di sezioni di questo rapporto in formato html.</i>
Realizzazione	<i>Area attiva ospitata nella GUI del Planner..</i>
TODO	
Snapshot	

8 Policies

8.1.1 User profiling

TODO: Non è prevista attualmente profilazione dell'interlocutore umano. Ad evitare appesantimento del sistema di regole interne degli assistenti Watson, questa potrebbe venire realizzata attraverso la configurazione ed attivazione a run-time di differenti istanze di servizio per l'HomeAssistant e l'HealthAssistant

8.1.2 Preemptive priority scheduling

Ogni task è una istanza di una tipologia di task riconosciuto per cui sono definiti un istante minimo e massimo di attivazione (task di sistema) ed una finestra di durata massima, pena interruzione precoce. (stato `_PREEMPTED`)

Il task può essere ad esecuzione immediata (task richiesto estemporaneamente dal paziente) o differita (pianificato dal sistema o accodato a seguito di richiesta utente). Il task ad esecuzione immediata non è accettato dal *dialogManager* mentre sta eseguendo un task sanitario (caso `_EXECUTE`), altrimenti è comunicato al *planner* ed eseguito immediatamente. Il task è inserito in testa allo stack di esecuzione, senza differimento del tempo di attivazione dei task successivi.

L'attivazione di un task programmato interrompe prematuramente il task utente e la condizione è notificata all'*interactor* che la trasmette al *dialogManager*.

Per la programmazione (inserimento in coda), ad ogni task è assegnata una finestra di esecuzione di lunghezza pari alla durata massima definita per il suo tipo. Per l'accodamento di un task di sistema, si cerca la prima finestra utile a partire dal tempo minimo di attivazione (o dal momento corrente) e di larghezza minima pari al suo tempo di durata massimo. Se trovata il task è dichiarato `_PENDING` fino al momento di attivazione, quando diviene `_RUNNING`. Se nessuna finestra è attualmente disponibile, il task è dichiarato `_REFUSED`.

Poiché la chiusura di un task prima del timeout libera un intervallo di tempo utile, la fine di un task provoca la riconsiderazione dei tempi di attivazione di tutti i task `_PENDING` e `_REFUSED`. Il task completato è etichettato come `_SUCCEEDED` o `_ABORTED` in funzione dello stato di uscita.

Problematiche non affrontate.

- La gestione della procedura di *preemption* è modellata.
- L'inserimento di un task utente differito avviene in coda alla lista dei task, mentre potrebbe essere utilmente inserito in mezzo alla coda, gestendo opportunamente la priorità dei task di sistema.

- Alcune tipologie di task di sistema, completati con insuccesso, potrebbero essere rischedulati direttamente , entro la finestra di possibile attivazione configurata, senza richiesta di ridefinizione del piano dei task.

8.1.3 Robot attitude

L'atteggiamento (*attitude*) da assumere verso il paziente da assumersi da parte del robot , è stato modellato attraverso tre stili di comportamento indicati come *_EMPATIC*, *_NEUTRAL* ed *_IMPERATIVE*, comunicati come parametri di configurazione attraverso un comando di flusso del dialogo.

La scelta dell'*attitude* opportuna ci si aspetta possa essere messa in relazione alle variazioni di umore del paziente (modellate come *_EXITED*, *_NEUTRAL* o *_DEPRESSED*), misurabile a partire da espressioni verbali sollecitate da una opportuna attività terapeutica.

L'altra variabile di interesse è l'andamento del alla valutazione del livello di aderenza (*compliance*) degli stati di uscita dei task terapeuti al protocollo sanitario, valutabile dall' health Controller in una scala di tempo medio (ore o giorni)

All'interno del sistema di regole dell'HealthAssistant, è stata esemplificato un adeguamento della risposta del robot all'atteggiamento programmato (dall' HealthController) e configurato dall' Interactor.

L'adeguamento è realizzato inserendo un prologo (motivante) ed un epilogo (giudicante) prima e dopo il corpo informativo, in sezioni di complemento alla sezione informativa della risposta o del modulo di dialogo. Il primo ed il terzo passo forniscono una introduzione ed una chiusura differente in funzione dell'atteggiamento, esprimendo incoraggiamento e vicinanza tollerante, o sollecito in finzione dell'atteggiamento richiesto.

TODO: Gli esempi di prologo motivante ed epilogo giudicante sono elementari e puramente indicativi. Non esiste, allo stato attuale, conforto di modellazione psicologica alla policy espressa nella macchina a stati sotto indicata, dove un miglioramento del mood e della compliance portano ad un rilassamento dell'atteggiamento verso l'empatia, ed un peggioramento delle stesse portano alla adozione di un atteggiamento meno permissivo.

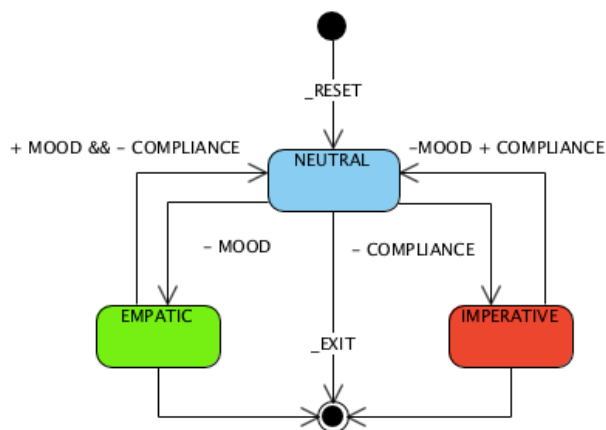


Fig. 6 Assegnazione dello stato di attitudie in funzione di mood e compliance

Prudentemente potrebbe riservarsi la modalit  permissiva solo nei casi di aderenza stabile (per non favorire rilassamento delle abitudini del paziente) e quella imperativa ai soli casi di grave inadempienza (per limitare irrigidimenti emotivi e perdita di confidenza col:

Patient MOOD vs Therapy COMPLIANCE	<i>COMPLIANT</i>	<i>MODERATE</i>	<i>POOR</i>
<i>DEPRESSED</i>	<i>SYMPATIC</i>	<i>EMPATIC</i>	<i>DIRECTIVE</i>
<i>NEUTRAL</i>	<i>EMPATIC</i>	<i>EMPATIC</i>	<i>EMPATIC</i>
<i>EXCITED</i>	<i>EMPATIC</i>	<i>EMPATIC</i>	<i>EMPATIC</i>

8.1.4 Animated speech

L'*animated speech* corrisponde alla integrazione di espressione corporea e parlato. Essa implica la sincronizzazione di parlato e gesti .

I robot Aldebaran Nao e Pepper offrono integrate nei behaviour di sistema textToSpeech (richiamabili attraverso l'interfaccia websocket naoQi) due modalit : un linguaggio di annotazione della frase con gestures predefinite o la attivazione delle stesse alla presenza di termini chiave nella frase da dire.

La policy attuale prevede la annotazione da parte dell'Assistente di dialogo Watson, di un identificatore di gesture base sotto forma di una valore di stato del contesto semantico, in funzione del senso e/o del valori di atteggiamento e

Realizzazione

Alle frasi di risposta del Gestore di dialogo sono associate espressioni di linguaggio corporeo

del robot selezionate fra quelle predefinite, per i robot di classe Pepper, nella classe *ALAnimation* delle librerie di sistema naoQi della Aldebaran.

L'associazione avviene attraverso la assegnazione, per ogni frase di risposta dell'Assistente di dialogo, dell'identificatore di animazione, nella variabile di contesto *animation* (sezione *semantica*).

Il valore di animazione viene trasmesso con l'oggetto contesto all' *interactor* che lo utilizza nella costruzione del comando di *AnimatedSpeech* per naoQi.

La tabella a seguito mostra le associazioni programmate nelle risposte dell' Home Manager e dell' HealthAssistant e trasferite nel contesto di dialogo nella proprietà *animation*

Risposta	Pattern animazione
provo a ballare	Emotions/Positive/Hysterical_1
spegnere cosa?la luce, la tv o il condizionatore?	BodyTalk/BodyTalk_8
mi organizzo per spegnere il televisore in cucina	BodyTalk/BodyTalk_1
questo non lo posso fare	Emotions/Negative/Bored_1
il mio senso etico mi richiede di non risponderti	Gestures/YouKnowWhat_1
non mi distrarre! sto eseguendo l'attività	Emotions/Negative/Bored_1
Non ho capito. Puoi riformulare la frase?	Gestures/YouKnowWhat_1
devo prima eseguire un altro Task	Gestures/CalmDown_5
Nel frattempo è arrivato questo messaggio	Gestures/YouKnowWhat_5
Arrivederci	Gestures/BowShort_1
Salve. Sono il tuo assistente domestico. Intrattengo e fornisco informazioni su quello che so.	BodyTalk/BodyTalk_8
Oops ! Ho riscontrato un errore non identificato	Gestures/YouKnowWhat_6

q uesto non lo posso fare	Emotions/Negative/Bored_1
adesso ti dico quello che so	Gestures/YouKnowWhat_1
fammi consultare il database	BodyTalk/BodyTalk_6
Non so chi o cosa sia	BodyTalk/BodyTalk_4

9 Dettagli implementativi

9.1 Servizi Watson

9.1.1 Skill HomeAssistant

Account IBM	<i>Chiedere per email a machi.alberto@pa.icar.cnr.it</i>
Skill ID	a4516ab6-904c-48b0-9d72-8d1ea6970595
Workspace ID	4bce215d-9f1d-44da-903d-f1a745ac0154
Legacy v1 Workspace	URL: https://gateway.watsonplatform.net/assistant/api/v1/workspaces/4bce215d-9f1d-44da-903d-f1a745ac0154/message
NLP Intents <i>(categorie che definiscono l'intenzione della frase):</i>	#Azione, #ChiCosa, #Conferme, #Controindicazioni, #Dove, #Effetti_collaterali, #Indicazioni, #Intercalare, #Negazione, #Nome, #Offesa, #Percezione, #Posologia, #Saluto
NLP Entities <i>(classi di Entità non tutte utilizzate nel workflow di dialogo)</i>	@Attivabile, @AzioneRiflessivaApparente, @CASE, @CodiceErrore, @ComandoDifferito, @ComandoImmediato, @ControlloConversazione, @Disattivabile, @Esibizione, @Farmaco, @LuogoConosciuto, @OggettoConosciuto, @ParteDelCorpo, @ParteDelCorpoLateralizzata, @Patologia, @PersonaConosciuta, @sintomi, @StatoAssistente, @SYSTEM_COMMAND, @SYSTEM_TASK, @TermineSanitario, @USER_TASK_STATUS,
Dialog Modules	Benvenuto,ControlloConversazione,Comando di Sistema,Richiesta verbale di Informazione,Intercalare,Intrattenimento,Etica, In altri casi,Verifica disponibilità,Lancio di task immediato,Pianificazione di task differito,Verifica di Notifica Pendente, Ritorno a Chat, Uscita

9.1.2 Skill HealthAssistant2

Account IBM	<i>Chiedere per email a machi.alberto@pa.icar.cnr.it</i>
Skill ID	a4516ab6-904c-48b0-9d72-8d1ea6970595
Workspace ID	4bce215d-9f1d-44da-903d-f1a745ac0154

<i>Legacy v1 Workspace</i>	URL:https://gateway.watsonplatform.net/assistant/api/v1/workspaces/4bce215d-9f1d-44da-903d-f1a745ac0154/message
<i>Assistant access</i>	Username:779a300c-472e-4bbf-be28-3a8d019a823f Password: BpqhQEpa8CQp
<i>NLP Intents</i>	#Conferma,#InfoSanitaria, #Interruzione, #Negazione, #Percezione, #SYSTEM
<i>NLP Entities</i>	@ansia,@ASSISTANT_STATUS,@astenia,@Atteggiamento,@bene,@CASE,@cosicosi,@SIALOG_CONTROL,@Frequenza,@Gradazione,@LivelloSintomo,@male,@NO,@ParametroBiometrico,@ParametroFarmacologico,@prurito,@SI,@Sintomo,@SYSTEM_COMMAND,@SYSTEM_TASK,@SYSTEM_TAK_STATUS,@tachicardia,@TermineSanitario,@ValoreElevato
<i>Dialog Modules</i>	Benvenuto, Controllo Conversazione, Verifica disponibilità, Esecuzione Comando di Sistema, Info Sanitaria, Altro, Premessa Motivante, Esecuzione del compito, Log risultati, Chiusura di rinforzo, Intercalare, Lettura di Notifica, Chiusura della sessione

9.2 Sintassi dei comandi

<i>Tipo Comando</i>	<i>Codificatore</i>	<i>Stringa di comando</i>
<i>Task utente</i>	“\$” + “\$”	“\$_STACK \$_BALLARE” “\$_QUEUE \$_ACCENDERE”
<i>Task sanitario</i>	“o”	“o _EXECUTE_TODO”, “o _EXECUTE_TEST” “o _EXECUTE_CHECKUP” “o _SOLICIT ricordati di pesarti” “o _NOTIFY è ora di prendere la pillola” “o _EXECUTE_TEST_MOOD”
<i>Task sistema (ROS)</i>	“\$”	“\$_EXECUTE_BACK_TO_DOCK” “\$_EXECUTE_STARTUP” “\$_EXECUTE_FSE_LOGGING” “\$_RESET”
<i>Controllo del flusso del dialogo</i>	“\$”	\$_RESET, \$_

9.3 Eventi del dialogo

Name	Description	Publisher	Subscriber
<i>homeAss-to-chatter</i>	frase	homeAssistant (utils1.js)	dialogManager (chatterboxAssistant.js)
<i>healthAss-to-chatter</i>	frase	health Assistant (utils2.js)	dialogManager (chatterboxAssistant.js)
<i>chatter-to-homeAss</i>	frase o comando o notifica	dialogManager (chatterbotAssistant.js)	homeAssistant GUI (home.html)
<i>chatter-to-healthAss</i>	frase, comando, o notifica	dialogManager (chatterbotAssistant.js)	healthAssistant GUI (health.html)
<i>chatter-to-interactor</i>	frase ,risultato query o notifica	dialogManager (chatterbotAssistant.js)	interactor (interaction-server.js)
<i>interactor-to-human</i>	messaggio al paziente o comando da verbalizzare	Interactor (interaction-server.js)	Interactor GUI (interactor.js)
<i>config</i>	Parametri configurazione (luogo e robot)	monitor GUI (monitor.js)	Interactot GUI Interactor.js plannerGUI (planner.js)
<i>planner-to-interactor</i>	Notifiche comandi esecutivi	planner (planner.js)	interactor (interaction-server.js)
<i>interactor-to-planner</i>	Comandi da accodare	Interactor (Interaction-server.js)	plannerGUI (planner.js)
<i>interaction-guard</i>	Guardia di abilitazione/disab.	Interactor (intractor-server.js)	Interactor (broadcast) (interactor.js) planner (planner.js)

9.4 IDE, inizializzazioni operative, bundle del software, bugs

9.4.1 Ambienti di sviluppo

- Le componenti software basate su *node.js v10.14.1* sono state sviluppate in linguaggio *javascript* e provate sotto sistema operativo *MacOs 10.12.6 (Sierra)* , utilizzando come ambiente di editing *Visual Studio Code v. 1.36*, *homebrew* come macOs installer, ed *npm v.6.10.1* come gestore di archivi *nodejs* .
- Le componenti software che utilizzano *ROS* ed i bridges sono state sviluppati in linguaggio *python v2.7* e provate sotto sistema operativo *Ubuntu v 14.04 LTS*. Installer *apt-get*

Si è utilizzato *ROS* versione *indigo* e *catkin* come gestore dei package *ROS*.

- Per la gestione del database si è utilizzato *MongoDB 4.2 Community Edition* installato sotto *MacOs Sierra* tramite *homebrew*. Come interfaccia *CRUD* è stata utilizzata *MongDB Compass [11]*

9.4.2 Bugs alla data di copertina

- L'interactor non invia correttamente alcuni task di sistema al *dialogManager* che non li esegue.

La gestione dello stato dell'Assistant alla ricezione degli eventi di notifica di avanzamento di stato dei task utente e di sistema non è completa e genera blocchi del dialogo

9.4.3 Software Bundle

Il codice del software sviluppato è stato raccolto nel bundle *AMICO.zip*

Mappa

node

assistant-simple-master (dialogManager e watson assistants)

package.json (configurazione moduli e dipendenze pe npm)

chatterbox-assistant.js (dialogManager)

public (pagine servite dal server http)

home.html (homeAssistantGUI)

health.html (healthAssistantGUI)

dashboard.html (dashboard)

js (codice di servizio)

node_modules (popolata da nmp install con librerie js)

eventBridge/examples/emulator (monitor+interactor+planner & GUIs)

package.json (configurazione moduli e dipendenze pe npm)

interaction-server.js (interactor)

public (pagine servite dal server http)

interactor.html (interactorGUI)

monitor.html (monitorGUI)

planner.html (plannerGUI)

testQi2.html (esercitatore di activeSpeech su naoQi)

js (codice di servizio)

interactor.js

monitor.js

planner.js

node_modules (popolata da nmp install)

catkin-ws-

src (sorgenti python)

amico (package amico)

action (descrizione messaggi rosActionLib)

launch (ROS launch files)

scripts (nodi e bridge, codice python)

Startup

- Inizializzazione mongDB su macOs:
`mongod --dbpath=/data/db --port=3010`
(o comando equivalente su altro Os)
- Inizializzazione componenti nodejs su Mac (dialogManager e interactor +planner+monitor+guis):
`cd <installation path>`
`cd AMICO/nodejs/ assistant-simple-master; npm start;`
`cd AMICO/nodejs/ eventBridge/examples/emulator;npm start`
(non ancora provato il porting del codice nodejs su ubuntu)
- Inizializzazione componenti ROS su Ubuntu
`cd <installation path>`
`cd AMICO`
`bash init_AMICO-ros.bash` (rosbridge web socket)
`bash init_AMICO-naoQi.bash` (pepper Qiws)
`python ./ros-catkin-ws/src/amico/scripts/dance_server.py`
(non ancora integrato come nodo eseguibile del modulo ros amico)

9.4.4 Esempi di codice

naoQiws.py (server ws in python)

```
import tornado.httpserver
import tornado.websocket
import tornado.ioloop
import tornado.web

import socket
import argparse
import sys
import json
import time
import almath
import math
from threading import Timer

import qi
import motion
from naoqi import ALProxy

chatTimeout = 300
wsTimeout = 300
wsTimer = None
chatTimer = None
wsTiming = None
application = None

class ws2naoQi(tornado.websocket.WebSocketHandler):
    def open(self):
        global wsTimer
        global chatTimer
        global wsTiming
        try:
            # metodo eseguito all'apertura della connessione
            print 'Nuova connessione'
            # Wake up robot standing
            posture_service.goToPosture("StandInit", 0.1)
            wsTiming.start()
        except RuntimeError:
            print ("websocket open error")
            sys.exit(1)

    def on_message(self, payload):
        # metodo eseguito alla ricezione di un messaggio
        # la stringa 'message' rappresenta il messaggio
        # reinizializza il timer dei messaggi
        global chatTimer
```

```

global wsTiming
global animation
try:
    animation      ='Emotions/Positive/Happy_4'
    chatTimer.cancel()
    chatTimer = Timer(chatTimeout,wsTiming.timeout)
    chatTimer.start()
    print payload
    commands= json.loads(payload)
    if 'animation' in commands.keys():
        animation=commands['animation'].encode('ascii','ignore')
    if 'message' in commands.keys():
        message =commands['message'].encode('ascii','ignore')
        command= '^start(animations/Stand/'+animation+') '+message
        print command
        print ("start")
        aspeech.say(command)
        self.write_message('speech_END')
        print ("end")

    if 'immediateTask' in commands.keys():
        print (commands)
except RuntimeError:

    print ("websocket message error")
    sys.exit(1)

def on_close(self):
    try:

        # metodo eseguito alla chiusura della connessione
        print 'Connessione chiusa'
        animation      ='Emotions/Positive/Happy_4'
        message        ="Chiudo la conessione e mi riposo "
        aspeech.say('^start(animations/Stand/'+animation+') '+message)
        motion_service.rest()
    except RuntimeError:
        print ("websocket open error")
        sys.exit(1)

def check_origin(self, origin):
    return True

class WsTiming():
    def wait(self):
        global wsTimer
        try:
            print("ws_timing WAIT")
            animation      ='Emotions/Positive/Happy_4'

```

```

        message      ='attendo apertura connessione entro'+str(wsTimeout)+'secondi'
        aspeech.say('^start(animations/Stand/'+animation+') '+ message)
        wsTimer = Timer(wsTimeout,self.end)
        wsTimer.start()
except RuntimeError:
    print ("Connection wait error")
    sys.exit(1)

def start(self):
    global wsTimer
    global chatTimer
    try:
        print("ws_timing START")
        wsTimer.cancel()
        message      ='attendo un messaggio entro'+str(chatTimeout)+'secondi'
        animation     ='Emotions/Positive/Happy_4'
        aspeech.say('^start(animations/Stand/'+animation+') '+message)
        chatTimer= Timer(chatTimeout,self.timeout)
        chatTimer.start()
    except RuntimeError:
        print ("Connection start error")
        sys.exit(1)

def end(self):
    if chatTimer!= None :
        chatTimer.cancel()
    if wsTimer!= None:
        wsTimer.cancel()
    try:
        print('ws_timing END')
        animation='Emotions/Negative/Bored_1'
        message='Adesso mi riposo. A dopo.'
        aspeech.say('^start(animations/Stand/'+animation+') '+ message)
        motion_service.rest()
    except RuntimeError:
        print ("Connection end error")
        sys.exit(1)

def timeout(self):
    global wsTimer
    global chatTimer
    try:
        print('ws_timing TIMEOUT')
        if chatTimer!= None :
            chatTimer.cancel()
        if wsTimer!= None:
            wsTimer.cancel()
        animation='Emotions/Negative/Bored_1'
        # message='Tempo per la connessione scaduto. Adesso mi riposo.'
        message="Vistu ca nun mi runi cchiu cuntutu. m' astutu."

```

```

        message.encode('ascii','ignore')
        aspeech.say('^start(animations/Stand/'+animation+') '+ message)
        motion_service.rest()
    except RuntimeError:
        print ("Connection timeout error")
        sys.exit(1)

def main(session):
    """
    Dummy main Never executed.
    """

if __name__ == "__main__":
    pepperUrl="172.31.0.100"
    naoQiPort=9559
    serverWsPort=8000
    parser = argparse.ArgumentParser()
    parser.add_argument("--ip", type=str, default=pepperUrl,
                        help="Robot IP address. On robot or Local Naoqi: use
'127.0.0.1'.")
    parser.add_argument("--port", type=int, default=naoQiPort,help="Naoqi port number")
    args = parser.parse_args()

    # start Qi session, proxies and services
    try:
        session = qi.Session()
        session.connect("tcp://" + args.ip + ":" + str(args.port))
    except RuntimeError:
        print ("Can't connect to pepper naoQi")
        sys.exit(1)

    try:
        tts =          ALProxy("ALTextToSpeech",  pepperUrl, naoQiPort)
        aspeech =      ALProxy("ALAnimatedSpeech", pepperUrl, naoQiPort)
        motion=        ALProxy("ALMotion",        pepperUrl, naoQiPort)
        navigationProxy=(ALNavigationProxy",pepperUrl, naoQiPort)
        print ("ALTextToSpeech", "ALAnimatedSpeech", "ALNavigationProxy", "proxies started")

    except RuntimeError:
        print ("Can't start one naoQi proxy")
        sys.exit(1)

    try:
        motion_service = session.service("ALMotion")
        posture_service = session.service("ALRobotPosture")
        print("ALMotion", "ALRobotPosture", "services started")
        #motion_service.wakeUp()

    except RuntimeError:
        print ("Can't start one naoqi service")

```

```
sys.exit(1)

try:
    wsTiming=WsTiming()
    wsTiming.wait()
except RuntimeError:
    print ("ws_timing error")
    sys.exit(1)

# start websocket server
application = tornado.web.Application([ (r'/ws', ws2naoQi),])
try:
    http_server = tornado.httpserver.HTTPServer(application).listen(serverWsPort)
    print ('webSocket server listening at '+pepperUrl+':'+str(serverWsPort)+'/'+ws')
    tornado.ioloop.IOLoop.instance().start()
except RuntimeError:
    print ("Can't start webserver +")
    sys.exit(1)
```


naoQiwS (client javascript, codice dall' interactor)

```
<script type="text/javascript"
src="http://static.robotwebtools.org/EventEmitter2/current/eventemitter2.min.js"></script
>
```

```
<script type="text/javascript"
src="http://static.robotwebtools.org/roslibjs/current/roslib.min.js"></script>
```

.....

```
function connectQi(){
  robotQiWs = new WebSocket('ws://172.31.0.115:8000/ws');
  system.rosNodes.robot.status='_WAIT';

  robotQiWs.onmessage = function(event) {
    system.rosNodes.robot.status='_READY';
    var leng;
    if (event.data.size === undefined) {leng = event.data.length}
    else {leng = event.data.size}
    console.log("onmessage. size: " + leng + ", content: " + event.data);
    message=event.data;
    if (message.includes('speech_END'))popSpeechAct();
  };
  robotQiWs.onopen = function(evt) {
    console.log('Connected to naoQi websocket server. ');
    system.rosNodes.robot.status='_READY';
    interactor_client_socket.emit('robot-status-update', {
      username: 'interactor',
      message: 'robot_ws connection opened',
      status:system.rosNodes.robot.status
    });
  };
  robotQiWs.onclose = function(evt) {
    console.log("naoQi ws connection closed.");
    system.rosNodes.robot.status='_OFF';
    interactor_client_socket.emit('robot-status-update', {
      username: 'interactor',
      message: 'robot_ws connection closed',
      status:system.rosNodes.robot.status
    });
  };
  robotQiWs.onerror = function(evt) {
    console.log("naoQi ws Error! ");
    system.rosNodes.robot.status='_ERROR';
    interactor_client_socket.emit('robot-status-update', {
      username: 'interactor',
      message: 'robot_ws error',
      status:system.rosNodes.robot.status
    });
  };
}
```

Dance.action

```
#goal definition
std_msgs/String params
std_msgs/String feedback
std_msgs/String context
---
#result definition
std_msgs/String status
std_msgs/String message
---
#feedback
std_msgs/String message
```

dance_server.py (ROS actionLibServer)

```
#!/usr/bin/env python
import rospy
import argparse
import actionlib
from std_msgs.msg import String
import json
import amico.msg
from geometry_msgs.msg import Twist
from turtlesim.msg import Pose
from math import pow, atan2, sqrt
from threading import Timer
from time import sleep

import qi
import motion
from naoqi import ALProxy

class DanceAction(object):
    # create messages that are used to publish feedback/result
    _feedback = amico.msg.DanceFeedback()
    _result = amico.msg.DanceResult()
    _duration = 10
    _genre = "samba"
    _feedfreq = 5
    _start_msg = 'start message..'
    _feedback_msg = 'feedback message...'
    _end_msg = 'end message.'

    def __init__(self, name):
        self._action_name = name
```

```

        self._as = actionlib.SimpleActionServer(self._action_name, amico.msg.DanceAction,
        execute_cb=self.execute_cb, auto_start = False)
        self._as.start()

def execute_cb(self, goal):
    print ('ok')
    print(goal.parms.data)
    print(goal.feedback.data)
    print(goal.context.data)

    parms =json.loads(goal.parms.data)
    if 'genre'      in parms.keys():
        self._genre = parms['genre'].encode('ascii','ignore')
    if 'duration'  in parms.keys():
        self._duration =parms['duration']
    if 'feedfreq'  in parms.keys():
        self._feedfreq =parms['feedfreq']

    messages =json.loads(goal.feedback.data)
    if 'start'     in messages.keys():
        self._start_msg  =messages['start'].encode('ascii','ignore')
    if 'feedback'  in messages.keys():
        self._feedback_msg =messages['feedback'].encode('ascii','ignore')
    if 'end'       in messages.keys():
        self._end_msg     =messages['end'].encode('ascii','ignore')

    animation      ='Emotions/Positive/Happy_4'
    animation1     ='Emotions/Positive/Happy_4'
    animation2     ='Emotions/Positive/Hysterical_1'
    animation3     ='BodyTalk/BodyTalk_8'
    animation4     ='BodyTalk/BodyTalk_1'
    animation4     ='Gestures/BowShort_1'

    # start executing the action
    print(self._start_msg)
    aspeech.say('^start(animations/Stand/'+animation1+')'+self._start_msg)
    sleep(0.5)
    aspeech.say('^start(animations/Stand/Emotions/Positive/Happy_4) ^run(animations/S
tand/BodyTalk/BodyTalk_8)^wait(animations/Stand/Emotions/Positive/Hysterical_1)'+self._fe
edback_msg)
    aspeech.say('^start(animations/Stand/Emotions/Positive/Happy_4) ^run(animations/S
tand/BodyTalk/BodyTalk_8)^wait(animations/Stand/Emotions/Positive/Hysterical_1)')
    print(self._feedback_msg)

    # todo controllare se aspeech est asincrona , nel qual caso sottoscrivere con ca
llback ad evento ALMemory di fine dizione ed attendere evento prima di tornare .
    sleep(4) #simulazione ""

    success = True

```

```

    if success:
        self._result.status.data = '_SUCCEEDED'
        self._result.message.data =self._end_msg
        self._as.set_succeeded(self._result)
        print(self._result.status.data+' '+self._result.message.data)
        aspeech.say('^start(animations/Stand/'+animation+') '+self._end_msg)
        sleep(1) # attendere handshake col client prima di chiudere

if __name__ == '__main__':
    pepperUrl="172.31.0.100"
    naoQiPort=9559
    serverWsPort=8000

# start Qi session, proxyes and services
try:
    session = qi.Session()
    session.connect("tcp://" + pepperUrl + ":" + str(naoQiPort))
except RuntimeError:
    print ("Can't connect to pepper naoQi")
    sys.exit(1)

try:
    tts = ALProxy("ALTextToSpeech", pepperUrl, naoQiPort)
    aspeech = ALProxy("ALAnimatedSpeech", pepperUrl, naoQiPort)
    motion= ALProxy("ALMotion", pepperUrl, naoQiPort)
    navigationProxy=("ALNavigationProxy",pepperUrl, naoQiPort)
    print ("ALTextToSpeech", "ALAnimatedSpeech", "ALNavigationProxy", "proxies started")

except RuntimeError:
    print ("Can't start one naoQi proxy")
    sys.exit(1)

try:
    motion_service = session.service("ALMotion")
    posture_service = session.service("ALRobotPosture")
    print("ALMotion", "ALRobotPosture", "services started")
    #motion_service.wakeUp()

except RuntimeError:
    print ("Can't start one naoQi service")
    sys.exit(1)

__node= rospy.init_node('dance_server', anonymous=False)
print('dance action server waiting for goal')
server = DanceAction(rospy.get_name())
rospy.spin()

```

actionLibClient (python)

```
#!/usr/bin/env python
from __future__ import print_function
import rospy
import actionlib
import amico.msg
import time
from std_msgs.msg import String

def dance_client():
    # Creates the SimpleActionClient, passing the type of the action
    # (DanceAction) to the constructor
    client = actionlib.SimpleActionClient('dance_server', amico.msg.DanceAction)

    # Waits until the action server has started up and started
    # listening for goals.
    print ('waiting for server')
    client.wait_for_server()

    # Creates a goal to send to the action server.
    goal = amico.msg.DanceGoal()
    goal.parms = String(data='{"genre":"samba","duration":15,"feedfreq":5}')
    goal.feedback = String(data='{"start":"comincio a ballare","feedback":"sto
ballando","end":"ho finito di ballare"}')
    goal.context = String(data='{"userTask":"ballare"}')

    # Sends the goal to the action server.
    client.send_goal(goal)

    # Waits for the server to finish performing the action.
    client.wait_for_result()

    # Prints out the result of executing the action
    return client.get_result() # A DanceResult
    time.sleep(1)

if __name__ == '__main__':
    try:
        # Initializes a rospy node so that the SimpleActionClient can
        # publish and subscribe over ROS.
        rospy.init_node('dance_client_py',anonymous=False)
        result = dance_client()
        print ('result= ',result.status.data)

    except rospy.ROSInterruptException:
        print("program interrupted before completion", file=sys.stderr)
```

actionLibClient (javascript dal planner)

```
<!--librerie javascript roslib →  
<script type="text/javascript"  
src="http://static.robotwebtools.org/EventEmitter2/current/eventemitter2.min.js"></script  
>  
  <script type="text/javascript"  
src="http://static.robotwebtools.org/roslibjs/current/roslib.min.js"></script>
```

```
// codice di inizializzazione del proxy ROS (websocket) e del topic goal  
ros_ws = new ROSLIB.Ros({url:situation.system.rosNodes.rosBridge.url});  
  situation.system.rosNodes.rosBridge.status='_WAITING';  
  
ros_ws.on('connection', function() {  
  console.log('Connected to ROS websocket server.');  situation.system.rosNodes.rosBridge.status='_READY';  
  interactor_socket.emit('ros-status-update', {  
    username: 'planner',  
    message: 'ros_ws connection opened',  
    status:situation.system.rosNodes.rosBridge.status  
  });  
  initRosTopics();  
  
});  
  
ros_ws.on('error', function(error) {  
  console.log('Error connecting to ROS websocket server: ');  
  situation.system.rosNodes.rosBridge.status='_ERROR';  
  interactor_socket.emit('ros-status-update', {  
    username: 'planner',  
    message: 'ros_ws error',  
    status:situation.system.rosNodes.rosBridge.status  
  });  
});  
  
ros_ws.on('close', function() {  
  console.log('Connection to ROS websocket server closed.');  if(situation.system.rosNodes.rosBridge.status!='_ERROR')  
    situation.system.rosNodes.rosBridge.status='_OFF';  
  interactor_socket.emit('ros-status-update', {  
    username: 'planner',  
    message: 'ros_ws connection closed',  
    status:situation.system.rosNodes.rosBridge.status  
  });  
});  
}  
  
function initRosTopics(){  
  danceGoalTopic = new ROSLIB.Topic({  
    ros : ros_ws,  
    name : '/danceServer/goal',
```

```
messageType : 'dance/DanceActionGoal'  
});
```

// client action server (specializzato per l'action server amico/dance_server che implementa l'Action amico/DanceAction

```
function rosActionServer(task){  
  if(task.label.includes('ballare')){  
  
    var danceClient = new ROSLIB.ActionClient({  
      ros : ros_ws,  
      serverName : '/dance_server',  
      actionName : 'amico/DanceAction'  
    });  
  
    var danceGoal = new ROSLIB.Goal({  
      actionClient : danceClient,  
      goalMessage : {  
        parms:{data:'{"genre":"samba","duration":15,"feedfreq":5}'},  
        feedback:{data:JSON.stringify(task.feedback)},  
        context:{data:JSON.stringify(situation.context.activity)}  
      }  
    });  
  
    danceGoal.on('feedback', function(feedback) {  
      console.log('Feedback: ' + feedback.message.data);  
      task.status='_RUNNING';  
      notifyTaskStatus(task);  
    });  
  
    danceGoal.on('result', function(result) {  
      console.log('Final Result: ' + result.message.data);  
      task.status='_SUCCEEDED';  
      notifyTaskStatus(task);  
      ros_ws.close();  
    });  
  
    log('sending goal to ros dance action server');  
    danceGoal.send();  
  }  
}
```

Descrizione di tipi di task e procedure di esempio

(dal file di configurazione dello scheduler)

```
var todo={
  "tasks":[
    {"type":0,"label":"Reminder
mattutino","priority":"HIGH","planned":30,"max_delay":220,"duration":100,"max_duration":1
50,"command":"° _EXECUTE_TODO"},
    {"type":1,"label":"Questionario Checkup sintomi","priority":"LOW","planned":-
1,"max_delay":200,"duration":30,"max_duration":60,"command":"° _EXECUTE_CHECKUP"},
    {"type":2,"label":"Test","priority":"LOW","planned":-
1,"max_delay":200,"duration":50,"max_duration":200,"command":"° _EXECUTE_TEST"},
    {"type":3,"label":"Sollecito misura
peso","priority":"LOW","planned":300,"max_delay":500,"duration":20,"max_duration":50,"com
mand":"° _SOLICIT ricordati di pesarti"},
    {"type":4,"label":"Alert orario
pillola","priority":"HIGH","planned":50,"max_delay":100,"duration":20,"max_duration":-
1,"command":"° _NOTIFY è ora di prendere la pillola"},
    {"type":6,"label":"Ritorno alla
postazione","priority":"LOW","planned":600,"max_delay":200,"duration":60,"max_duration":-
1,"command":"$ _EXECUTE_BACK_TO_DOCK","feedback":{"start":"Torno alla postazione di
riposo","feedback":"vado","end":"Adesso mi ricarico."}},
    {"type":5,"label":"Startup dalla
postazione","priority":"HIGH","planned":5,"max_delay":400,"duration":20,"max_duration":20
,"command":"$ _EXECUTE_STARTUP","feedback":{"start":"Sono carico, vado alla postazione
di lavoro","feedback":"vado","end":"Sono pronto."}},
    {"type":7,"label":"Registrazione su
FSE","priority":"LOW","planned":500,"max_delay":100,"duration":20,"max_duration":80,"comm
and":"$ _EXECUTE_FSE_LOGGING","feedback":{"start":"Adesso registro i dati nella cartella
sanitaria","feedback":"","end":"Dati registrati."}},
    {"type":8,"label":"Reset dialogo","priority":"HIGH","planned":-
1,"max_delay":10,"duration":10,"max_duration":-1,"command":"$ _RESET"},
    {"type":9,"label":"Test2","priority":"LOW","planned":-
1,"max_delay":100,"duration":30,"max_duration":50,"command":"° _EXECUTE_TEST2"},
    {"type":10,"label":"CHECKUP3","priority":"HIGH","planned":-1,"max_delay":-
1,"duration":20,"max_duration":30,"command":"° _EXECUTE_CHECKUP3"},
    {"type":11,"label":"ballare","priority":"LOW","planned":-1,"max_delay":-
1,"duration":5,"max_duration":80,"command":"$ _STACK §
_BALLARE","feedback":{"start":"Comincio a ballare","feedback":"sto ballando","end":"Ho
finito di ballare."}},
    {"type":12,"label":"accendere","priority":"LOW","planned":-1,"max_delay":-
1,"duration":30,"max_duration":80,"object":"","place":"","command":"$ _QUEUE §
_ACCENDERE","feedback":{"start":"Vado ad accendere","feedback":"sto
andando","end":"Fatto.Ho acceso la luce"}},
    {"type":13,"label":"Mood
Test","priority":"HIGH","planned":100,"max_delay":120,"duration":150,"max_duration":200,"
command":"° _EXECUTE_TEST_MOOD"},
  ],
```



```
"procedures":[
  {"type":100,"label":"Protocollo del Lunedì","tasks":[
    {"type":6,"start":5},
    {"type":8,"start":30},
    {"type":0,"start":45},
    {"type":3,"start":85},
    {"type":4,"start":110},
    {"type":2,"start":135},
    {"type":7,"start":190},
    {"type":5,"start":215},
  ]},
  {"type":101,"label":"Procedura di test",tasks:[
    {"type":5,"start":0},
    {"type":0,"start":2},
    {"type":2,"start":25},
    {"type":6,"start":150},
  ]}
];
```

10 Ringraziamenti

Ringrazio

- l'ing. Ignazio Infantino, ricercatore ICAR Palermo, coautore della modellazione dello scenario e delle policies di interazione affettiva [3]
- gli Ing. Massimo Esposito e Aniello Minutolo, ricercatori dell' ICAR Sede di Napoli, per le stimolanti conversazioni sulla ingegnerizzazione processi gestori di dialogo e sulla introduzione all'utilizzo della piattaforma Watson IBM
- l'ing. Patrizia Ribino, ricercatore ICAR Palermo per i contributi forniti al gruppo di lavoro AMICO ICAR Palermo sulla modellazione dello scenario di assistenza domestica.
- l'ing Pietro Storniolo Tecnologo ICAR Palermo, per i briefing sugli strumenti di programmazione del robot Pepper Aldebaran
- l'ing. Angelo Rubino, borsista ICAR Palermo, che, nell'ambito della attività sperimentale Corso post laurea fi formazione on the job "Applicazioni ICT.SMART-CITIES P.O. FSE 2014/2020. 2014.IT.05.SFOP.014/3/10.4/9.2.10/0017", sotto il mio tutoraggio ha partecipato allo sviluppo della logica di dialogo ed ha configurato i dialoghi degli assistenti Watson HomeAssistant ed HealthAssistant2.
- Vorrei ringraziare inoltre Giampiero Rizzo, collaboratore tecnico di ricerca ICAR, per il supporto sistemico ed i briefing sull'utilizzo del S.O. Ubuntu.

11 BIBLIOGRAFIA

- [1] I. Infantino, A. Machì : Towards an assistive social robot interacting with human patient to establish a mutual affective support WIVACE 2019
- [2] Progetto AMICO: <https://www.icar.cnr.it/progetti/amico-assistenza-medica-in-contextual-awareness/>
- [3] Modello OSI https://it.wikipedia.org/wiki/Modello_OSI
- [4] websocket : <https://it.wikipedia.org/wiki/WebSocket>
- [5] Robot Operating System: <https://www.ros.org/>
- [6] Assistente domestico AMY-Robot * <https://www.robotiko.it/amy-robot/>
- [7] H. Gropss, S. Mueller, C. Shroeter, M. Volkhardt, A. Scheidig, K. Debes, K. Ritcher, N. Doering: "Robot Companion for Domestic Health Assistance: Implementation, Test and Case Study under Everiday Conditions in Private Apartments."
- [8] socket.io :<https://socket.io/>
- [9] Rosbridge: http://wiki.ros.org/rosbridge_suite
- [10] MongoDB <https://docs.mongodb.com/>
- [11] MongoDB Compass <https://www.mongodb.com/products/compass>