Interfacce per End-User Debugging nel contesto Internet of Things

Candidata: Alessia Carosella
Relatori: Fulvio Corno, Luigi De Russis, Alberto Monge Roffarello

INTRODUZIONE

Negli ultimi anni il mondo dell'Internet of Things (IoT) si sta sviluppando sempre più rapidamente, contaminando ogni aspetto della vita sia da un punto di vista personale che industriale. Un sistema IoT comprende dispositivi fisici e applicazioni web che riescono ad interagire tra di loro e con l'ambiente circostante. In passato i sistemi IoT erano programmati da tecnici specializzati ma negli ultimi anni sono sempre più gli utenti finali che vogliono gestire in maniera autonoma i propri dispositivi e applicazioni Web. Per far ciò, sono stati creati dei tool di End-User Development (EUD) che, attraverso l'uso di linguaggi visuali, rendono la programmazione dei sistemi IoT accessibile ad ogni utente. Solitamente la gestione di un sistema IoT è fatta mediante la definizione di regole. Queste regole sono composte secondo il paradigma del Trigger-Action Programming (TA paradigm) dove ad un evento (trigger) è associata un'azione (action). Ogni trigger o action è offerto da un servizio e fornito attraverso un canale. Un esempio di regola TA è:
"*Se entro a casa (Android Location) accendi la lampada in cucina (Philips Hue)*"
dove il trigger è "se entro a casa" ed è offerto da Android Location (applicazione web), mentre l'action è "accendi la lampada" ed è offerta da Philips Hue (dispositivo fisico). Ogni qualvolta un trigger sarà scatenato, il sistema eseguirà l'action specificata.
Creare delle regole trigger-action può apparire molto facile ma l'interazione tra le varie regole create può portare a dei problemi come comportamenti anomali e falle di sicurezza. Per questo motivo le fasi di debug e correzione degli errori risultano fondamentali. Allo stato attuale, però, l'assistenza che gli attuali tool per EUD forniscono durante le operazioni di debug risulta essere carente.

OBIETTIVO DELLA TESI

L'obiettivo di questa tesi è quello di progettare, costruire, e valutare sperimentalmente un tool che possa aiutare un utente finale nel processo di debug di regole trigger-action per il controllo di dispositivi e applicazioni web. La sfida più importante sarà quella di far comprendere ad utenti senza alcuna conoscenza informatica eventuali conflitti che possono nascere dall'interazione di più regole. Questo perché i conflitti legati all'interazione di più regole risultano essere problemi difficili da risolvere e facili da generare. Questi conflitti possono essere principalmente di tre tipi: loop, inconsistente e ridondante. Utilizzando il tool progettato gli utenti potranno comporre regole, comprendere eventuali conflitti che una regola può generare, e decidere di modificare o eliminare le regole per risolvere i problemi segnalati.

ANALISI PRELIMINARE

Il primo passo della tesi è stato quello di effettuare una ricerca in letteratura su tre temi principali: linguaggi visuali, processo di debug (visto dal punto di un utente finale) e metodologie di debug con linguaggi visuali. Durante la ricerca sono stati definiti pro e contro per ogni tipo di linguaggio visuale e sono stati identificati i processi cognitivi compiuti da utenti durante il debug. Sono state inoltre identificate le metodologie e funzionalità utilizzate in altri ambiti di EUD che si occupano di assistere gli utenti in fase di debug. Partendo da queste informazioni sono state definite le linee guida da seguire nel proseguo della tesi che definiscono quali linguaggi visuali utilizzare e quali funzionalità implementare. In particolare, è stato deciso di utilizzare il block-programming con rappresentazione mediante puzzle[1] per la composizione delle regole, e per la descrizione dei conflitti si è deciso di utilizzare in modo congiunto una rappresentazione grafica mediante dataflow e una descrizione testuale che segue i principi dell'Interrogative Debugging[2].

[1] Linguaggio visuale che rappresenta le informazioni sotto forma di pezzi di puzzle con i quali è possibile interagire utilizzando dei meccanismi basati sul drag&drop.

[2] Andrew J. Ko and Brad A. Myers. "Designing the whyline: A debugging interface for asking questions about program behavior", In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* NY, USA, 2004. ACM.

DESIGN E PROGETTAZIONE

Seguendo le linee guida precedentemente definite sono stati creati dei mockup che rappresentano una possibile soluzione per il tool EUD. A partire dai mockup è stato progettato e implementato un tool che assista gli utenti in fase di composizione e debug di regole. Il tool progettato svolgerà il ruolo del client in un'architettura client-server, mostrata in Figura 1, dove il ruolo del server è svolto da un server REST realizzato precedentemente dal gruppo e-Lite[3]. Il server è in grado di memorizzare delle regole e, attraverso l'utilizzo di una rete di Petri, rilevare loop, ridondanze e inconsistenze. L'utente interagendo con l'interfaccia grafica del tool EUD potrà comporre una regola, gestire regole salvate e risolvere i conflitti rilevati.

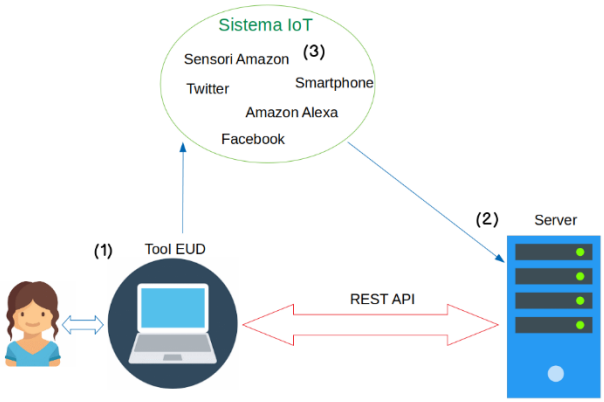

Figura 1: Architettura del sistema. (1) tool EUD; (2) server REST; (3); sistema IoT.

Appena l'utente comporrà una regola il tool, attraverso l'interazione con il server, rileverà se la regola composta genera dei problemi e notificherà all'utente l'esito del controllo già nell'area di composizione attraverso l'uso di colori e spiegazioni testuali. L'utente potrà avere maggiori informazioni sulle cause che hanno generato il problema accedendo all'area di risoluzioni conflitti. In questa area troverà delle spiegazioni testuali e grafiche del problema e avrà inoltre la possibilità di modificare la regola composta per risolvere i problemi segnalati.

IMPLEMENTAZIONE

Il tool EUD creato in questa tesi è un'applicazione web supportata da un server REST. I linguaggi di programmazione utilizzati per l'implementazione dell'applicazione web sono: HTML, CSS e Javascript. Sono state, inoltre, utilizzate le librerie W3.CSS e jQuery. Tutte le rappresentazioni grafiche (puzzle e dataflow) e testuali sono state implementate in Javascript. Le interazioni con il server avvengono attraverso l'uso delle API messe a disposizione e le informazioni sono trasmesse in formato JSON. Le richieste effettuate dal tool, GET e POST, sono fatte in maniera asincrona e per questo motivo è stato necessario implementare dei meccanismi di sincronizzazione attraverso l'uso di flag, variabili condivise e meccanismi di lock.

FUNZIONAMENTO DELL'INTERFACCIA FINALE

L'interfaccia finale del tool EUD implementato è diviso in 2 parti principali: composizione e risoluzione conflitti.

Composizione

L'area di composizione è mostrata in Figura 2. In questa area l'utente potrà comporre delle regole attraverso delle semplici operazioni di drag&drop. L'utente dopo aver scelto il servizio ed il canale desiderato nei menù sulla sinistra potrà trascinare i pezzi (trigger ed action) dal repository all'area di lavoro. Non appena l'utente avrà terminato di comporre una regola riceverà un riscontro dal tool sui possibili problemi generati attraverso un messaggio e dei colori. Il tool fornirà 3 possibili feedback: nessun problema rilevato (verde), rilevato almeno un problema non grave (ridondanza, giallo), rilevato almeno un problema grave (loop o inconsistenza, rosso). Un utente potrà decidere in ogni

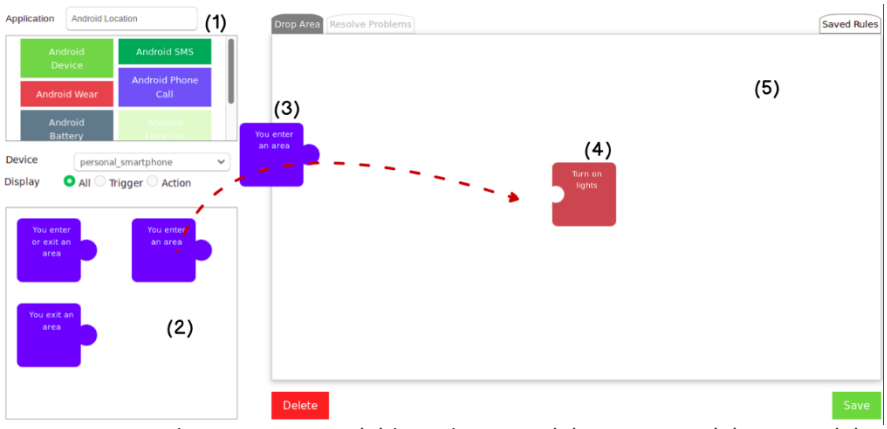

Figura 2: Area di composizione. (1) lista di servizi; (2) repository; (3) trigger; (4) action; (5) area di lavoro.

[3] "https://elite.polito.it/", ultimo accesso: 10/10/2018

momento se salvare o eliminare la regola composta, indipendentemente dai problemi rilevati, utilizzando i pulsanti che si trovano nella parte bassa della schermata.

Risoluzione conflitti

Nell'area di risoluzione dei conflitti (Figura 3) è possibile trovare una descrizione testuale del problema, una descrizione grafica e un'area apposita per modificare la regola composta. Nel caso in cui un utente decida di modificare la regola il tool andrà ad aggiornare in modo automatico ogni campo dell'interfaccia con le informazioni relative alla nuova regola. In questo modo l'utente riceverà un riscontro immediato sulle conseguenze causate dalla modifica apportata. L'area testuale descrive il problema rispondendo alla domanda "*Perché non funziona?*", elencando le azioni svolte dal sistema e utilizzando delle parole chiavi come ad esempio nel caso del loop "*per sempre*". La descrizione grafica del problema è implementata da un dataflow che utilizza come elementi base dei cerchi per rappresentare i trigger, dei quadrati per le action e delle frecce orientate (continue o tratteggiate) per descrivere le relazioni che legano gli elementi del dataflow. Nel caso di inconsistenze e ridondanze è possibile inoltre individuare esplicitamente le regole che hanno effetti contrastanti o ridondanti, attraverso l'uso di un riquadro rosso e un testo che dichiara il tipo di conflitto.

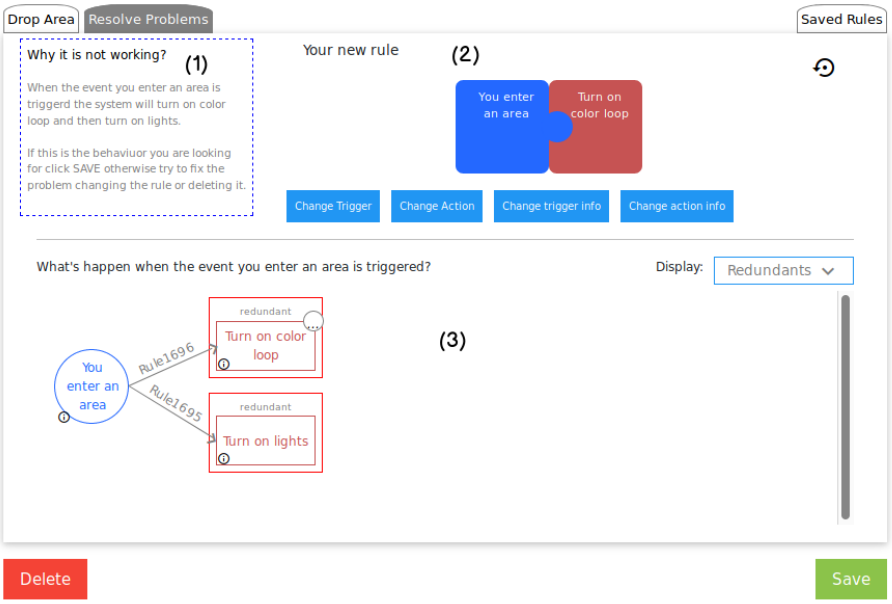

Figura 3: Area di risoluzione dei conflitti. (1) descrizione testuale dei problemi; (2) area di modifica della regola; (3) rappresentazione grafica dei problemi.

VALUTAZIONE CON UTENTI E CONCLUSIONI

Il tool creato è stato valutato in uno studio con 6 utenti (3 uomini e 3 donne). Tutti i partecipanti hanno dichiarato di non avere conoscenze informatiche ed in particolare di non conoscere i concetti di loop, ridondanza e inconsistenza. Nel corso del test ogni partecipante ha composto 12 regole di cui 5 generavano problemi. In particolare, sono state generate 2 inconsistenze, 2 ridondanze e 1 loop. I partecipanti potevano decidere liberamente se salvare, modificare, o eliminare le regole problematiche, fornendo poi una spiegazione delle scelte fatte. Gli utenti hanno deciso di salvare una regola che generava un problema solo nel 10% dei casi (3 volte su 30) riuscendo comunque a fornire una definizione corretta del problema segnalato. Nei casi restanti gli utenti hanno deciso di eliminare una regola conflittuale nel 60% dei casi, mentre hanno deciso di risolvere il conflitto modificando la regola nel 30%. Questi dati dimostrano la bontà del tool realizzato: la maggior parte degli utenti è riuscita ad interagire correttamente con l'interfaccia, comprendere i problemi generati dalle regole e risolverli in modo corretto. In particolare, tutti i partecipanti sono stati in grado di spiegare i problemi generati fornendo delle definizioni corrette per i vari problemi riscontrati. Il tool ha raggiunto quindi ottimi risultati in termini di usabilità e utilità, e questo costituisce un primo passo verso la creazione di un'interfaccia EUD che possa aiutare ogni utente a gestire in modo facile e veloce i propri dispositivi e servizi IoT attraverso la creazione di regole trigger-action corrette. Sono ancora però ancora molti gli aspetti da migliorare. Tra gli sviluppi più importanti è possibile citare l'aggiunta di un'ambiente di simulazione delle regole appena composte e l'estensione a regole più complesse, con trigger e action multiple. Inoltre, come proposto da alcuni partecipanti al test, sarebbe utile ricevere dal tool dei suggerimenti su quali modifiche apportare alle regole per risolvere i problemi segnalati.