ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DISI

LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

TESI DI LAUREA

in

Ingegneria dei Sistemi Software

Ubuntu Phone - Un Sistema Operativo convergente per Desktop e Cellulari

CANDIDATO
Leonardo IANNACONE

RELATORE
Chiar.mo Prof. Antonio NATALI

Anno Accademico 2014/2015

Sessione III

Indice

Capitolo 1

Introduzione

In un mercato per smartphone, dove la suddivisione degli utenti e dei sistemi operativi sembra già ben delineata ormai da diversi anni, Ubuntu, il sistema operativo supportato da Canonical e mantenuto insieme alla comunità Open Source, fa il suo debutto con la versione **Touch**.

Si tratta di un sistema rivoluzionario, che sfrutta a pieno la potenza computazionale dei dispositivi moderni, oggi pari quasi a quella dei laptop. È da questa visione di insieme da cui è nata l'esigenza di unire Desktop e Smartphone in un unico device, aggiungendo un livello di convergenza fra due mondi oggi ancora divisi.

Si tratta di un cambiamento non solo doveroso, ma anche necessario. Un primo passo verso un futuro dove tutti saremo dotati di un mini-computer portatile, appunto uno smartdevice, che porteremo sempre con noi. Poco importa se sarà un dispositivo in grado di effettuare chiamate (smartphone) o più semplicemente un orologio (smartwatch). L'evoluzione dei dispositivi si muove nella direzione degli oggetti intelligenti, in grado anche di comunicare fra di loro, di scambiare informazioni e di adattarsi a condizioni esterne, sono gli smartdevice che costituiscono la Internet of Things, o Internet delle cose. Secondo le stime di Gartner[1], nel 2015 ci saranno 4,9 miliardi di oggetti

[1] 4.9 Billion Connected "Things" Will Be in Use in 2015 http://www.gartner.com/newsroom/id/2905717

connessi a livello globale, circa il 30% in più rispetto al 2014, una proiezione che indica cambiamenti radicali nel nostro modo di vivere. La presenza di oggetti con capacità decisionali e in grado di automatizzare azioni ci regalerà scenari oggi impensabili, come ad esempio il risparmio energetico intelligente, sia a livello individuale (domotica e smart-home) sia a livello macroscopico (smart-city e smart grid).

Ubuntu, negli ultimi anni, ha guardato con particolare attenzione all'evoluzione di questo campo e ha fortemente influenzato lo sviluppo dei propri componenti sulla base delle analisi degli scenari futuri. A conferma di questa tendenza, ha da poco annunciato una nuova versione del sistema operativo chiamata Ubuntu Core, studiata e dedicata appunto all'Internet delle cose.

Informazioni su Ubuntu e Canonical

Ubuntu è una distribuzione GNU/Linux nata nel 2004[2] dalla volontà del fondatore e imprenditore Mark Shuttleworth. Pensata per il mercato Desktop e Server, con milioni di installazioni ogni anno, ha successivamente conquistato il Cloud con una presenza di quasi il 64% nelle istanze OpenStack in produzione[3]. Ha negli ultimi anni dedicato il suo sviluppo verso dispositivi di prossima generazione, rilasciando prima una versione per smartphone e tablet, chiamata Touch, e poi una per l'Internet of Things chiamata Core o Snappy.

Canonical Ltd è l'azienda che supporta attivamente lo sviluppo di Ubuntu e offre soluzioni tecnologiche infrastrutturali e supporto tecnico professionale per i clienti che intendono adottare Ubuntu come sistema operativo a livello aziendale, amministrativo e governativo.

[2]The Ubuntu Story http://www.ubuntu.com/about/about-ubuntu

[3]OpenStack User Survey Insights: November 2014 http://superuser.openstack.org/articles/openstack-user-survey-insights-november-2014

1.1 Lo stato attuale del mercato

L'intero mercato mondiale degli smartphone è cresciuto anno dopo anno fino a raggiungere vette impensabili, circa 335 milioni di unità spedite nel solo terzo quadrimestre del 2014[4]. Per la fine del 2014 si sono stimati circa 1,3 miliardi di dispositivi spediti in tutto il mondo.

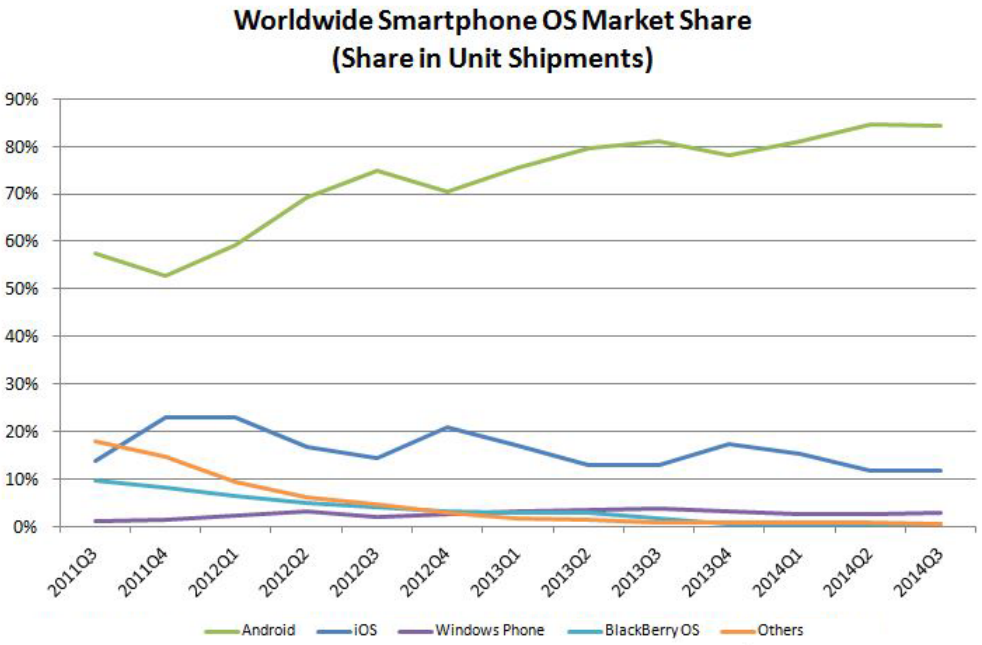
Il mercato è in forte aumento, si parla di una crescita costante di circa il 30% annuo su unità vendute, un trend che non accenna ad invertire. Alla fine del 2014 si è previsto che il numero delle vendite sia considerevolmente aumentato con l'uscita dei dispositivi iPhone 6 e iPhone 6 Plus e, soprattutto, per l'arrivo dei nuovi venditori cinesi che, espandendosi sempre di più oltre oceano, hanno ormai raggiunto e conquistato i mercati europei e statunitensi con le proprie vantaggiose e attrattive offerte di dispositivi low-cost sotto i 200 Euro.

Android con circa 283 milioni di unità spedite e circa l'84% del market share, è il leader indiscusso a livello mondiale dei sistemi operativi. L'entrata in gioco di nuovi venditori cinesi, come Xiaomi e Lenovo, ha costretto Samsung, il leader hardware del mercato, ad abbassare il costo dei propri dispositivi per far fronte alla concorrenza. Di conseguenza, la media dei prezzi di vendita dei device che montano questo sistema operativo si è abbassata nell'ultimo anno di circa il 20%, e ciò ha permesso di mantenere un livello di crescita positivo delle vendite globali.

iOS perde circa un punto percentuale in un solo anno. Una perdita che corrisponde alla richiesta sempre più forte da parte degli utenti di nuovi dispositivi low-cost. Il lancio dell'iPhone 6 e 6 Plus tuttavia potrebbe aver diminuito leggermente la perdita complessiva, facendo aumentare le vendite specialmente nel quarto quadrimestre durante le vacanze natalizie. Tuttavia, i dati di questo periodo non sono ancora disponibili.

Windows Phone con circa il 2,9% del market share non ha mai preoccupato i propri avversari negli ultimi anni. Un trend in stallo dovuto princi-

[4]Fonte IDC International Data Corporation http://www.idc.com/prodserv/smartphone-os-market-share.jsp



Period	Android	iOS	Windows Phone	BlackBerry OS	Others
Q3 2014	84.4%	11.7%	2.9%	0.5%	0.6%
Q3 2013	81.2%	12.8%	3.6%	1.7%	0.6%
Q3 2012	74.9%	14.4%	2.0%	4.1%	4.5%
Q3 2011	57.4%	13.8%	1.2%	9.6%	18.0%

Source: IDC, 2014 Q3

Figura 1.1: Smartphone OS Market Share, Q3 2014

palmente alla limitata gamma di modelli, ad oggi quasi il 90% dei dispositivi venduti con questo sistema operativo sono solo i modelli Nokia Lumia. Tuttavia sono state annunciate delle nuove partnership con venditori importanti come Samsung e HTC che potrebbero portare ad un incremento delle vendite nei prossimi anni.

1.2 Ubuntu Touch

Nel gennaio del 2013 è stato annunciato da Canonical Ubuntu Touch, il sistema operativo Open Source per smartphone e tablet. In realtà non si tratta di una modifica ad-hoc per dispositivi mobili, quanto di un vero e proprio sistema operativo convergente fra sistemi eterogenei.

Figura 1.2: La convergenza di Ubuntu fra dispositivi eterogenei

Ubuntu Touch si colloca in un mercato già saturo, ormai da anni conquistato da Android e da iOS, tuttavia nei suoi primi mesi di vita ha conquistato una curiosità globale che, secondo i dipendenti Canonical, ha già portato più di 10.000 utenti unici ad installare Ubuntu Phone prima del rilascio ufficiale[5].

Di fatto, la curiosità è più che legittima in quanto Ubuntu Touch non è un adattamento di un sistema conosciuto verso dispositivi nuovi, ma una nuova piattaforma che parte da una nuova idea di convergenza, portando un unico sistema su più dispositivi, dallo smartphone al Desktop, dalla TV all'auto intelligente.

[5]10,000 Users of Ubuntu Phone http://discourse.ubuntu.com/t/10-000-users-of-ubuntu-phone/1729

Questa astrazione è possibile grazie ad Unity, l'interfaccia utente di Ubuntu, che venne rilasciata per la prima volta nel 2010 e che, fin dai primi momenti, venne pensata e sviluppata con il concetto di mobile-devices-in-mind. Dopo circa quattro anni di sviluppo, messa appunto e refactory, Unity con la versione 8 è finalmente pronta per la produzione. Non solo un'interfaccia per sistemi Desktop, quindi, ma soprattutto per sistemi mobili, grazie soprattutto all'introduzione del touch e di nuovi widget che hanno reso le applicazioni portabili e completamente interscambiabili fra diversi dispositivi grafici, in maniera del tutto trasparente, appunto convergenti.

Diversamente dalle altre piattaforme in commercio, Ubuntu Touch segue un sistema di sviluppo e di rilascio aggiornamenti diverso, sfruttando le tempistiche in uso già ora per il sistema operativo di casa Canonical e più in generale dell'Open Source: gli aggiornamenti arrivano non appena disponibili, non solo a livello applicativo, ma anche di sistema. Nella maggior parte dei casi, chi acquisterà uno smartphone con sistema operativo Ubuntu non soffrirà quindi dei ritardi degli aggiornamenti che caratterizzano oggi i più famosi produttori di dispositivi.

1.3 L'esigenza di unire Desktop e Smartphone

> *When we began developing Unity a few years ago, the aim was to create a single family of interfaces that work the same way on different devices. This means that unlike most of our rivals, we are able to use a single underlying OS across all the devices which people use, be they PCs, phones or any other device.*
>
> Jane Silber, CEO di Canonical, 2 gennaio 2013[6]

[6]It's official: Ubuntu now fits phones http://blog.canonical.com/2013/01/02/its-official-ubuntu-now-fits-phones/

Nella vision di Unity è sempre stata presente l'integrazione fra Desktop e smartphone, dispositivi mobili che oggi sono sempre più potenti e con prestazioni hardware che ormai assomigliano quasi a quelle di laptop. Quasi tutti gli sviluppatori di sistemi per smartphone hanno tentanto questo approccio di convergenza, ma ad oggi nessuno è ancora riuscito a presentare un prodotto pronto per il mercato, nessuno tranne Ubuntu e Canonical. Unity 8, attraverso Mir[7] (il nuovo server grafico di Ubuntu che sostituirà il pluridecennale X[8]) è pronto per rendere quest'integrazione reale.

Nella vision di Unity c'è il completo interscambio fra ambiente Desktop e smartphone, un caso d'uso possibile tramite l'utilizzo di una docking station che collega il dispositivo mobile ad un monitor esterno.

Figura 1.3: Convergenza fra Desktop e Smartphone

Questo permette all'utente non solo di avere gli stessi dati sempre a disposizione, senza l'utilizzo di software di terze parti per la sincronizzazione (cloud, ad esempio), ma soprattutto permette l'utilizzo delle stesse applicazioni che, grazie al livello di astrazione offerto dall'interfaccia, si adeguano automaticamente alla grandezza del display e ai nuovi dispositivi di input.

Questo risultato è stato possibile abbandonando le librerie grafiche GTK+[9], sulle quali è basato il Desktop Environment GNOME[10] finora utilizzato da

[7]Mir, the next generation display server http://unity.ubuntu.com/mir/
[8]X Widnows System http://www.x.org/wiki/
[9]The GTK+ Project http://www.gtk.org/
[10]GNOME Project Homepage http://www.gnome.org/

Ubuntu, in favore delle librerie Qt[11], framework assai più moderno e pensato per velocizzare lo sviluppo di codice operativo in ambienti eterogenei.

Lo sviluppatore che sceglie Ubuntu come piattaforma per la propria applicazione è quindi doppiamente invogliato: se da una parte ha un framework grafico nuovo e potente, dall'altra ha la possibilità di scrivere una singola applicazione che viene resa automaticamente disponibile sia per la versione Desktop che per quella smartphone, in maniera trasparente e direttamente dal sistema, abbattendo quindi i costi di produzione e di manutenzione del codice di progetti semi-cloni.

1.4 Il primo dispositivo con Ubuntu Touch

Dopo quasi un anno dall'annuncio della collaborazione fra Canonical e Bq[12], nel febbraio del 2015[13] è stato messo in vendita il primo smartphone con Ubuntu Touch pre-installato: il **BQ Aquaris E4.5**[14].

Il dispositivo ha dimesioni 137 x 68 x 9 mm e un peso di soli 123 grammi. È dotato di due slot per l'inserimento di micro-SIM per poter usufruire di più numeri telefonici contemporaneamente. Ha una fotocamera da 8 megapixel Full HD 1080p con autofocus e doppio flash. Il display è da 4.5 pollici. Monta un processore Quad Core Cortex A7 da 1.3 GHz e RAM da 1 GB. Una memoria interna da 8 GB con uno slot esterno per inserire MicroSD fino a 32 GB. Il prezzo attuale è pari a 170 Euro ed è venduto solo nel mercato europeo.

Ulteriori dispositivi sono attesi nell'immediato futuro.

[11] Qt Project homepage http://qt-project.org/

[12] Canonical announces first partners to ship Ubuntu phones https://insights.ubuntu.com/?p=5224

[13] BQ's new Aquaris E4.5 Ubuntu Edition – the smartphone that puts content and services at your fingertips https://insights.ubuntu.com/?p=9567

[14] Aquaris E4.5 Ubuntu Edition http://www.bq.com/gb/ubuntu.html#aquaris-e4-5

Figura 1.4: Anteprima di Bq Aquaris E4.5 Ubuntu Edition

Informazioni su Bq

Bq è un'azienda spagnola che produce componenti elettronici e sviluppa software. È fra le principali aziende a livello europeo nell'ambito tecnologico, con una presenza in circa 40 Paesi. Opera principalmente nel settore dei dispositivi multimediali, stampanti 3D e robotica per l'educazione, annoverata per essere una delle compagnie più vicine alla filosofia del DIY (Do It Yourself) e dell'Open Source.

1.5 Ubuntu Core nell'Internet of Things

Il fondatore di Canonical, Mark Shuttleworth, nel dicembre del 2014 ha annunciato Ubuntu Core[15], una nuova versione di completa rottura con la tradizione di Ubuntu.

Ubuntu Core[16] si allontana completamente dall'architettura base che negli ultimi 10 anni ha caratterizzato questo sistema operativo, e introduce un nuovo concetto di applicazioni chiamate Snappy che operano in contesti isolati, aumentando quindi la sicurezza e la modularità dell'intera distribuzione.

Dal punto di vista del codice e delle librerie messe a disposizione, Ubuntu Core risulta essere nient'altro che un'immagine notevolmente ridotta della versione Server[17] e richiede pochissimo spazio per l'installazione, dal punto di vista architetturale corrisponde invece ad una naturale evoluzione di Ubuntu Touch, più lontana dal concetto di telefonino e più vicina a quella di smartdevice.

Proprio grazie alla sua limitata dimensione, alla grande modularità e alla astrazione del dispositivo fisico introdotta dalla nuova architettura, Ubuntu Core si presta bene alle esigenze dell'Internet of Things.

1.5.1 Architettura Snappy

L'architettura Snappy è costituita da tre livelli principali: il livello di Sistema Operativo (insieme dei driver e dell'astrazione hardware del dispositivo), il livello Framework (l'ambiente applicativo e l'insieme delle librerie disponibili per un applicazione) e il livello Application (livello applicativo, comunemente accorciato in App) .

Le applicazioni sono eseguite quindi su un ambiente personalizzato e disgiunto dal resto del sistema. In questo modo è possibile eseguire la stessa applicazione in configurazioni di dispositivo diverse e/o con librerie diverse

[15] Announcing Ubuntu Core, with snappy transactional updates! http://www.markshuttleworth.com/archives/1434

[16] Snappy Ubuntu https://developer.ubuntu.com/en/snappy/

[17] Ubuntu Server http://www.ubuntu.com/server

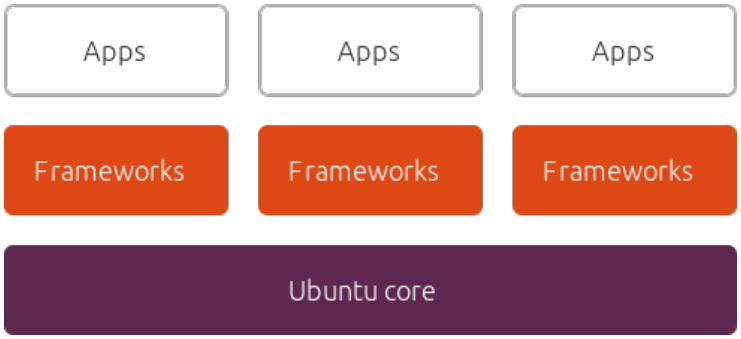


Figura 1.5: Architettura delle applicazioni Snappy

per testarne a pieno il funzionamento, e anche avere sullo stesso dispositivo applicazioni che altrimenti fra loro sarebbero completamente incompatibili.

La suddivisione in livelli aumenta anche la sicurezza dell'intero sistema, un'applicazione maligna non può attaccare il funzionamento di altre se non sono nel proprio Framework, perché completamente confinata. Anche l'aggiornamento del sistema si rivoluziona completamente, con la possibilità di creare dei framework di transizione dove applicare i cambiamenti prima di inserirli nel sistema, cosicché se qualcosa non va a buon fine il sistema non è mai in uno stato inconsistente.

1.6 Outline

In questo documento si studierà lo stato dell'arte di Ubuntu Touch, se ne analizzerà la piattaforma e i suoi componenti principali come il nuovo server grafico, l'interfaccia utente Unity e il nuovo formato dei pacchetti Click.

Si discuteranno la struttura, il comportamento e l'interazione delle applicazioni, sia quelle sviluppate utilizzando il linguaggio QML, sia quelle in HTML5 e degli Scope, i motori di ricerca innovativi di Ubuntu.

Si stenderanno informazioni dettagliate di come vengono gestiti gli aggiornamenti di sistema, con particolare attenzione all'implementazione dei vari componenti, come ad esempio il server, il client e di come sono gestiti i canali e la promozione delle immagini.

Infine, si mostrerà quanto semplice sia installare il sistema operativo su uno dei dispositivi ufficialmente supportati e quanto veloce sia la produzione di nuovo codice e di applicazioni attraverso Ubuntu SDK. Per dimostrare ciò si è sviluppato un nuovo Scope, una delle tecnologie innovative di Ubuntu Touch, che effettua ricerche e verifica la disponibilità dei volumi nel database della Biblioteca Universitaria Dore della Facoltà di Ingegneria di Bologna.

Capitolo 2

La piattaforma

Ubuntu Touch è, come tutte le piattaforme moderne, suddiviso in due super blocchi concettuali, uno per il kernel e uno per lo userspace.

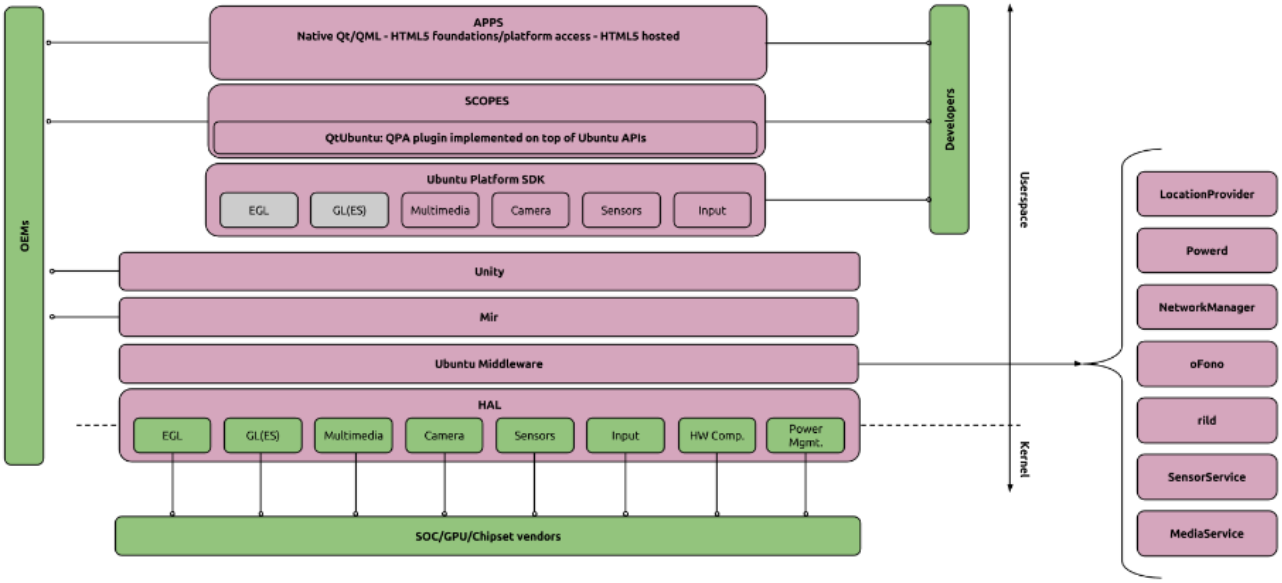


Figura 2.1: Schema della piattaforma di Ubuntu Touch

A livello kernel si distinguono due blocchi interessanti, l'Hardware Abstraction Layer (HAL) e l'Ubuntu Middleware che offrono alle strutture di

più alto livello un astrazione del dispositivo fisico e delle sue periferiche. È utile far notare che nell'Ubuntu Middleware sono presenti parti di codice Android per l'accesso ai vari device.

A livello userspace si trovano invece tutti le parti che contraddistinguono Ubuntu Touch:

- **Mir** - il server grafico sviluppato ad-hoc per gestire nuovi tipi di input
- **Unity** - l'interfaccia grafica, detta anche Shell
- **Ubuntu Platform SDK** - che offre, attraverso API, l'accesso ai servizi di sistema e alle periferiche hardware
- **Scopes** - servizi per la ricerca di contenuti
- **Apps** - le applicazioni standard

2.1 Il server grafico Mir

Mir è un componente a livello di sistema sviluppato per sostituire il vecchio server grafico X in dispositivi mobili e Desktop.

L'intero progetto si divide in due grandi librerie:

- *libmiserver* - contiene i componenti lato server di Mir, usata per implementare un compositor e per far comunicare il sistema con la scheda video e i display
- *libmirclient* - contiene le API per far comunicare le applicazioni con i server Mir, usata dai toolkit grafici

2.1.1 Struttura interna

Ad oggi la struttura interna di Mir è formata dai seguenti componenti[1]:

[1]Mir specifiche https://wiki.ubuntu.com/Mir/Spec

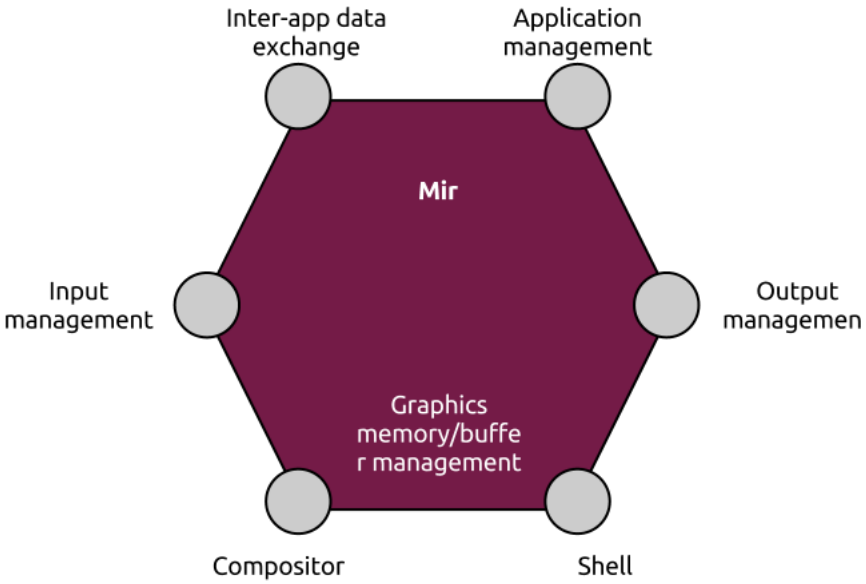


Figura 2.2: Struttura interna di Mir

- **Compositor** Genera il buffer finale da mostrare a display e applica le trasformazioni (geometriche, ombre) alle surface (finestre) conosciute.

- **Input managment** Inoltra gli eventi dai dispositivi di input al display server e alle applicazioni/client in ascolto.

- **Inter-app data exchange** Si occupa della gestione della *clipboard* e delle funzionalità di *drag'n'drop*.

- **Application managment** Associa ad ogni surface/finestra l'applicazione relativa.

- **Ouput managment** Gestisce i monitor.

- **Shell** Implementa le funzionalità di gestione delle finestre, disegna gli elementi dell'interfaccia utente (dash, launcher, ...) e gestisce gli eventi di sistema dei dispositivi di input.

2.2 Unity

Sin dal 2010, quando Unity venne creata, uno dei requisiti principali del progetto era la convergenza fra sistemi eterogenei, affinché l'interfaccia fosse in grado di auto adattamento su display di diverse dimensioni. Questo concetto, con il passare del tempo, è diventato sempre più forte, fino a ad essere considerato la parte chiave del progetto con l'annuncio di Ubuntu Touch.

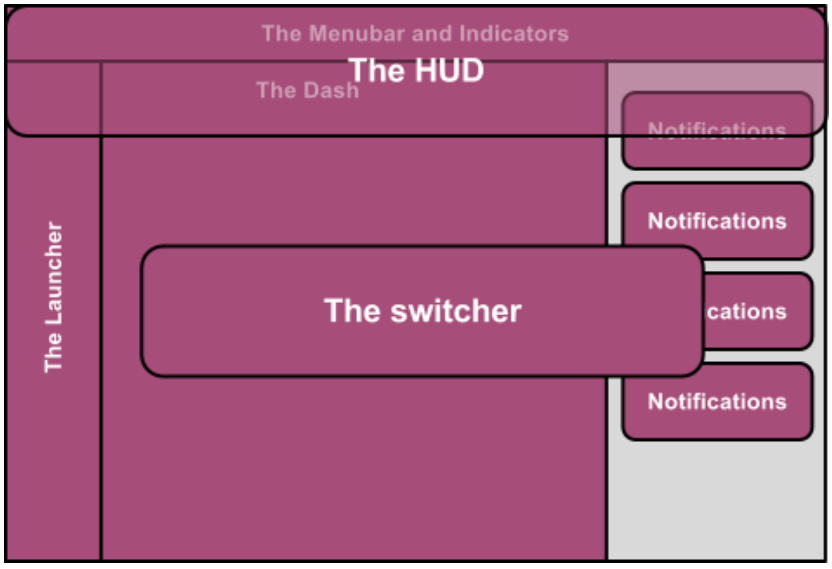


Figura 2.3: Diagramma concettuale della Shell, l'interfaccia utente di Unity

Per questo motivo, il codice è stato completamente rivisto ed è stato creato un progetto parallelo chiamato Unity8 che rappresenta l'evoluzione logica di Unity, creata prima per i Desktop, adattata (per un breve periodo di tempo) alla TV ed infine propagata ai display di smartphone e tablet.

2.2.1 Proprietà e requisiti

Fra i requisiti ultimi di Unity8, che si sono poi trasformati in codice e quindi proprietà uniche di Unity, troviamo[2]:

[2]Specifiche di Unity8 https://wiki.ubuntu.com/UnityNextSpec

- Facilità di adattamento a diverse forme e grandezze di monitor, con ottimizzazione dell'interfaccia in base alle proprietà del dispositivo
- Continuità di design attraverso dispositivi diversi, presentando all'utente sempre la stessa consistente interfaccia
- Efficienza e utilizzo del minimo numero di risorse, per far fronte alla povertà di risorse e di potenza di calcolo degli attuali dispositivi mobili
- Essere indipendente dal dispositivo utilizzato, e non dipendere da specifiche proprietà hardware di un dispositivo, come periferiche di input particolari (ad esempio touchscreen) così da garantire una facile transizione fra ambienti di esecuzione diversi

2.2.2 Architettura di sistema

Unity è essenzialmente un controllore di applicazioni e gestore di eventi. Gli eventi esterni possono essere originati da periferiche, temporizzatori, animazioni o da altre parti del sistema.

Gestione delle applicazioni

Unity estende la nozione classica di gestore di finestre, elevandolo ad concetto di contenitore di istanze di applicazioni, le quali possono avere zero o più finestre contemporaneamente. Ridefinisce anche il concetto di finestra, come surface (superficie) sullo schermo, capace di ricevere eventi di input. Unity si trasforma e diventa pertanto un gestore di applicazioni, più che di finestre. Per permettere questo, Unity implementa nuove funzionalità per:

- rimuovere e assegnare il focus alle applicazioni
- terminare un'istanza di un'applicazione
- avviare una nuova istanza di un'applicazione

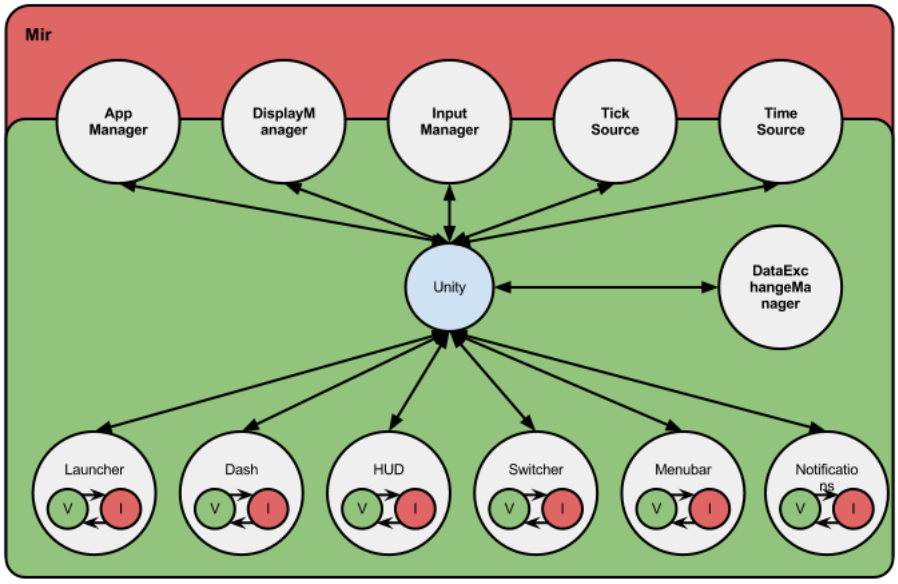


Figura 2.4: Architettura di Unity

Eventi di input

Unity intercetta gli stream di eventi prima che questi vengano inoltrarti alle applicazioni, li gestisce al meglio per avere il totale controllo dei componenti a livello di sistema.

Gestione del Display

Unity è responsabile della gestione dei monitor (singoli, multipli, cloni, ...). Il componente che se ne è occupa è chiamato DisplayManager ed è responsabile di rilevare cambiamenti di stato nella configurazione dei monitor, in modo da permetter l'auto adattamento della Shell.

Tic unificato e tempo interno

Unity fa pesante uso di animazioni e transizioni di stato per creare una piacevole e reattiva interfaccia utente. Per assicurarsi che le animazioni siano

perfettamente sincronizzate con il resto del sistema è stato introdotto un tic o tempo interno condiviso in tutto il sistema.

2.2.3 Tecnologie usate e integrazione nel sistema

Unity è stata sviluppata utilizzando Qt (librerie grafiche) e QML (linguaggio di programmazione dichiarativo basato su JavaScript) [3] per la creazione degli effetti grafici, le quali utilizzano a loro volta lo stack OpenGL[4].

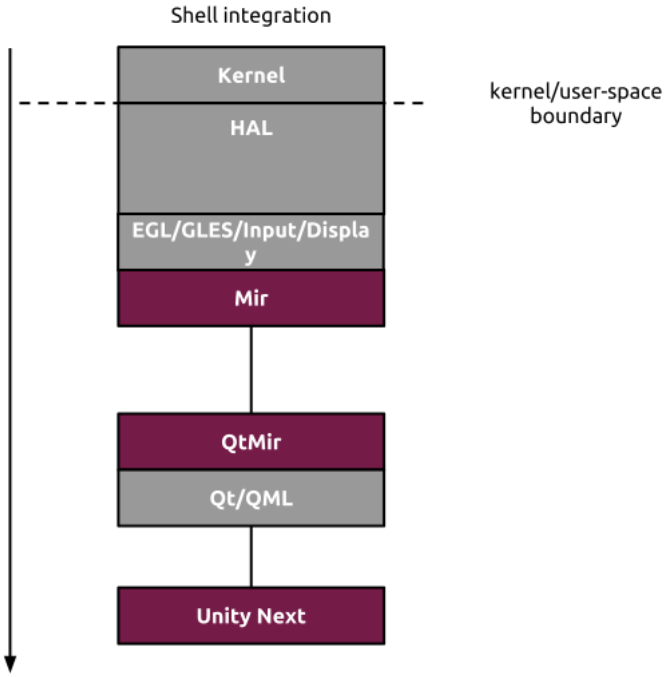


Figura 2.5: Integrazione di Unity in Ubuntu Touch

Per la gestione degli eventi invece si è scelto Mir, il server grafico, che si comporta come un *abstraction layer* con l'hardware sottostante, permettendo a Unity di adattarsi facilmente a dispositivi diversi.

[3]Qt Modeling Language http://doc.qt.io/qt-5/qmlapplications.html
[4]OpenGL homepage https://www.opengl.org/

2.2.4 Personalizzazioni e punti di estensioni

Unity offre due principali punti di estensione per sviluppatori:

- indicatori personalizzati per permettere accesso a particolari funzionalità del sistema
- Scope personalizzati per mostrare contenuti specifici e custom all'utente

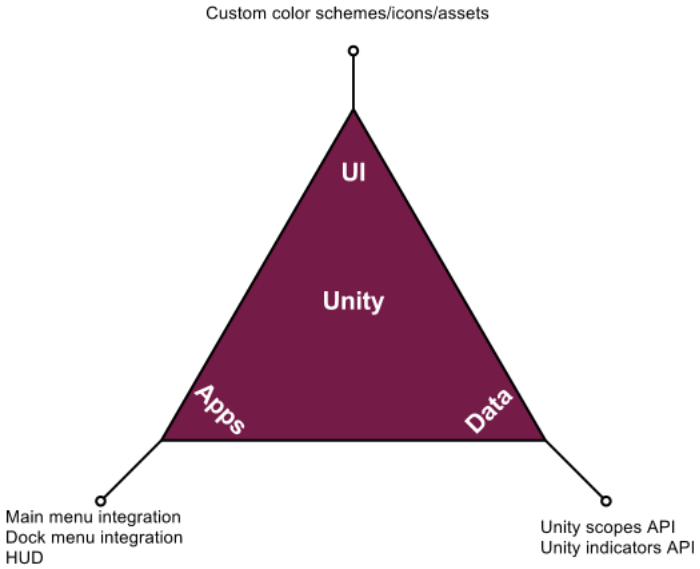


Figura 2.6: Punti di estensione in Unity

API per gli indicator

Gli indicator in Unity sono delle entità grafiche che rappresentano gli stati di un applicazione o di un sottosistema (ad esempio, lo stato della rete). Gli indicator possono essere sia icone che etichette, e possono avare un menu associato. Il menu presenta delle voci che danno informazioni sullo stato dell'applicazione o offrono azioni per cambiarne lo stato (ad esempio, selezionare una rete WiFi a cui connettersi). Per questo motivo, la logica di un indicator risiede direttamente nella sua applicazione. È utile far notare

che, a seconda del display in uso, gli indicator potrebbero occupare posizioni diverse in Unity.

API per gli Scope

Uno Scope in Unity è un entità che rende determinati dati sia ricercabili che accessibili direttamente dalla Dash. Questi dati possono essere di natura diversa, come musica, video, fotografie, applicazioni, condizioni meteorologiche, sia presenti fisicamente sul proprio dispositivo, sia disponibili in Internet. Per questo motivo, l'interfaccia di uno Scope è pensata per essere in grado sia di accettare una stringa come parametro di ricerca, sia di restituire il risultato sotto forma di una struttura dati conosciuta a Unity. È utile far notare che più Scope possono essere attivi contemporaneamente durante una ricerca.

2.2.5 Modello base dell'interfaccia

Un'anteprima del modello base dell'interfaccia utente è riportata in Figura 2.7. Gli indicator e la barra del menu sono accessibili dal lato superiore. Fare swipe sul lato destro del dispositivo riporta in primo piano l'applicazione precedente. I controlli dell'applicazione si trovano sul lato inferiore del dispositivo. Per lanciare le applicazioni o visualizzare la Dash è sufficiente fare swipe sul lato sinistro.

2.3 Click - un nuovo formato dei pacchetti

In Ubuntu Touch è stato sviluppato un nuovo formato per i pacchetti con il nome di Click[5]. Questo formato specifica come le applicazioni sono scaricate sui dispositivi, come vengono *pacchettizzate* e come si installano attraverso i package manager.

[5]Click documentazione ufficiale https://click.readthedocs.org/en/latest/

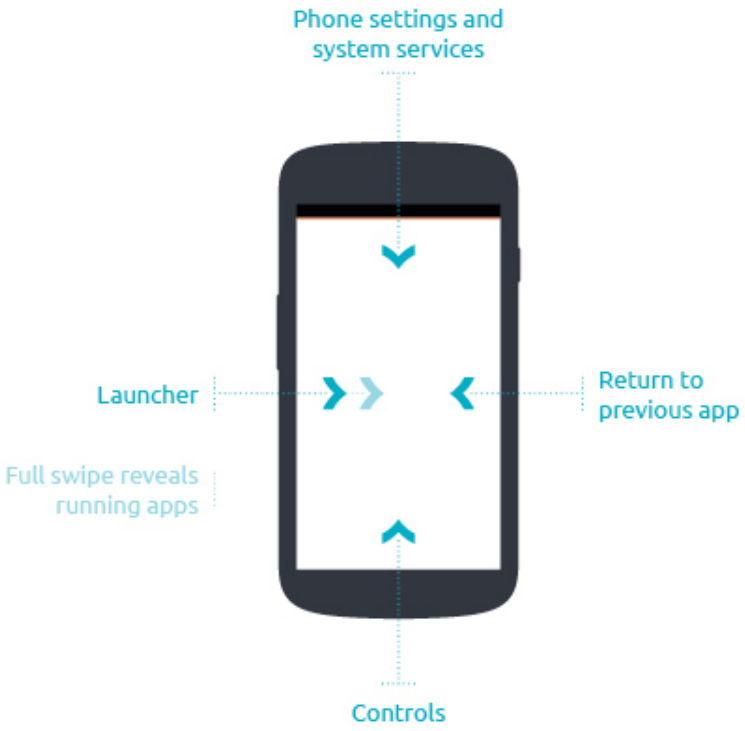


Figura 2.7: Modello base dell'interfaccia utente

Click è stato sviluppato per facilitare la creazione di pacchetti e per semplificare la gestione degli archivi, in modo da garantire facile scalabilità dei server principali. Click controlla anche che durante la fase di installazione il pacchetto non esegui, con i privilegi di root, operazioni non verificate sul sistema ospitante. Nella versione 0.4 le specifiche aggiungono la possibilità di includere nel pacchetto del codice di terze parti. Click è basato su dpkg[6], per ulteriori informazioni sulla creazione di pacchetti deb si rimanda alla documentazione ufficiale di Debian[7].

Il pacchetto risultante dalla compilazione è un archivio di tipo **ar**[8] che a

[6]Debian Package http://it.wikipedia.org/wiki/Dpkg

[7]Debian - Guida per il nuovo Maintainer https://www.debian.org/doc/manuals/maint-guide/

[8]ar command line http://unixhelp.ed.ac.uk/CGI/man-cgi?ar

sua volta contiene archivi di tipo **tar** con file di controllo e di dati.

2.3.1 Il file control

Essendo basato su dpkg, click estende i file di controllo di quest'ultimo. In aggiunta al file `control` classico di Debian, è necessario specificare ulteriori informazioni utili alla compilazione del pacchetto in Click, come il campo `Click-Version` che dichiara la versione di Click compatibile con il file.

Altri campi sono automaticamente copiati dal file manifest per facilitare l'interoperabilità e la retro-compatibilità con vecchi package manager basati su dpkg.

2.3.2 Il file manifest

Il file manifest è un nuovo file (JSON UTF-8) aggiunto all'architettura dpkg nelle specifiche di Click.

Campi obbligatori

Il file `manifest.json` è obbligatorio nei pacchetti sorgenti e deve includere necessariamente le seguenti chiavi:

- **name**: nome unico dell'applicazione
- **version**: numero di versione dell'applicazione
- **framework**: la piattaforma di sistema per il qualche il pacchetto è stato compilato
- **installed-size**: la grandezza del pacchetto dopo la decompressione (questo campo viene generato automaticamente durante la compilazione)

Sono considerati pacchetti corrotti tutti quelli che non presentano una di queste voci o che ne hanno valore incompatibile.

L'implementazione dei package manager deve rifiutarsi di processare pacchetti che dichiarano un framework diverso da quello del sistema in esecuzione. Esempi di framework validi sono "ubuntu-sdk-14.10-qml" o "ubuntu-sdk-15.04-html".

Il valore del nome identifica univocamente l'applicazione e segue le regole dei nomi nei pacchetti Debian[9]. Il nome è unico in tutto l'ecosistema Ubuntu e per la sua scelta è suggerita la convezione usata per i pacchetti Java, ad esempio "*com.miodominio.miaapp*".

Il valore della versione fornisce un numero di versione unico per l'applicazione e segue le regole di versioning di Debian, per maggiori informazioni si rimanda alla documentazione ufficiale[10].

Campi opzionali

Il file manifest può contenere campi opzionali, utili alla descrizione del pacchetto:

- **title**: breve titolo (su una riga) dell'applicazione.
- **description**: descrizione estesa (multi riga) dell'applicazione.
- **maintainer**: nome e indirizzo email del mantenitore del pacchetto.
- **architecture**: una stringa (sono permessi valori multipli) per l'architettura con cui l'applicazione è compatibile. Il valore "all" indica tutte le architetture.
- **hooks**: routine di sistema necessari in fase di installazione, aggiornamento e rimozione del pacchetto, per maggiori informazioni si rimanda alla documentazione ufficiale[11].

[9]Policy dei nomi dei pacchetti Debian https://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-Source

[10]Policy delle versioni dei pacchetti Debian https://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-Version

[11]Hooks in Click https://click.readthedocs.org/en/latest/hooks.html

- **icon**: nome dell'icona o percorso al file da usare per mostrare il pacchetto nei package manager.

Capitolo 3

Le applicazioni

L'ecosistema applicativo di Ubuntu Touch è in costante crescita. Giorno dopo giorno, un numero sempre maggiore di nuove applicazioni viene sviluppato e reso pubblico agli utenti. Ubuntu è già completo di tutte le applicazioni basi, necessarie all'utilizzo quotidiano del sistema, applicazioni sviluppate (interamente o sotto la supervisione) dagli ingegneri Canonicali e testate dalla comunità internazionale.

In Ubuntu esistono applicazioni di due tipi, HTML5 o native. Le applicazioni HTML5 sfruttano le diffusissime tecnologie web per creare software completi e si inseriscono perfettamente nel design della piattaforma Ubuntu. Le applicazioni native sono invece sviluppate in QML, un nuovo meta-linguaggio di programmazione, e hanno il vantaggio di avere un totale controllo dell'interfaccia grafica e dell'accesso al sistema rivelandosi la scelta ideale ad esempio per giochi e applicazioni ad-hoc.

Guide linea per lo stile A chi sceglie di sviluppare in Ubuntu è chiesto di seguire le principali linee guide[1] per il Design e lo stile, questo per garantire continuità grafica fra il sistema base e le applicazioni di terze parti.

[1]Ubuntu App Design Vision https://design.ubuntu.com/apps/get-started/ui-model

Pubblicare il proprio lavoro Tutte le applicazioni sono pubblicate e rese disponibili agli utenti tramite il Software Store di Ubuntu. Per maggiori informazioni sulla pubblicazione di nuovi programmi si rimanda alla guida ufficiale[2].

3.1 Ciclo di vita delle applicazioni

Il ciclo di vita delle applicazioni in Ubuntu è stato pensato per essere semplice, sicuro ed efficiente a fronte di una migliore gestione dei consumi energetici.

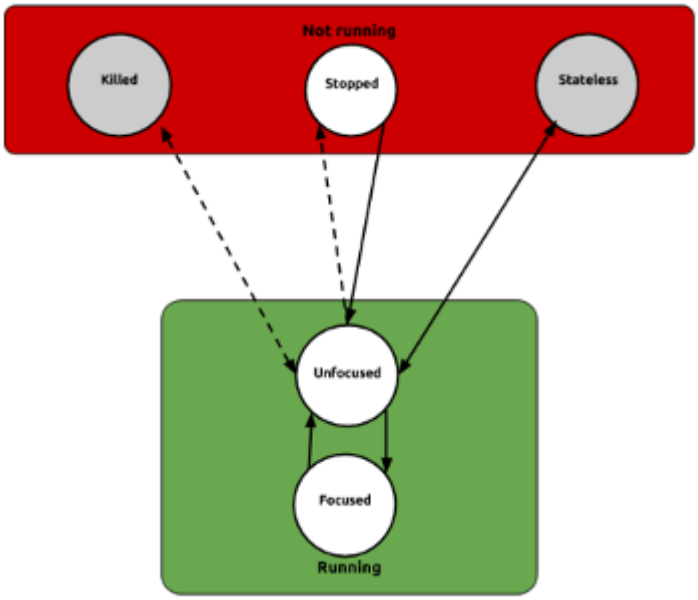


Figura 3.1: Ciclo di vita di un'applicazione in Ubuntu Touch

Gli stati in cui un'applicazione può trovarsi sono:

Focused L'applicazione è certamente in esecuzione, visibile all'utente e ha accesso a tutte le risorse necessarie.

[2]Pubblicare Applicazioni su Ubuntu https://developer.ubuntu.com/en/publish/

Unfocused L'applicazione può non essere in esecuzione (ad esempio, è in attesa che gli venga assegnata la CPU). Nello scenario di dispositivi smartphone, l'applicazione non è visibile all'utente.

Killed Il processo dell'applicazione è stato rimosso dalla memoria principale.

Stopped L'applicazione ha ricevuto un segnale di SIGSTOP ed è in pausa.

Stateless L'applicazione è in attesa di rimozione o di un reset dopo aver ricevuto un segnale di SIGKILL e il suo processo non è più in un stato consistente.

Per ulteriori informazioni sulla gestione dei consumi energetici e sul ciclo di vita delle applicazioni si rimanda al Blueprint relativo[3].

3.2 Accesso alle risorse e scambio di contenuti tramite Content Hub

Tutte le applicazioni accedono al sistema utilizzando **AppArmor**[4], un modulo di sicurezza che confina le applicazioni ad un limitato set di risorse. AppArmor è un sistema di Mandatory Access Control (MAC) inserito nel kernel di Linux con una gestione dello controllo degli accessi più orientata ai programmi che agli a utenti (in Apple viene usato un modulo simile chiamato XNU Sandbox[5]).

Ogni applicazione dichiara un profilo di configurazione per AppArmor basato sulle impostazione del proprio Manifest (per ulteriori informazioni si rimanda alla pagina Wiki dedicata[6]).

[3] Add app-model and -lifecycle to platform api https://blueprints.launchpad.net/ubuntu/+spec/client-1303-add-app-model-and-lifecycle-to-platform-api

[4] AppArmor homepage http://apparmor.net/

[5] Douglas Comer, Operating System Design – The XINU approach, Prentice-Hall, 1984, ISBN 0-13-637539-1, https://www.cs.purdue.edu/homes/dec/osbooks.html

[6] Manifest file - security https://wiki.ubuntu.com/SecurityTeam/Specifications/ApplicationConfinement/Manife

Le applicazioni hanno diritto di lettura e scrittura solo nelle proprie cartelle. Per permettere lo scambio e la condivisione di contenuti fra applicazioni diverse si fa uso di un servizio chiamato **Content Hub**[7].

In Ubuntu for Phone, ogni singolo contenuto ha uno e un solo proprietario, non esiste ad esempio un applicazione che ha diritto di possesso su tutte le immagini della galleria, e ogni tipo di contenuto ha un'applicazione predefinita che lo gestisce, ad esempio l'applicazione di default per gestire le fotografie del dispositivo è Galleria. Le applicazioni si registrano per gestire un tipo di contenuto, in questo modo il sistema riesce a visualizzare all'utente una lista di applicazioni da scegliere per portare a termine la propria azione, come ad esempio al modifica di una fotografia. A livello implementativo, il contenuto trasferito da un'applicazione all'altra è un oggetto che mantiene in memoria l'applicazione sorgente, l'applicazione destinazione, il contenuto e un'informazione di stato. Per salvare un contenuto modificato si fa uso di ContentStore, un servizio offerto dalle API del Content Hub.

È utile far notare che ad oggi Content Hub non risulta ancora completo.

3.2.1 Scenario di un'applicazione che importa contenuti

1. L'applicazione destinazione, attraverso le API di Content Hub, fa richiesta di importo del contenuto.

2. Il Content Hub crea l'oggetto di trasferimento e lo ritorna all'applicazione richiedente, connettendola all'applicazione sorgente; il Content Hub avvia ed esegue l'applicazione sorgente in modalità trasferimento.

3. L'utente attraverso l'applicazione sorgente seleziona il contenuto da importare (ad esempio, le immagini delle galleria). L'applicazione sorgente cambia lo stato dell'oggetto trasferimento a Pronto.

[7] Content Hub Guide https://developer.ubuntu.com/en/apps/platform/guides/content-hub-guide/

4. L'applicazione destinazione rileva lo stato Pronto del trasferimento e importa il contenuto.

5. L'utente modifica, eventualmente, il contenuto importato e fa uso del ContentStore per salvarlo in maniera persistente.

3.2.2 Scenario di un'applicazione che esporta contenuti

1. L'applicazione sorgente si registra presso il Content Hub come sorgente per un determinato tipo di contenuto.

2. L'applicazione sorgente implementa la modalità trasferimento che il Content Hub invoca per effettuare lo scambio.

3. La modalità di trasferimento deve permette all'utente di selezionare il contenuto da importare.

4. L'applicazione deve cambiare lo stato dell'oggetto trasferimento indicando che il contenuto è pronto.

3.2.3 Registrare un'applicazione come sorgente o destinazione

Per registrare un'applicazione come sorgente o destinazione si fa uso degli *hooks* di Click nel file `manifest.json`. Segue un esempio:

```
{
    ...
    "hooks": {
        "APPNAME": {
            "apparmor": "APPNAME.apparmor",
            "content-hub": "APPNAME.content.json",
            "desktop": "APPNAME.desktop"
```

```
        },
        ...
    }
    ...
}
```

Gli *hooks* sono quindi una lista di dizionari che specificano le varie informazioni sulle applicazioni con cui si vuole scambiare contenuti. Se un'applicazione vuole registrarsi come sorgente per un determinato tipo di contenuto deve aggiungere le proprie informazioni alla lista.

Segue un esempio del file `APPNAME.content.json` di un'applicazione che si registra come destinazione e sorgente per i contenuti di tipo `pictures` e `video`:

```
{
    "destination": [
        "pictures",
        "videos"
    ],
    "source": [
        "pictures",
        "videos"
    ]
}
```

Ecco un esempio del file `APPNAME.apparmor` della stessa applicazione:

```
{
    "policy_groups": [
        "networking",
        "content_exchange_source",
        "content_exchange"
    ],
```

```
    "policy_version": 1
}
```

In questo esempio `content_exchange` è il *policy group* di AppArmor che permettere all'applicazione sia di ricevere che di inviare contenuti, `content_exchange_source` è invece necessaria solo nelle applicazioni sorgenti.

3.3 Applicazioni in HTML5

Le applicazioni scritte in HTML5 vengono eseguite in web container e sono completamente integrate nel sistema come un qualsiasi altro software. Il container web fornisce l'accesso ad una serie di API JavaScript[8] che permettono l'accesso al sistema. Queste includono le API della piattaforma di Ubuntu (come il Content Hub, gli Account OnLine, le notifiche, etc ...) e le API Cordova[9] che forniscono accesso al sistema e alle periferiche del dispositivo come fotocamera e accelerometri.

3.3.1 Esecuzione e debug

Le applicazioni scritte in HTML5 possono essere create attraverso Ubuntu SDK, che offre lo scheletro applicativo per un progetto base. Solitamente i file interessanti risiedono, all'interno del progetto, in cartella di nome `www`, per lanciare una di queste applicazioni è sufficiente digitare da terminale:

```
ubuntu-html5-app-launcher --www=www
```

Il container web è basato su WebKit[10], un browser engine Open Source, e offre i medesimi tool di sviluppo di quest'ultimo. Aggiungendo infatti l'opzione `--inspector` al comando precedente, il container web avvia l'Inspector

[8] HTML5 Ubuntu API https://developer.ubuntu.com/en/apps/html-5/api/
[9] Apache Cordova homepage http://cordova.apache.org/
[10] The WebKit Open Source Project https://www.webkit.org/

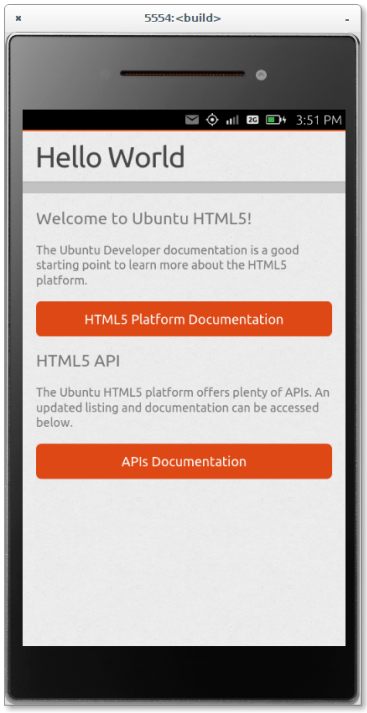


Figura 3.2: Anteprima dell'applicazione HTML5 base in Ubuntu SDK

server, il servizio predefinito di debug, e ne stampa a video l'URL. Aprendo l'URL con un browser basato su WebKit (come Chrome[11] o Chromium[12]) è possibile accedere alla modalità di debug. Ecco un esempio dell'output:

```
Inspector server started successfully.
Try pointing a WebKit browser to http://192.168.0.3:9221
```

3.3.2 Descrizione dei file principali

Creando un nuovo progetto per un'applicazione HTML5 in Ubuntu SDK, molti file vengono auto generati. Nella cartella `www` si trovano tutti i file relativi all'applicazione (CSS, JavaScript, immagini) e il file `index.hml`, che importa tutto il necessario per l'esecuzione e definisce la struttura grafica dell'applicazione.

[11] Google Chrome https://www.google.com/chrome/
[12] The Chromium browser web http://www.chromium.org/Home

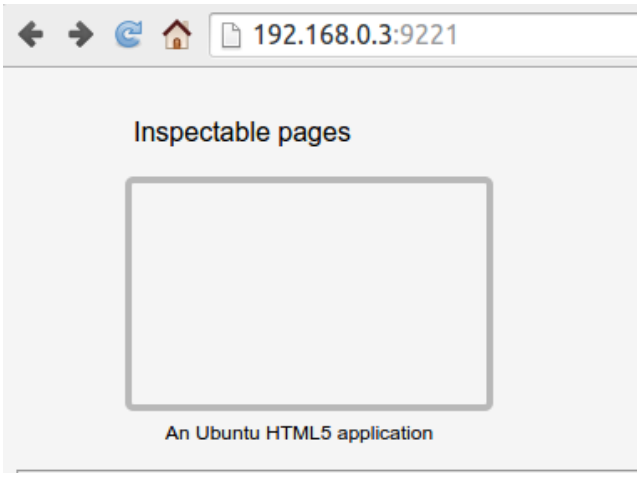


Figura 3.3: Vista della modalità di debug in Chromium

Il file `www/js/app.js` è il JavaScript base per un applicazione HTML5, nel quale va aggiunto il comportamento e il controllo del software. In esso si trova l'inizializzazione dell'interfaccia utente tramite uso delle API:

```
window.onload = function () {
    var UI = new UbuntuUI();
    UI.init();
    ...
}
```

Nell'esempio mostrato, l'interfaccia è inizializzata all'interno di una funzione anonima che fa da *event handler* per l'evento **window.onload**, un evento generato quando il Document Object Model (DOM)[13] è completamente caricato in memoria. Il framework grafico viene successivamente inizializzato tramite la chiamata a funzione **UI.init()**.

Segue la descrizione di altri file importanti per il progetto:

- **APPNAME.desktop**: Il file utilizzato dal sistema per gestire e lanciare l'applicazione. Le specifiche di questo file sono dettate dal pro-

[13]Document Object Model (DOM) http://www.w3.org/DOM/

getto ***freedesktop.org***[14]. Qui si nota il campo **Exec**, che dichiara il comando di esecuzione dell'applicazione (come riportato precedentemente), e i campi **Name** e **Icon** che specificano il nome dell'applicazione il path dell'icona (di solito locale all'applicazione) che il sistema visualizza all'utente.

- **APPNAME.ubuntuhtmlproject**: file utilizzato da Ubuntu SDK per gestire il progetto dell'applicazione.
- **APPNAME.apparmor**: il file con le specifiche per la configurazione di AppArmor, qui sono dichiarati i gruppo di accesso per i permessi.
- **manifest.json**: file con le specifiche per la compilazione dell'applicazione con Click.

3.3.3 Struttura dell'HTML5 in Ubuntu

Ubuntu HTML5 è l'insieme di tutti i markup (*tag* o etichette) che sono aggiunti all'HTML5 base per sviluppare a pieno l'interfaccia utente delle applicazioni in modo che siano completamente integrate nel sistema.

È utile subito definire i layout disponibili per le applicazioni. In Ubuntu ne troviamo due principali:

- **Un *header* con *tabitems***
  quando un utente clicca su un tabitem, la GUI ne mostra il contenuto. È di solito chiamato anche layout "Flat" navigation, perché le tab sono tutte sullo stesso livello e l'utente le naviga orizzontalmente, facendo clic sull'header.
- **Un *pagestack* di pagine**:
  l'utente naviga le pagine in maniere verticale e utilizza di solito un pulsante di "Indietro" per tornare al contenuto precedente. È anche chiamato layout "Deep" navigation.

[14]Desktop Entry Specification http://standards.freedesktop.org/desktop-entry-spec/latest/

I **widget** più importanti sono quelli classici di una normale interfaccia utente: *tab, pagestack/page, button, dialog, list, shape, popover, footers,* etc... e sono dichiarati nel file `index.html` sotto forma di elementi HTML. Nessun nuovo elemento è stato introdotto nello standard, i widget sono infatti dichiarati attraverso l'uso dell'attributo *data-role*, che ne specifica il tipo. Ecco alcuni esempi:

lista `<div data-role="list">`

dialog `<div data-role="Dialog">`

pulsante `<button data-role="button">`

Per importarli è necessario aggiungere all'header HTML le seguenti specifiche per importare il tema e il comportamento dei widget:

```
<head>
...
    <!-- Stile -->
    <link href="ROOT_PATH/css/appTemplate.css"
        rel="stylesheet" type="text/css" />
    <!-- JavaScript -->
    <script src="ROOT_PATH/js/core.js"></script>
    ...
    <script src="ROOT_PATH/js/fast-buttons.js"></script>
...
</head>
```

Dove `ROOT_PATH` è il percorso base al framework di Ubuntu: `/usr/share/ubuntu-html5-ui-toolkit/0.1/ambiance`.

È utile far notare che è possibile importare anche librerie JavaScript esterne, come ad esempio jQuery[15].

Il corpo pricipale dell'applicazione è invece definito all'interno dell'elemento `<body>` ed è così formato:

[15]jQuery project homepage http://jquery.com/

```
<body>
    <header data-role="header">
    ...
    </header>
    <div data-role="content">
    ...
    </div>
</body>
```

Sia l'*header* che il *content* sono utilizzati sia nei layout Flat che in quelli Deep.

Esempio di Flat layout

Segue un esempio di come è struttura un'applicazione con layout Flat, ovvero con navigazione orizzontale a tab.

Nell'header è definita la lista non ordinata delle tab. Ogni oggetto della lista ha un *data-role* definito come *tabitem*, e un attributo *data-page* che dichiara l'identificato unico del tab associato. Quando un utente clicca su un tabitem, la *tab* con *id* uguale a quello definito nell'attributo data-page viene visualizzata:

```
<header data-role="header">
  <ul data-role="tabs">
    <li data-role="tabitem" data-page="main">Main</li>
    <li data-role="tabitem" data-page="anotherpage">Another</li>
  </ul>
</header>
<div data-role="content">
  <div data-role="tab" id="main">
  ...
  </div>
  <div data-role="tab" id="anotherpage">
```

```
  ...
  </div>
</div>
```

Esempio di Deep layout

Segue un esempio di come è struttura un'applicazione con layout Deep, ovvero con navigazione attraverso pagine.

Molte applicazioni sono naturalmente compatibili con questo tipo di struttura, come un lettore di feed RSS ad esempio che potrebbe essere così suddiviso:

- la pagina principale mostra lista di tutti i feed
- quando un utente seleziona un singolo feed, l'applicazione mostra un nuova pagina una lista di articoli
- l'utente seleziona un articolo e una nuova pagina, con il contenuto di quest'ultimo, viene visualizzata

È quindi chiara la gerarchia o *stack* delle pagine: ***Lista dei Feed → Lista di Articoli → Articolo***.

Nel framework di Ubuntu HTML5, un oggetto *pagestack* è usato per tener traccia di questa gerarchia, di quali pagine esistono e quale è attualmente visibile, ovvero che si trova in alto allo stack. Le pagine sono dichiarate all'interno del pagestack markup:

```
<body>
  <div data-role="mainview">
    <header data-role="header">
    ...
    </header>
    <div data-role="content">
      <div data-role="pagestack">
        <div data-role="page" id="main">
```

```
        ...
        </div> <!-- page: main -->
        <div data-role="page" id="anotherPage">
        ...
        </div> <!-- page: anotherPage -->
      </div> <!-- pagestack -->
    </div> <!-- content -->
  </div> <!-- mainview -->
</body>
```

Se necessario un **Footer** viene aggiunto automaticamente dal Framework e permette all'utente, attraverso un pulsante "Indietro", di risalire la gerarchia. In caso di necessità, è possibile anche sovrascrivere il footer predefinito specificandone uno personalizzato per ogni pagina.

3.4 Applicazioni native in QML

Le applicazioni native di Ubuntu sono scritte in QML.

QML (Qt Modeling Language) è un meta linguaggio moderno, progettato per essere semplice ed estendibile, con una sintassi e paradigmi simili a quelle dalle tecnologie web oggi onnipresenti, come CSS e JavaScript. È sviluppato sul framework grafico Qt, e si integra al punto da diventarne parte fondamentale. In breve, QML insieme alle librerie Qt costituiscono una tecnologia di interfaccia utente di alto livello, un'unione che permette una veloce creazione di applicazioni touch, animate e leggere.

QML si differenzia dai tradizionali linguaggi procedurali, come Python e C, perché completamente dichiarativo. Questa proprietà permette al linguaggio sia di modellare che di descrivere gli elementi della UI e le loro interazioni.

QML è anche un linguaggio di scripting, che non richiede la compilazione o cross-compilazione e può immediatamente essere eseguito su qualsiasi dispositivo. Ciò fornisce un flusso di lavoro di sviluppo e test molto rapido,

```
// Simple QML example
import QtQuick 2.3
Rectangle {
    width: 200 // this is a property
    height: 200
    color: "blue"
    Image {
        source: "pics/logo.png"
        anchors.centerIn: parent
    }
}
```

Figura 3.4: Esempio di codice QML

e la creazione di prototipi per le interfacce utente diventa un processo molto semplice e agile. Nonostante sia un linguaggio di scripting, QML può sfruttare il supporto OpenGL per fornire alte prestazioni, quasi simili a quelli dei framework per giochi.

Ubuntu offre una collezione di API completa per l'accesso al dispositivo, per maggiori informazioni si rimanda alla documentazione ufficiale[16].

3.4.1 Oggetti e tipi base

I componenti e i loro elementi sono i blocchi principali che costituiscono un applicazione QML. Gli elementi dell'interfaccia utente base, come rettangoli e cerchi, possono essere usati per creare l'esperienza utente di un'applicazione, ma diventano davvero utili quando sono raggruppati insieme e incapsulati dentro componenti e moduli per offrire e costruire nuove funzionalità.

Gli elementi QML o i componenti hanno di solito diverse proprietà che aiutano a definirli e vengono dichiarati con una sintassi simile al CSS. Ad esempio, un elemento rettangolo ha una proprietà di *width* e una di *height*

[16]QML Ubuntu API https://developer.ubuntu.com/en/apps/qml/api/

che ne definiscono la grandezza. Ubuntu SDK sfrutta a pieno il vantaggo dell'estendibilità del linguaggio QML per offire un ventaglio completo di componenti per l'interfaccia utente, componenti che aggiungono innumerevoli funzionalità e che si integrano pienamente nel sistema.

I **tipi base** che QML offre sono: *bool*, *double*, *enumeration*, *int*, *list*, *real*, *string*, *url*, *var* (proprietà generica). A questi si aggiungono i **JavaScript** Objects che QML supporta. In questo esempio si mostra come sia *Date* che *Array*, oggetti JavaScript, siano supportati dal linguaggio:

```
import QtQuick 2.2
Item {
    property var theArray: new Array()
    property var theDate: new Date()
    Component.onCompleted: {
        for (var i = 0; i < 10; i++)
            theArray.push("Item " + i)
        console.log("Ci sono", theArray.length, "oggetti nell'array")
        console.log("L'ora e'", theDate.toUTCString())
    }
}
```

Gli ulteriori Object del linguaggio sono invece tutti derivati da QtObject[17], un elemento non-visuale che contiene un unica proprietà, l'*objectName*.

Un oggetto che deriva da QtObject è quindi strutturato come segue:

```
import QtQuick 2.2
Item {
    QtObject {
        id: attributes
        property string name
        property int size
```

[17]QObject Class doc http://doc.qt.io/qt-5/qobject.html

```
        property variant attributes
    }
    Text { text: attributes.name }
}
```

La maggior parte dei QtObject predefiniti sono disponibili nel modulo QtQuick che offre anche una solida base di tipi, come ad esempio: *color, date, font, point, rect, size* e altri ancora.

3.4.2 Il sistema di eventi

QML ha un meccanismo di **Signal** e **Handler** per l'interazione fra componenti, dove il *signal* è definito come l'evento che innesca un determinato *signal handler*. Per rispondere ad un evento è quindi necessario aggiungere del codice direttamente nell'handler relativo.

Per ricevere la notifica di quando un particolare segnale esterno è emesso, è necessario che nella definizione dell'oggetto venga dichiarato un signal handler nella forma `on<Signal>`, dove `<Signal>` è il nome del segnale di interesse con la prima lettera in maiuscolo.

Ad esempio, il tipo **MouseArea** del **QtQuick** ha un segnale chiamato `clicked` che viene emesso ogni qualvolta che con il mouse si fa clic sull'area. Visto che il nome del segnale è `clicked`, il suo handler è chiamato `onClicked`.

Controllando la documentazione[18] si nota anche che l'evento `clicked` è emesso con un parametro `mouse`, che è un oggetto di tipo **MouseEvent** e contiene informazioni riguardo il mouse. Nell'handler è quindi noto questo oggetto, proprio con il nome di `mouse`, anche se non è esplicitamente dichiarato. Ecco un esempio dell'evento `clicked` su una **MouseArea**, si noti che il signal handler contiene codice JavaScript.:

```
Rectangle {
    id: rect
```

[18]MouseArea QML doc http://doc.qt.io/qt-5/qml-qtquick-mousearea.html

```
    width: 100; height: 100
    MouseArea {
        anchors.fill: parent
        onClicked: {
            // accesso a 'mouse'
            console.log("Coordinate ", mouse.x, mouse.y)
        }
    }
}
```

Segnali emessi quando una proprietà cambia

Un segnale è emesso anche quando il valore di una proprietà cambia. Questo tipo di segnale è definito come *property change signal*, e il suo signal handler è dichiarato nella forma `on<Property>Changed` dove `<Property>` è il nome della proprietà con la prima lettera in maiuscolo.

Riprendendo l'esempio di prima, la **MouseArea** ha una proprietà chiamata `pressed`. Per ricevere una notifica di quando questa proprietà cambia è necessario creare un signal handler con nome `onPressedChanged`. Segue un esempio:

```
Rectangle {
    id: rect
    width: 100; height: 100
    MouseArea {
        anchors.fill: parent
        onPressedChanged: {
            console.log("Valore di pressed:", pressed)
        }
    }
}
```

Si noti anche che questo tipo di segnali di solito non sono esplicitamente descritti nella documentazione, ma fanno parte intrinsecamente del linguaggio. Il signal handler `onPressedChanged` è infatti valido semplicemente perché l'oggetto dichiara la proprietà di nome `pressed`.

Utilizzo dell'oggetto Connections

In alcuni casi potrebbe essere utile accedere a un segnale al di fuori dell'oggetto che lo emette. Per questo tipo di casi d'uso, il modulo QtQuick mette a disposizione l'oggetto **Connections**[19] che si connette ai segnali di altri oggetti definiti nella proprietà `target`. Segue del codice di esempio:

```
Rectangle {
    id: myRect
    width: 100; height: 100
    MouseArea {
        id: myMouseArea
        anchors.fill: parent
    }
    Connections {
        target: myMouseArea
        onClicked: {
            myRect.width = Math.random();
        }
    }
}
```

Aggiungere Signal personalizzati

Dei nuovi segnali possono essere aggiunti agli oggetti QML attraverso la *keyword* `signal`.

La sintassi per la definizione di un nuovo segnale è la seguente:

[19] Connections QML doc http://doc.qt.io/qt-5/qml-qtqml-connections.html

```
signal <name>[([<type> <parameter name>[, ...]])]
```

Il segnale è emesso invocando il segnale stesso come metodo.

Nell'esempio seguente si definisce un **SquareButton** e si aggiunge un nuovo signal a **Rectangle** invocandolo dalla **MouseArea**:

```
// SquareButton.qml
Rectangle {
    id: myRect
    signal activated(real xPosition, real yPosition)
    width: 100; height: 100
    MouseArea {
        anchors.fill: parent
        onPressed: myRect.activated(mouse.x, mouse.y)
    }
}
```

A questo punto è possibile scrivere la propria applicazione e fare uso del nuovo segnale, come riportato di seguito:

```
// myapplication.qml
SquareButton {
    onActivated: console.log("Attivato " + xPosition + "," + yPosition)
}
```

3.5 Gli Scope

In Ubuntu è possibile cercare contenuti e servizi da numerose fonti contemporaneamente senza la necessità di installare applicazioni dedicate. Questo nuovo tipo di servizio, integrato pienamente nel dispositivo e in Unity, prende il nome di Scope[20].

[20]Introduzione agli Scope http://www.ubuntu.com/phone/scopes

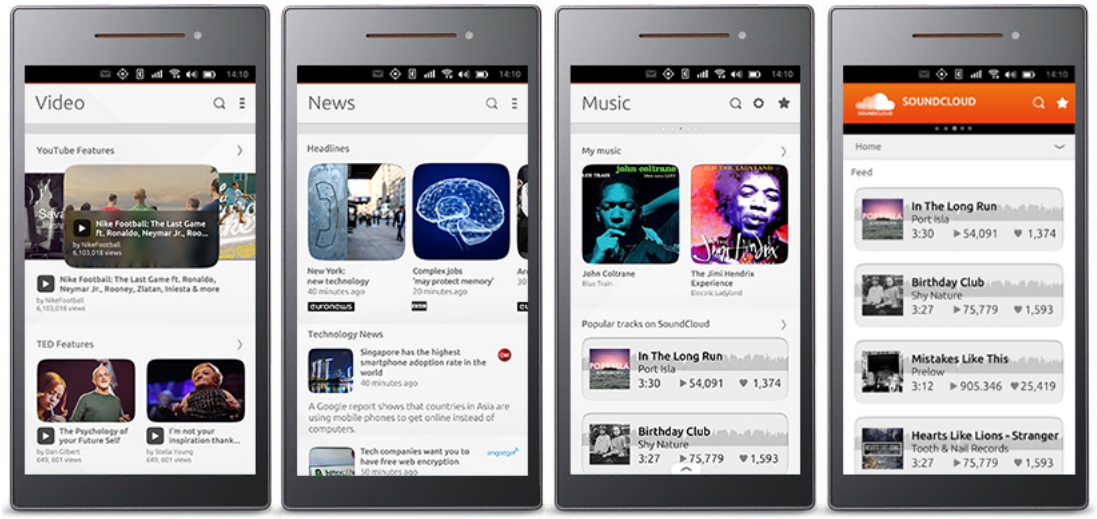


Figura 3.5: Esempi di Scope in Ubuntu Touch

Gli Scope sono quindi una completa reinvenzione dell'esperienza utente. Se da una parte gli utenti potranno accedere a contenuti in un modo tutto nuovo, senza il bisogno di scaricare e installare singole applicazioni specifiche, dall'altra gli Scope semplificano il lavoro agli sviluppatori, offrendo loro la possibilità di far usufruire dei propri servizi senza la necessità di scrivere o mantenere un'applicazione dedicata.

Esistono sono due tipi di Scope, di aggregazione e di brand.

Aggregation Scope Sono gli Scope predefiniti nel sistema, quelli che, in base ad una pattern di ricerca, offrono contenuti misti e da fonti multiple. Ad esempio, lo Scope Video cerca file video da diverse sorgenti (come Youtube), così come lo Scope Musica ricerca file musicali provenienti da diversi provider.

Branded Scope Sono gli Scope che più assomigliano alle classiche applicazioni. Attraverso l'uso delle API di Ubuntu è possibile creare velocemente un applicazione dedicata che, in maniera semplice e sicura, si connette al proprio servizio e ne estrae i contenuti di interesse da mostrare sul dispositi-

vo. Esempi di Branded Scope sono gli Scope di Soundcloud o di Instagram, che offrono contenuti specifici per un determinato servizio o brand.

3.5.1 L'architettura degli Scope

Una delle caratteristiche più innovative di Unity è la **Dash** che permette agli utenti di cercare velocemente informazioni sia locali al dispositivo (applicazioni installate, file recenti, file musicali, etc...) sia dislocate su Internet (Twitter, Google Drive, Facebook, etc...) attraverso l'utilizzo di Scope.

Gli Scope possono essere sia locali, sono eseguiti sul proprio dispositivo, che remoti, sono eseguiti su un Scope Server. Questi ultimi, quelli remoti, non sono ancora pienamente supportati, ma saranno aggiunti nei prossimi rilasci di Ubuntu.

La parte centrale del ciclo di vita degli Scope è lo **Scope Registry**, un processo che mantiene una lista di tutti gli Scope (locali o remoti) e che gli esegue in maniera su richiesta e solo quando richiesti, cosicché solo gli Scope necessari per una determinata ricerca sono eseguiti durante una query.

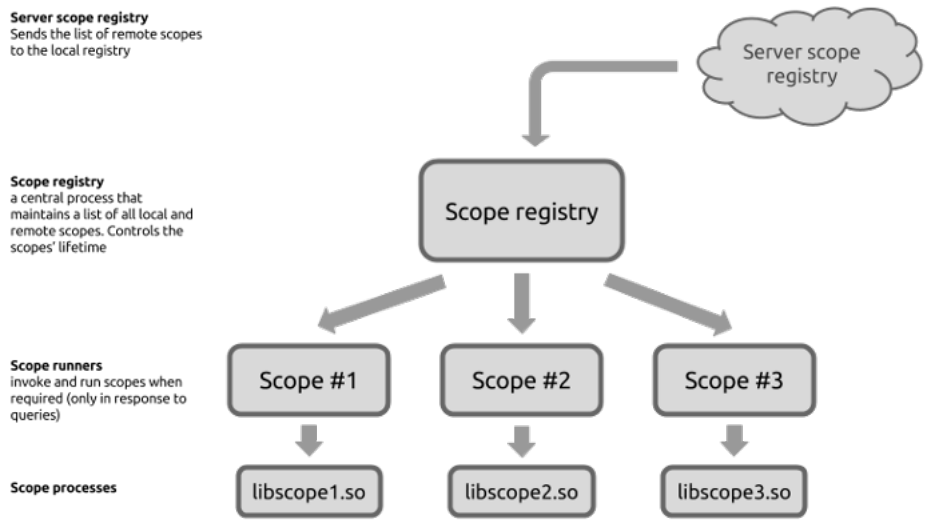


Figura 3.6: Architettura dello Scope Registry

3.5.2 Comportamento e data flow

Uno Scope è sostanzialmente una Query che ritorna risultati (contenuti) all'utente che l'invocata. Ad esempio, un utente può cercare un Video attraverso lo Scope dedicato, lo Scope effettua la query e ritorna i risultati alla Dash che li mostra all'utente.

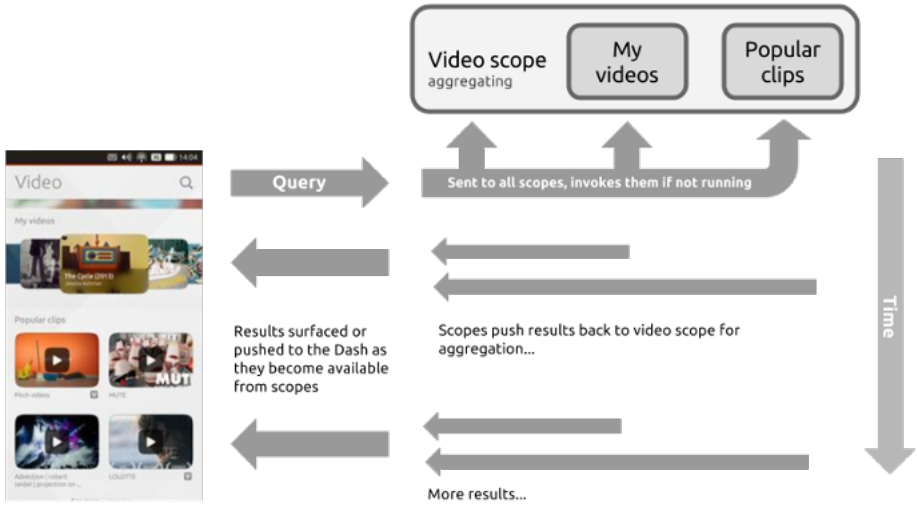


Figura 3.7: Esempio di data flow dello Scope Video

È utile far notare che uno Scope può aggregare dati da diverse sorgenti, incluso altri Scope, in altre parole uno Scope può chiamare altri Scope (*aggregation Scope*).

3.5.3 Separazione fra dati e visualizzazione

Il risultato di uno Scope viene rappresentato visivamente in maniera controllata dalla Dash. Detto questo, un utente che sviluppa uno Scope non si preoccupa minimamente di come visualizzare il contenuto del risultato, ma è il framework che se ne occupa.

Ci sono tuttavia dei margini di personalizzazione, attraverso delle opzioni che lo sviluppatore può configurare in semplice oggetto JSON, ma di fatto, non vi è accesso diretto alle librerie grafiche.

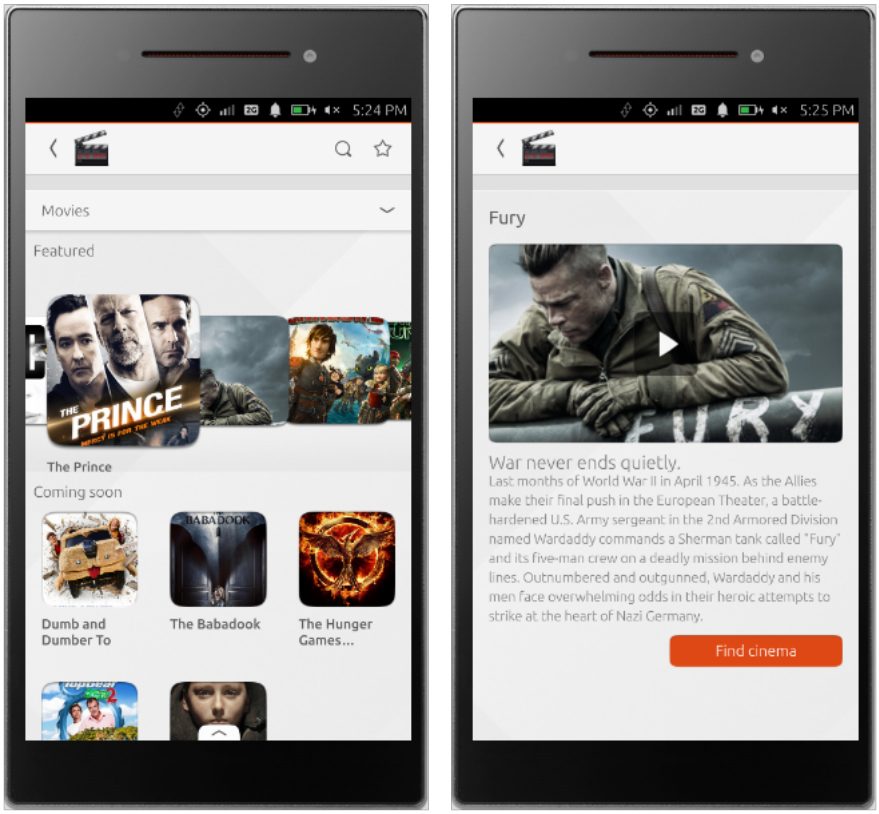


Figura 3.8: Le due modalità visuali di uno Scope

Lo stesso approccio di separazione è usato nell'anteprima del contenuto. Quando l'utente selezione dalla lista dei risultati l'elemento di interesse, l'interfaccia utente cambia e entra in modalità anteprima. In questa modalità, l'elemento selezionato è in primo piano e ne vengono visualizzati i dettagli. Anche qui è possibile configurare leggermente l'interfaccia attraverso parametri di configurazione, ma non è possibile accedere al toolkit grafico.

In pratica, la separazione fra dati e visualizzazione semplifica enormemente lo sviluppo di uno Scope, che si riduce - molte volte - alla realizzazione di una sola query.

Capitolo 4

Aggiornamenti di sistema

Con il porting di Ubuntu per dispositivi mobili è nata l'esigenza di introdurre un nuovo paradigma per la gestione degli aggiornamenti di Ubuntu for Phone.

Dopo le valutazioni iniziali e l'analisi dei requisiti strettamente connessi alla poca potenza computazionale degli smartphone oggi in commercio, gli ingegneri di Canonical hanno scelto di introdurre un nuovo workflow per aggiornare il core dei dispositivi: lato server vengono generate le immagini aggiornate del sistema e, successivamente, pubblicate delle *delta* che includono solo i cambiamenti introdotti da questi aggiornamenti.

I client quindi scaricano e importano l'aggiornamento delta, che risulterà più veloce e performante da applicare rispetto ad un più classico aggiornamento di sistema con gli strumenti standard dello scenario GNU/Linux, i così detti *package manager* come *apt* (Debian) o *rpm* (Red Hat), che spostano lato client le operazioni di controllo nelle fasi di pre e post installazione[1].

4.1 Caratteristiche principali

Gli obbiettivi principali di questo nuovo sistema di aggiornamenti sono:

- avere un sistema base comune su quanti più dispositivi possibile

[1] Ubuntu System Image https://launchpad.net/ubuntu-system-image

- garantire che con l'applicazione dell'aggiornamento parziale tutti i dispositivi continueranno ad avere un sistema base identico
- spostare le operazioni di controllo sul server riducendo l'utilizzo delle risorse dei dispositivi mobili
- avere una perfetta suddivisione fra sistema, applicazioni e dati utenti, permettendo semplici formattazioni di sistema senza la perdita di dati personali o velici pulizie dei dati personali riportando il dispositivo alle impostazioni iniziali (*factory reset*)

E i componenti principali sviluppati sono:

- il **server** mantiene traccia degli aggiornamenti e gli indicizza in modo che ogni client possa trovare il pacchetto giusto per il proprio dispositivo
- il **client** scarica gli indici dal server, li verifica e trova il percorso migliore per scaricare tutti gli aggiornamenti disponibili, infine scarica i file necessari e li passa al upgrader
- l'**upgrader** riceve una lista di aggiornamenti dal client, li verifica e gli applica al sistema
- il **downloader** un servizio di sistema usato per gestire il download (con funzioni di pausa/continua/cancella) degli aggiornamenti in maniera efficiente

I principali utilizzatori di questo servizio oggi sono solo i dispositivi con installato Ubuntu Touch (smartphone e tablet), tuttavia il sistema è facilmente modificabile in modo che anche i computer standard, con architetture x86 ad esempio, siano in grado di utilizzarlo.

Versioning Gli aggiornamenti sono rilasciati di solito con un numero di versione, un intero in modo che siano facilmente ordinabili.

Frequenza di rilascio Non esiste un requisito che limiti il numero di rilasci. Infatti, a seconda del canale di interesse, è possibile ricevere aggiornamenti quotidiani (versione in fase di sviluppo) o settimanali/mensili (versione stabile). Se necessario, i rilasci sono pubblicati sia come aggiornamenti parziali che come completi.

Per ogni aggiornamento completo è possibile indicare una versione minima supportata per applicare i cambiamenti, il che indica al client di procedere preventivamente all'aggiornamento del dispositivo ad una versione più recente come requisito di installazione.

4.2 I canali

Le immagini dei dispositivi Ubuntu sono rilasciati attraverso diversi canali.

Il canale di default del dispositivo viene scelto durante la fase di installazione e, di default, l'immagine del rilascio più recente presente in quel canale viene installata.

4.2.1 La promozione delle immagini

Il numero di canali che si può abilitare su un singolo dispositivo è variabile e dipende da diversi fattori. Ad esempio, per i dispositivi che vogliono testare la versione in fase di sviluppo di Ubuntu è suggerito abilitare una coppia di canali **devel** e **devel-proposed**.

Le immagini in fase di sviluppo sono generate quotidianamente (spesso anche più volte al giorno), e sono distribuite principalmente sul canale **devel-proposed**. Quando soddisfano una serie di criteri di qualità vengono promosse sul canale principale, il canale **devel**[2].

[2]Ubuntu image channels https://developer.ubuntu.com/en/start/ubuntu-for-devices/image-channels/

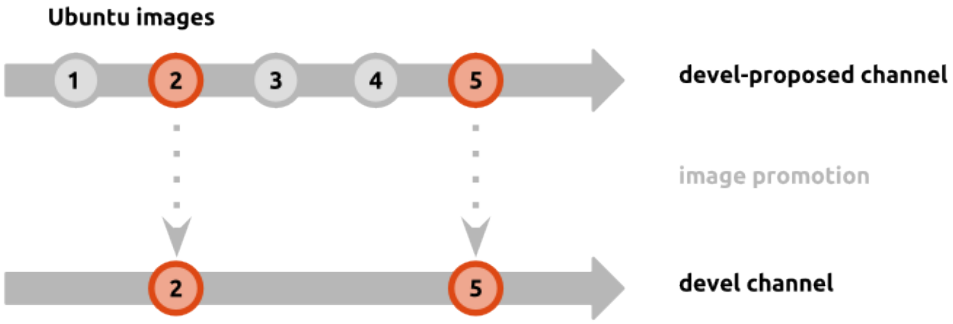


Figura 4.1: Promozione delle immagini attraverso i canali

Questo workflow viene applicato a tutte le coppie di canali proposed/non-proposed, ad esempio `<release-name>` + `<release-name>-proposed` per una determinata versione di Ubuntu.

4.2.2 Lo schema dei nomi

I canali ufficiali seguono un determinato schema per i nomi e hanno tutti prefisso `ubuntu-touch/`, ad esempio per il canale devel:

```
ubuntu-touch/devel | <codename> [-proposed]
```

`devel` in questo caso è un alias alla versione attualmente in fase di sviluppo; `codename` indica il nome del rilascio di Ubuntu; `proposed` (opzionale) indica se il canale contiene tutte le immagini disponibili o solo quelle promosse.

4.3 L'implementazione del server

Il server degli aggiornamenti è l'host dove tutti i client si connettono per scaricare la lista degli aggiornamenti disponibili. Gli aggiornamenti sono divisi in *channel* e *device*. Il client chiede la lista di tutti i canali e dopo la filtra in base al modello del proprio dispositivo.

Il server è stato disegnato per essere il più semplice possibile, tutti i file sono generati staticamente per garantire facile scalabilità del servizio attraverso *mirroring* se necessario.

Le immagini ufficiali di Ubuntu sono disponibili all'indirizzo `https://system-image.ubuntu.com`.

4.3.1 Il file channels.json

La lista dei canali è registrata all'interno di un file chiamato `/channels.json` che ad ogni canale associa una lista di dispositivi e ad ogni dispositivo il relativo file indice.

Struttura

Il file channel.json è così strutturato:

- Nome del canale (stringa)
  - `alias` - eventuale alias per il nome del canale (stringa, opzionale)
  - `devices` - lista dei dispositivi supportati dal canale (dizionario, obbligatorio)
    - `index` - percorso del file di index per il dispositivo, solitamente nella forma `/<channel>/<some-path>/<some-file>.tar.xz` (stringa, obbligatorio)
    - `keyring` - dizionario del keyring del dispositivo (dict, opzionale)
      - `path` - percorso al keyring del dispositivo, solitamente nella forma `/<channel>/<device>/device-signing.tar.xz` (stringa, obbligatorio)
      - `signature` - percorso alla firma digitale del keyring, solitamente nella forma `/<channel>/<device>/device-signing.tar.xz.asc` (stringa, obbligatorio)

Esempio di file channel.json

```
{
  "devel": {
    "alias": "saucy",
    "devices": {
      "grouper": {
        "index": "/devel/grouper/index.json"
      },
      "maguro": {
        "index": "/devel/maguro/index.json"
      },
      "mako": {
        "index": "/devel/mako/index.json"
      },
      "manta": {
        "index": "/devel/manta/index.json",
        "keyring": {
          "path": "/.../device-signing.tar.xz",
          "signature": "/.../device-signing.tar.xz.asc"
        }
      }
    }
  },
  ...
}
```

4.3.2 Il file index.json

La lista degli aggiornamenti disponibili per un determinato dispositivo è contenuta nel file `/<channel>/<model>/index.json`.

Questo file viene utilizzato dal client per capire il percorso degli aggiornamenti da intraprendere.

Struttura

- `global` - dizionario di variabili globali
  - `generated_at` - data di creazione del file (stringa, obbligatoria)
- `images` - lista di dizionari che rappresentano le immagini disponibili
  - `base` - versione base per l'immagine delta (16bit intero senza segno, obbligatorio solo per immagini delta)
  - `bootme` - flag che indica al downloader se è necessario riavviare il dispositivo per applicare l'aggiornamento (booleano, opzionale, di default falso)
  - `description` - descrizione dell'immagine (stringa, obbligatorio)
  - `description-XX_YY` - descrizione tradotta nella lingua `XX_YY` (stringa, opzionale)
  - `files` - lista di dizionari che rappresentano i file necessari che sono parte di un aggiornamento (lista, obbligatorio)
    - `checksum` - impronta SHA256 del file (stringa, obbligatoria)
    - `order` - l'ordine del file nella lista (intero, ordine ascendente, obbligatorio)
    - `path` - il percorso o l'URL al file (stringa, obbligatorio)
    - `signature` - il percorso o l'URL alla firma digitale del file (stringa, obbligatorio)
    - `size` - dimensione del file (intero, obbligatorio)
  - `minversion` - versione minima del sistema necessario all'aggiornamento (intero, opzionale, solo per le immagini complete, di default 0)

- `phased-percentage` - la percentuale di utenti che dovrebbero ricevere l'aggiornamento (intero fra 0 e 100, opzionale, di default 100)
- `type` - tipo dell'immagine (stringa, obbligatorio, uno fra `full` o `delta`)
- `version` - numero di versione dell'immagine (16 bit intero senza segno, obbligatorio)
- `version_detail` - maggiori informazioni sulla versione (stringa, opzionale, arbitraria)

Un aggiornamento può contenere un qualsiasi numero di file che verranno applicati in maniera sequenziali in un determinato ordine, tuttavia, per le immagini di Ubuntu, solitamente se ne usano solo tre:

1. `hardware-dependent`: contiene bit specifici per un determinato tipo di hardware
2. `hardware-independent`: contiene bit comuni fra dispositivi eterogenei
3. `version` - contiene un singolo file `/etc/ubuntu-build` che indica il numero di versione dell'immagine

È utile far notare che un'immagine conterrà quindi almeno due file, uno dipendente o indipendente dall'hardware (a seconda del tipo di aggiornamento) e un file di versione.

Utilizzando questo paradigma di divisione dei bit è possibile risparmiare enormi risorse sul server in termini di spazio allocato, dispositivi diversi utilizzeranno lo stesso file hardware-indipendent (tendenzialmente sempre più grande dell'hardware-dependent, perché destinato ad includere immagini, file di testo, manuali, etc..) durante l'aggiornamento e successivamente scaricheranno il più piccolo file relativo al proprio hardware.

4.3.3 Sicurezza

Il server è stato progettato per mantenere alto il livello di sicurezza; per assicurare che non vi siano intercettazioni o intromissioni nelle connessioni fra client e server è possibile:

- servire i file channels.json e index.json attraverso il protocollo HTTPs, in modo da evitare attacchi di tipo man-in-middle
- firmare digitalmente i file channels.json e index.json con GPG[3] e obbligare il client a verificarne l'autenticità
- verificare l'integrità dei file di aggiornamento controllando l'impronta SHA256 presente nei file di index
- verificare ulteriormente l'autenticità dei file di aggiornamento utilizzando ancora la firma GPG (staccata e in chiaro nel file `<filename>.asc`)

Per offrire questo servizio è quindi utile aggiungere ai requisiti del server il possesso di una chiave privata GPG e di un certificato valido SSL per le connessioni di tipo HTTPs.

4.4 L'implementazione del client

L'applicazione client ha il compito di calcolare l'aggiornamento, scaricare i file necessari, validarli e passarli all'upgrader impostando le opzioni necessarie per l'installazione.

L'implementazione attuale è un tool da riga di comando, con futura implementazione di un servizio DBus[4] in modo da esporre le funzioni del client alle interfacce grafiche.

[3] GNU Privacy Guard https://www.gnupg.org/

[4] DBus a message bus system http://www.freedesktop.org/wiki/Software/dbus/

4.4.1 Caratteristiche principali

Le caratteristiche principali del client sono:

- download sicuro dei file indice e dei file, attraverso l'uso del protocollo HTTPs e della verifica della firma digitale via GPG
- download e validazione dei file di aggiornamento
- utilizzo avanzato dello strumento GPG (per ulteriori informazioni si rimanda alle specifiche[5])
- risoluzione del miglior percorso di aggiornamenti da effettuare, in base alle differente politiche disponibili (grandezza totale dei file da scaricare, numero totale di riavvii da effettuare, ...)
- supporto alla gestione di sospensione e ripresa del download, con futura implementazione di un limitatore di banda

4.4.2 Esempio di aggiornamento

Durante la fase di aggiornamento, il client si comporta più o meno come segue:

1. scarica `https://server/channels.json` e cerca l'indice del canale a cui è interessato. Se presente, scarica anche il keyring GPG del proprio dispositivo
2. scarica il file indice `https://server/<channel>/<model>/index.json`
3. legge le informazioni sulla versione attualmente installata sul dispositivo dal file `/etc/ubuntu-build`
4. cerca l'aggiornamento più recente
5. calcola il percorso dell'aggiornamento, minimizzando la grandezza del download e il numero di riavvii da effettuare

[5] https://wiki.ubuntu.com/ImageBasedUpgrades/GPG

6. scarica tutti i file necessari all'aggiornamento fino al prossimo riavvio
7. convalida tutti i file scaricati
8. sposta i file nella partizione cache
9. scrive la lista degli aggiornamenti da fare, utile all'upgrader
10. riavvia il dispositivo in modalità upgrade/recovery

Per ulteriori scenari e gestioni degli errori si rimanda alla lettura della pagina Wiki relativa[6].

4.5 L'implementazione dell'upgrader

L'upgrader è un'applicazione che viene eseguita durante la modalità di *recovery*.

Se esistono degli aggiornamenti pendenti, l'upgrader al suo avvio li convalida attraverso il controllo della firma digitale e li scompatta in modo sequenziale. Successivamente applica gli aggiornamenti al dispositivo e rimuove l'archivio. Se occorrono errori durante la fase di aggiornamento, il processo viene prontamente interrotto e il dispositivo è riavviato in modalità normale.

Se invece l'upgrader viene lanciato, ma non ci sono aggiornamenti disponibili, viene visualizzato il menu della modalità recovery che permette all'utente di effettuare un reset del dispositivo alle impostazioni di factory o di avviare un aggiornamento dalla memoria SD (il file deve essere firmato per la convalida).

4.5.1 L'interazione con il client

La richiesta di aggiornamento viene effettuata dal client che posiziona i file necessari nella cartella `/android/cache/recovery/` e scrive una lista di comandi all'interno del file `/android/cache/recovery/ubuntu_command`.

[6]Wiki: Images Based Upgrades / Client / Mock scenarios https://wiki.ubuntu.com/ImageBasedUpgrades/Client#Mock_scenarios

Si mostra dunque un esempio di comandi che il client passa all'upgrader per eseguire un aggiornamento completo del sistema:

```
load_keyring image-master.tar.xz image-master.tar.xz.asc
load_keyring image-signing.tar.xz image-signing.tar.xz.asc
load_keyring device-signing.tar.xz device-signing.tar.xz.asc
format system
mount system
update ubuntu-20150100.full.tar.xz ubuntu-20150100.full.tar.xz.asc
update nexus4-20150100.full.tar.xz nexus4-20150100.full.tar.xz.asc
update version-20150100.tar.xz version-20150100.tar.xz.asc
unmount system
```

Ecco invece un esempio di aggiornamento parziale:

```
load_keyring image-master.tar.xz image-master.tar.xz.asc
load_keyring image-signing.tar.xz image-signing.tar.xz.asc
load_keyring device-signing.tar.xz device-signing.tar.xz.asc
mount system
update ubuntu-20150101.delta-20150100.tar.xz ubuntu-20150101.delta-20150100.tar.x
update nexus4-20150101.delta-20150100.tar.xz nexus4-20150101.delta-20150100.tar.x
update version-20150101.tar.xz version-20150101.tar.xz.asc
unmount system
```

Un esempio di factory reset:

```
format data
```

Come già detto in precedenza, il file dei comandi è analizzato sequenzialmente e ad ogni errore o fallimento interrompe il processo.

Capitolo 5

Sviluppare in Ubuntu

In questo capitolo mostreremo quanto sia semplice sviluppare nuove applicazioni in Ubuntu for Phone, grazie all'utilizzo di API di sistema, messe a disposizione dello sviluppatore, e a Ubuntu SDK, l'ambiente di sviluppo e IDE predefinito.

Il codice sviluppato è interamente disponibile del dispositivo di archiviazione allegato a questo documento e rilasciato sotto licenza libera GPL 3+[1].

5.1 Installazione del sistema

Vengono di seguito riportate le istruzioni per installare[2] Ubuntu Touch su smartphone e tablet.

I dispositivi supportati ufficialmente sono il Nexus 4, Nexus 7 (versione 2013 WiFi) e il Nexus 10[3]. La comunità ha comunque contribuito al

[1]General Public License ver 3 http://www.gnu.org/copyleft/gpl.html

[2]Installare Ubuntu Touch https://developer.ubuntu.com/en/start/ubuntu-for-devices/installing-ubuntu-for-devices/

[3]Ubuntu for devices https://developer.ubuntu.com/en/start/ubuntu-for-devices/devices/

porting del sistema su altri dispositivi e una lista di device non supportati ufficialmente si può trovare sulla pagina Wiki dedicata[4].

Attenzione, installare Ubuntu Touch sul dispositivo comporta una completa perdita dei dati. In caso si voglia ripristinare un backup del dispositivo in un secondo momento è possibile seguire la guida relativa[5], ma non ne è garantito il successo.

Ubuntu per i dispositivi mobili è **ancora in fase di sviluppo**. Installare Ubuntu Touch potrebbe rendere il proprio dispositivo non usabile. Importanti feature potrebbero mancare o non funzionare correttamente. Nuove versioni del prodotto potrebbero introdurre nuovi bug. Ubuntu Touch, in altre parole, non è ancora pronto per sostituire il proprio sistema sul dispositivo. Per una panoramica sullo stato attuale dell'implementazione si rimanda alle note di rilascio di Ubuntu 14.04[6].

5.1.1 Preparazione del Desktop

Per installare Ubuntu Touch sul proprio dispositivo è necessario avere una workstation Ubuntu sulla quale installare le applicazioni necessarie per l'interazione con il device. Per sapere come installare Ubuntu sul proprio PC, consultare la relativa guida[7].

Assicurarsi di aver abilitato il repository *universe* sulla propria installazione, per maggiori informazioni leggere la guida relativa[8].

1. Aggiungere il PPA (Pesonal Package Archive)[9] di Ubuntu SDK, in un terminale digitare il comando:

[4]Ubuntu Touch Devices https://wiki.ubuntu.com/Touch/Devices

[5]Restore Android after Ubuntu Touch Installation https://developer.ubuntu.com/en/start/ubuntu-for-devices/reinstalling-android/

[6]Ubuntu 14.04 (Trusty Tahr) Release Notes https://wiki.ubuntu.com/TrustyTahr/ReleaseNotes#Ubuntu_Touch

[7]Installare Ubuntu Desktop http://www.ubuntu.com/download/desktop/install-ubuntu-desktop

[8]Aggiungere repository Universe e Multiverse in Ubuntu https://help.ubuntu.com/community/Repositories/CommandLine#Adding_the_Universe_and_Multiverse_Repositori

[9]Personal Package Archive https://help.launchpad.net/Packaging/PPA

```
sudo add-apt-repository ppa:ubuntu-sdk-team/ppa
```

2. Aggiornare la lista dei pacchetti disponibili, eseguire il comando:

```
sudo apt-get update
```

3. Installare il pacchetto **ubuntu-device-flash**, il tool principale per installare Ubuntu sui dispositivi mobili:

```
sudo apt-get install ubuntu-device-flash
```

Per gestire il dispositivo è possibile installare anche il pacchetto **phablet-tools**, che comprende una serie di tool utili per l'amministrazione di device connessi via USB.

I tool di Android - adb e fastboot

Durante l'installazione di ubuntu-device-flash vengono automaticamente aggiunti al sistema altri due tool importanti del mondo Android:

- adb, per connettersi al dispositivo quando è in modalità standard
- fastboot, per connettersi al dispositivo quando è in modalità "bootloader"

Per ulteriori informazioni si rimanda direttamente all'help delle applicazioni:

```
adb help 2>&1 | less
fastboot help 2>&1 | less
```

5.1.2 Backup del dispositivo

Prima di installare Ubuntu Touch è consigliato effettuare un backup del dispositivo.

Backup dei dati e delle applicazioni

1. Abilitare "**Opzioni sviluppatore**", in ***Impostazioni → Informazioni sul dispositivo***, cliccare sette volte su **Versione build**

2. Abilitare "**Debug USB**" in ***Impostazioni → Opzioni sviluppatore***

3. Collegare il dispositivo al computer e autorizzare il Debug da USB in Android. Per verificare la connessione è possibile usare **adb** con il seguente comando:

   ```
   adb devices
   ```

4. Effettuare il backup del dispositivo con il seguente comando:

   ```
   adb backup -apk -shared -all
   ```

   Sarà necessario autorizzare il backup in Android.

Alla fine dell'operazione verrà creato il file `backup.ab` con il contenuto salvato.

Registrare il tipo di dispositivo e la versione di build

Per reinstallare in futuro la versione corretta di Android sarà necessario essere in possesso di alcune informazioni sul dispositivo. Queste informazioni sono tutte registrate nel file di sistema `/system/build.prop` del dispositivo, per convenienza si riportano qui i comandi utili al salvataggio di questi dati:

- Per conoscere il tipo di immagine, in un terminale digitare con il comando:

  ```
  adb shell grep ro.product.name system/build.prop
  ```

  Successivamente salvare il valore di **ro.product.name**

- Per conoscere il tipo di dispositivo, digitare:

```
adb shell grep ro.product.device system/build.prop
```

Salvare il valore di **ro.product.device**

- Per conoscere l'identificato del build, digitare:

```
adb shell grep ro.build.id system/build.prop
```

Salvare il valore di **ro.build.id**

5.1.3 Sbloccare il dispositivo

Per installare Ubuntu è necessario sbloccare il dispositivo, si noti che la procedura cancella tutti i dati dallo smartphone.

1. Riavviare in modalità bootloader il dispositivo, da terminale digitare:

```
adb reboot bootloader
```

2. Verificare che il dispositivo venga riconosciuto dall'applicazione fastboot con il seguente comando:

```
fastboot devices
```

3. Sbloccare il dispositivo con il comando:

```
fastboot oem unlock
```

Accettare sul dispositivo le condizioni di sblocco.

4. Riavviare il dispositivo con il comando:

```
fastboot reboot
```

Completare la fase iniziale di installazione di Android inserendo il numero minimo di informazioni. Queste impostazioni verranno cancellate in seguito.

5.1.4 Installare Ubuntu sul dispositivo

Per installare Ubuntu Touch seguire la seguente procedura:

1. Spegnere il dispositivo

2. Riavviare in modalità fastboot utilizzando la combinazione di tasti del proprio dispositivo come riportato nella guida di riferimento di Android[10]

3. Installare Ubuntu Touch in versione di sviluppo con il seguente comando:

   ```
   ubuntu-device-flash --channel=devel --bootstrap
   ```

   L'opzione **channel** specifica il canale di sviluppo da cui scaricare l'immagine.
   L'opzione **bootstrap** è necessaria solo la prima volta che si installa Ubuntu sul dispositivo.

4. Attendere il riavvio del dispositivo.

È utile far notare che l'immagine scaricata viene salvata e mantenuta anche localmente nel proprio computer. Per liberare spazio occupato è sufficiente eseguire il comando:

```
ubuntu-device-flash --clean-cache
```

Abilitare la modalità di sviluppatore

Per abilitare la modalità di sviluppatore in Ubuntu Touch selezionare ***Settings → About this device → Developer mode***.

[10] Booting into fastboot mode https://source.android.com/source/building-devices.html#booting-into-fastboot-mode

5.1.5 Aggiornare il sistema

Per abilitare gli aggiornamenti automatici è sufficiente verificare che ***Settings → Updates*** sia selezionato.

Se gli aggiornamenti automatici fossero disabilitati, il seguente comando mostrerebbe l'aggiornamento da effettuare:

```
adb shell system-image-cli --dry-run
```

In caso si voglia aggiornare il dispositivo in maniera manuale è sufficiente utilizzare il comando:

```
ubuntu-device-flash --channel=CHANNEL
```

Specificando un canale da cui scaricare l'aggiornamento.

5.2 Ubuntu SDK - Software Development Kit

Ubuntu SDK[11] è un Integrated Development Environment (IDE) basato su QtCreator[12] per lo sviluppo di applicazioni in Ubuntu sia in QML che in HTML5.

Fra le proprietà del progetto sono annoverate:

- la semplicità di creare una nuova applicazione grazie ad un wizard guidato
- l'integrazione di Bazaar[13] (e altri sistemi) per il controllo di versione del sorgente
- la possibilità di eseguire le applicazioni sia nel sistema di sviluppo, che in dispositivi collegati o in emulatori di Ubuntu
- gestione e controllo sia dei dispositivi collegati che delle istanze degli emulatori

[11] Ubuntu SDK https://developer.ubuntu.com/en/apps/sdk/
[12] QtCreator homepage http://qt-project.org/wiki/Category:Tools::QtCreator
[13] Bazzar project homepage http://bazaar.canonical.com/en/

- veloce pacchettizzazione del sorgente attraverso il sistema click, per facilitarne la pubblicazione
- cross-compilazione dei pacchetti click con CMake[14] su architetture diverse (come ARM hard float, armhf)
- test di istallazione e di esecuzione dei pacchetti click, sia sui dispositivi collegati che sugli emulatori
- supporto per il debug del codice sorgente

5.2.1 Installazione di Ubuntu SDK

Ubuntu SDK è disponibile per Ubuntu 14.04 (nome in codice Trusty) rilasciato nell'aprile del 2014.

Per l'installazione è consigliato far uso del PPA dedicato allo sviluppo dell'SDK. Seguire il seguente procedimento:

1. Aprire un terminale e digitare il seguente comando:

```
sudo add-apt-repository ppa:ubuntu-sdk-team/ppa
```

2. Aggiornare la lista dei pacchetti e installare Ubuntu SDK:

```
sudo apt-get update && sudo apt-get install ubuntu-sdk
```

L'applicazione è ora installata ed è possibile avviarla tramite la Dash di Ubuntu o tramite riga di comando comando:

```
ubuntu-sdk
```

5.3 Creare uno Scope

In questo capitolo partiremo dallo Scope di default generato dal wizard di Ubuntu SDK, uno Scope sul meteo, e lo modificheremo per effettuare ricerche di libri nella **Biblioteca Universitaria di Ingegneria Dore**.

[14]CMake cross-platform build system http://www.cmake.org/

Per semplicità e visto che Sebina[15], il sistema bibliotecario informatico utilizzato dalla biblioteca, non offre delle API JSON, si è realizzato un applicativo Web parallelo che effettua in back-end ricerche sull'interfaccia Web della biblioteca e ritorna il risultato in un formato conosciuto, appunto JSON.

5.3.1 Sviluppo del Server Web

Si descrive qui l'analisi e l'implementazione del Server Web che semplifica la creazione dell'intero Scope. Si farà uso del meta-linguaggio Contact per analizzare

Requisiti

Il Server Web deve effettuare ricerche nel data base della sola biblioteca universitaria Dore. Deve essere possibile anche cercare l'immagine della copertina di un libro e la disponibilità del libro stesso all'interno della biblioteca. I risultati delle ricerche devono essere disponibili su protocollo HTTP.

Analisi dei requisiti

È necessario identificare due Subject principali:

- un servizio Web/HTTP che mostra i risultati all'utente, chiamato **http**
- uno di che si interfaccia esternamente al database della biblioteca e effettua query su di esso, chiamato **opac**

Analisi del Progetto in Contact

```
ContactSystem SebinaJSONAPI
Subject http;
Subject opac;
// Messages
Request search;
```

[15]Per le Biblioteche, Sebina OpenLibrary http://www.sebina.it/SebinaNet/home.do#1

```
Request cover;
Request availability;
// Highlevel communications
sendSearch: http demand search to opac;
receiveSearch: opac grant search;
sendCover: http demand cover to opac;
receiveCover: opac grant cover;
sendAvailability: http demand availability to opac;
receiveAvailability: opac grant availability;
```

Non sono riportarti i Behaviour dei soggetti in quanto i vari messaggi scambiati fra i due Subject sono serviti *on demand*. Ad ogni messaggio inviato da **http** corrisponde una chiamata a funzione in **opac**.

Il subject **http** è un servizio *always run* e non muore mai.

Il subject **opac** viene creato su richiesta da **http**, effettua delle connessioni esterne verso il database della biblioteca alla ricerca di informazioni, organizza i risultati e li ritorna a **http**, successivamente viene distrutto.

Sviluppo

Si sceglie di sviluppare il Server Web in Python, in quanto offre molta flessibilità per progetti di questo tipo. Si farà uso delle librerie esterne come **stdnum**, per trattare gli ISBN dei libri, e di **pquery** per gestire velocemente l'HTML di ritorno dalle query sul servizio Web della biblioteca.

In Tabella 5.1 è presente una la lista delle query da effettuare sul servizio Web Sebina all'indirizzo base *http://sol.unibo.it/SebinaOpac/Opac*.

Si noti come l'ID della biblioteca sia sempre fisso a `UBOIN` il quale identifica la Biblioteca Dore nel sistema.

API JSON finale

Il codice finale è in esecuzione all'indirizzo *http://doreuniboit.leoiannacone.com*.

Le Api per invocare la ricerca sono di seguito definite.

Azione	Parametri Query	Parametri
Cerca libri	action=search PoloBiblio=UBOIN sort=Anno TitoloBase2={*PATTERN*}	PATTERN: pattern di ricerca
Informazioni su un libro	action=search thNomeDocumento={*BOOK*}	BOOK: id del libro
Disponibilità	action=collocLiv bib=UBOIN thNomeDocumento={*BOOK*}	BOOK: id del libro

Tabella 5.1: Tabella delle query su servizi Sebina

Ricerca di libri La ricerca di libri è stata notevolmente semplificata, è sufficiente lanciare la query `/api?search={`*`PATTERN`*`}` per ritornare una lista di oggetti JSON che rappresentato ogni singolo libro trovato nella libreria Dore.

Segue un esempio di rappresentazione del libro nel suddetto formato:

```
{
    "isbn": "9788871925400",
    "description": "I moderni sistemi operativi / Andrew S. Tanenbaum ...",
    "author": "Tanenbaum, Andrew S. <1944- >",
    "url": "http://sol.unibo.it/SebinaOpac/.do?idopac=UBO2740955",
    "title": "I moderni sistemi operativi",
    "year": "2009",
    "id": "UBO2740955T"
}
```

Immagine di copertina È possibile ottenere l'immagine di copertina di un libro tramite la query `/api/cover/{`*`ISBN`*`}`. Il server effettua una ricerca su un servizio esterno (www.excalibooks.com, servizio esterno che colleziona

copertine di libri) e risponde con HTTP 301 Redirect, ovvero re-direziona la richiesta verso quest'ultimo.

Disponibilità del testo Per avere informazioni sulla disponibilità di un testo è sufficiente effettuare la query `/api/availability/{BOOK_ID}`. Il servizio ritorna un immagine, un pallino, che se di colore verde indica la disponibilità del libro in biblioteca, se di colore rosso il contrario.

5.3.2 Creazione del progetto in Ubuntu SDK

Si descrive qui lo sviluppo di un nuovo Scope, basato su quello predefinito che crea Ubuntu SDK, che si connette al nostro Server Web per la ricerca di libri.

Aprire Ubuntu SDK e creare un nuovo progetto.

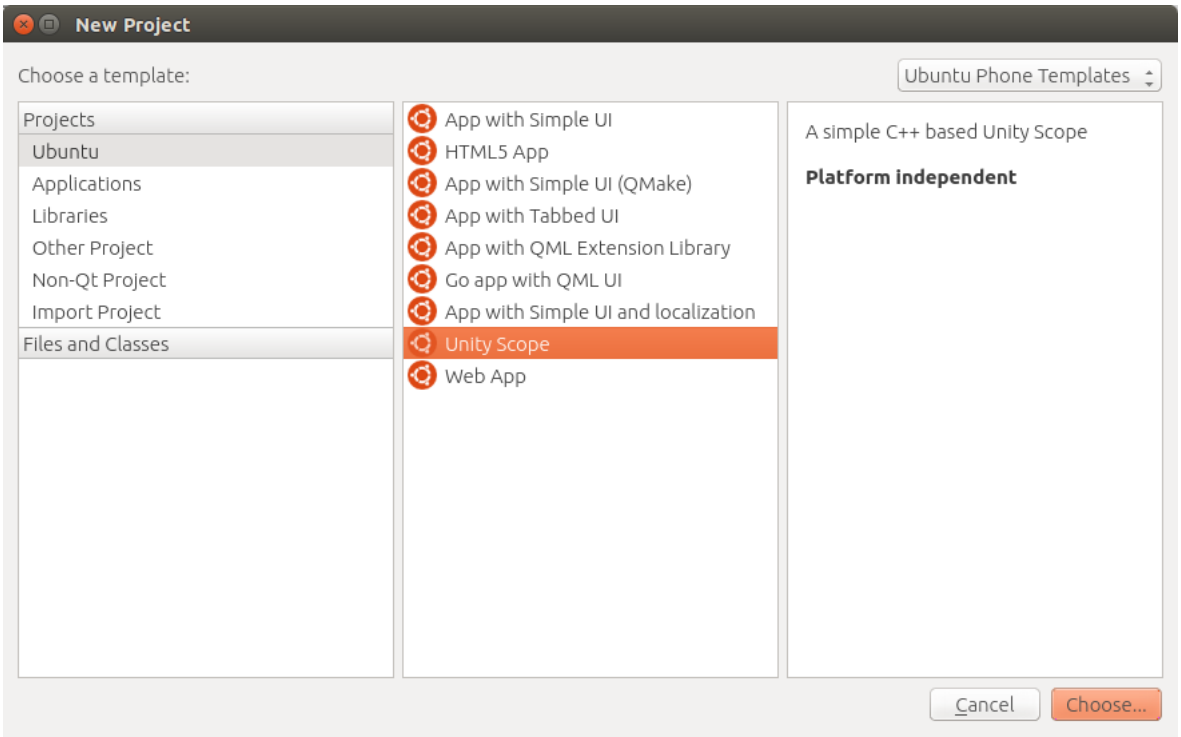


Figura 5.1: Creazione di un progetto in Ubuntu SDK

Scegliere **Unity Scope** nel wizard di creazione progetto e inserire successivamente il nome del progetto *doreuniboit*, lasciare le impostazioni predefinite nel resto dei campi. Nella pagina di selezione dei **Kits** scegliere **Desktop**.

È possibile eseguire il progetto utilizzando la scorciatoia da tastiera `Ctrl+R`.

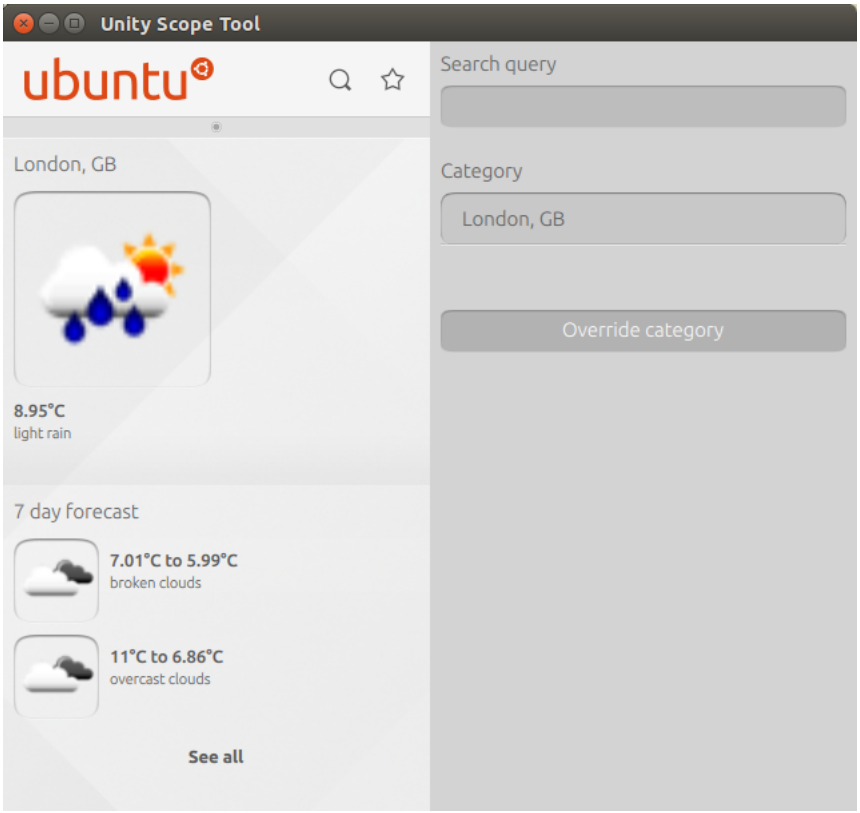


Figura 5.2: Anteprima dello Scope di default di Ubuntu SDK

Lo Scope di default di Ubuntu SDK permette la ricerca di condizioni meteo su sito *openweather.org*.

5.3.3 Pulizia del codice base

Dal codice automaticamente generato si eliminano le parti non necessarie, quelle relative alle condizioni meteo, in modo da estrapolare un template base.

Il file src/Scope/query.cpp

Questo file contiene la definizione di categorie e la trasformazione dei risultati in "result cards", qui è dove la UI manda la propria query e si aspetta una lista di risultati.

Rimuovere quindi le definizioni di WEATHER_TEMPLATE e CITY_TEMPLATE, inutili per il nostro Scope.

La funzione Query::run() è la funzione invocata esternamente e dove la query viene eseguita. Rimuovere quindi tutto il codice dopo:

```
string query_string = alg::trim_copy(query.query_string());
```

fino a:

```
} catch (domain_error &e) {
```

Nello stesso file rimuovere anche la seguente parte di codice:

```
// Open weather map API error code can either be a string or int
QVariant cod = root.toVariant().toMap()["cod"];
if ((cod.canConvert<QString>() && cod.toString() != "200")
        || (cod.canConvert<unsigned int>() && cod.toUInt() != 200)) {
    throw domain_error(root.toVariant().toMap()["message"].toString().toStdString()
}
```

Il file src/api/client.cpp

Rimuovere interamente la funzione `Client::Current Client::weather()` (è dove l'URI è costruito e il risultato salvato nella variabile `weather`) e la funzione `Client::Forecast Client::forecast_daily()`.

Il file include/api/client.h

Qui c'è la definizione delle classi che abbiamo eliminato, occorre quindi aggiornare anche questo file rimuovendo le strutture e i tipi per `Forecast`, `Current`, `WeatherList`, `Weather`, `Temp` e `City`. Rimuovere anche:

```
virtual Current weather(const std::string &query);
virtual Forecast forecast_daily(const std::string &query, unsigned int days = 7)
```

5.3.4 Inserimento del nuovo codice

Si riportano qui di seguito le modifiche da inserire nei vari file con il codice relativo al nostro Scope.

Il file src/api/config.h

Modificare l'assegnamento della variabile `apiroot`, che indica il server a cui collegarsi:

```
std::string apiroot { "http://doreuniboit.leoiannacone.com" };
```

Opzionalmente è possibile modificare anche la variabile `user_agent` come segue:

```
std::string user_agent { "dore-unibo-it-scope 0.1; (training)" };
```

Il file include/api/client.h

È possibile qui inserire la struttura dei propri dati. Avremo quindi bisogno di inserire la definizione di strutture come Book, BookList e BookRes (per salvare una lista di risultati). Dopo le seguenti righe:

```
class Client {
public:
```

aggiungere:

```
/**
 * The Book object.
 */
struct Book {
```

```
    std::string id;
    std::string isbn;
    std::string title;
    std::string author;
    std::string year;
    std::string url;
    std::string description;
    std::string cover_img;
    std::string available_img;
};
/**
 * A list of Book objects.
 */
typedef std::deque<Book> BookList;
/**
* Book results.
*/
struct BookRes {
    BookList books;
};
/**
 * Get the book list for a query
 */
virtual BookRes books(const std::string &query);
```

Il file src/api/client.cpp

Prima della seguente riga:

```
http::Request::Progress::Next Client::progress_report(
```

Aggiungere:

```
Client::BookRes Client::books(const string& query)
{
    QJsonDocument root;
    get( { "api" }, { { "search", query } }, root);
    BookRes result;
    QVariantList variant = root.toVariant().toList();
    for (const QVariant &i : variant) {
        QVariantMap item = i.toMap();
        auto id = item["id"].toString().toStdString();
        auto isbn = item["isbn"].toString().toStdString();
        // We add each result to our list
        result.books.emplace_back(
        Book {
            id,
            item["isbn"].toString().toStdString(),
            item["title"].toString().toStdString(),
            item["author"].toString().toStdString(),
            item["year"].toString().toStdString(),
            item["url"].toString().toStdString(),
            item["description"].toString().toStdString(),
            config_->apiroot + "/api/cover/" + isbn + ".jpg",
            config_->apiroot + "/api/availability/" + id + ".jpg"
        }
        );
    }
    return result;
}
```

Abbiamo così costruito una lista di risultati contenente elementi di tipo Book.

Il file src/Scope/query.cpp

È ora momento di definire il layout dello Scope. Prima della definizione della funzione `Query::Query`, aggiungere:

```
/**
* Define the layout for the books results
*/
const static string BOOKS_TEMPLATE =
   R"(
       {
           "schema-version": 1,
           "template": {
               "category-layout": "grid",
               "card-layout": "horizontal",
               "card-size": "large"
           },
           "components": {
               "title": "title",
               "art" : {
                   "field": "art"
               },
               "subtitle": "author",
               "attributes": {
                    "field": "attributes", "max-count": 2
               }
           }
       }
   )";
```

Il file src/Scope/preview.cpp

Modificare l'anteprima di un libro modificando la funzione `Preview::run()` come segue:

```
void Preview::run(sc::PreviewReplyProxy const& reply) {
    // Support three different column layouts
    sc::ColumnLayout layout1col(1), layout2col(2), layout3col(3);
    // Single column layout
    layout1col.add_column( { "image", "header", "summary", "buttons" });
    // Two column layout
    layout2col.add_column( { "image" });
    layout2col.add_column( { "header", "summary", "buttons" });
    // Three cokumn layout
    layout3col.add_column( { "image", "buttons" });
    layout3col.add_column( { "header", "summary" });
    layout3col.add_column( { "buttons" });
    // Register the layouts we just created
    reply->register_layout( { layout1col, layout2col, layout3col });
    // Define the header section
    sc::PreviewWidget header("header", "header");
    // It has title and a subtitle properties
    header.add_attribute_mapping("title", "title");
    header.add_attribute_mapping("subtitle", "author");
    header.add_attribute_mapping("attributes", "attributes");
    // Define the image section
    sc::PreviewWidget image("image", "image");
    // It has a single source property, mapped to the result's art property
    image.add_attribute_mapping("source", "art");
    // Define the summary section
    sc::PreviewWidget description("summary", "text");
    // It has a text property, mapped to the result's description property
    description.add_attribute_mapping("text", "description");
```

```
    sc::PreviewWidget buttons("buttons", "actions");
    sc::VariantBuilder builder;
    builder.add_tuple({
        {"id", sc::Variant("open")},
        {"label", sc::Variant(_("See in the browser"))}
    });
    buttons.add_attribute_value("actions", builder.end());
    // Push each of the sections
    reply->push( { image, header, buttons, description });
}
```

5.3.5 Esecuzione e anteprima

Premendo ora la scorciatoia da tastiera `Ctrl+R` in Ubuntu SDK verrà eseguito lo Scope nell'anteprima Desktop. Seguono delle immagini di anteprima per mostrare il lavoro compiuto.

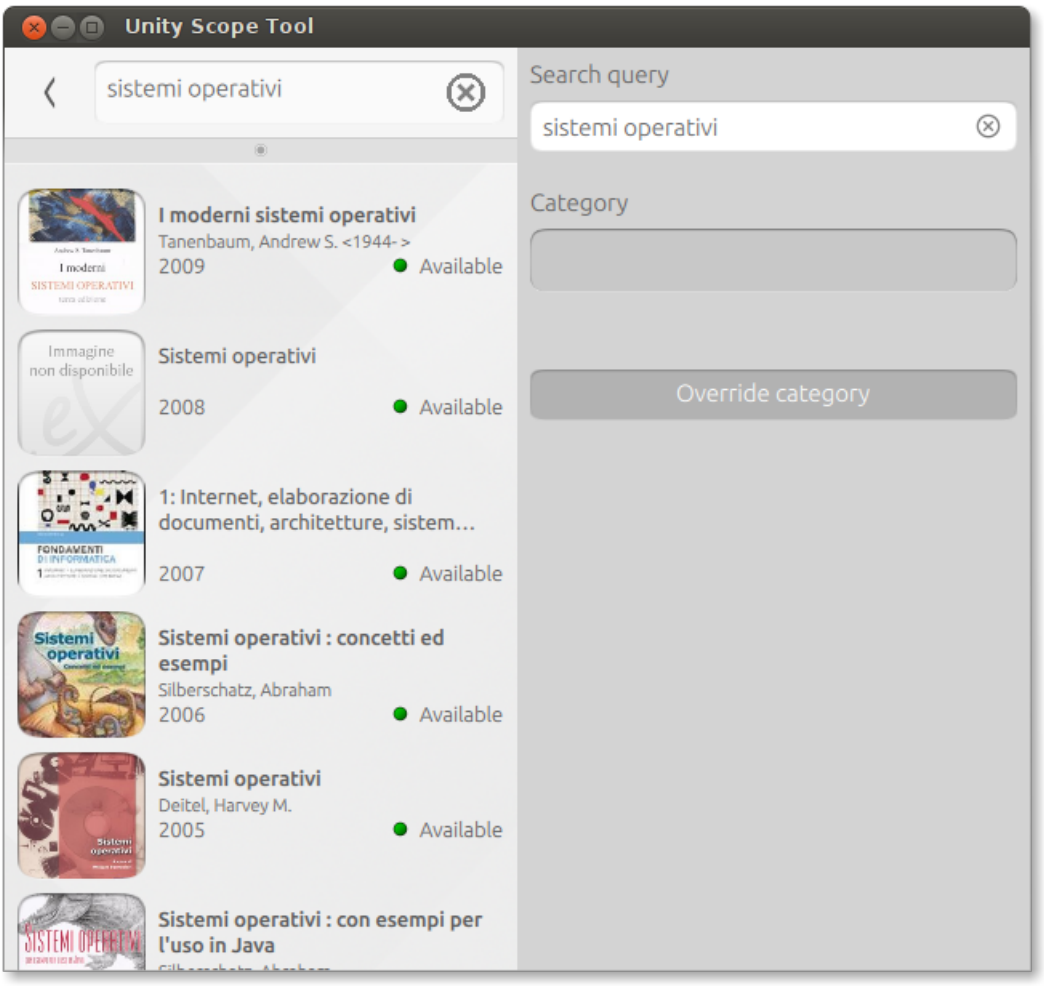


Figura 5.3: Anteprima dello Scope - Vista Risultati

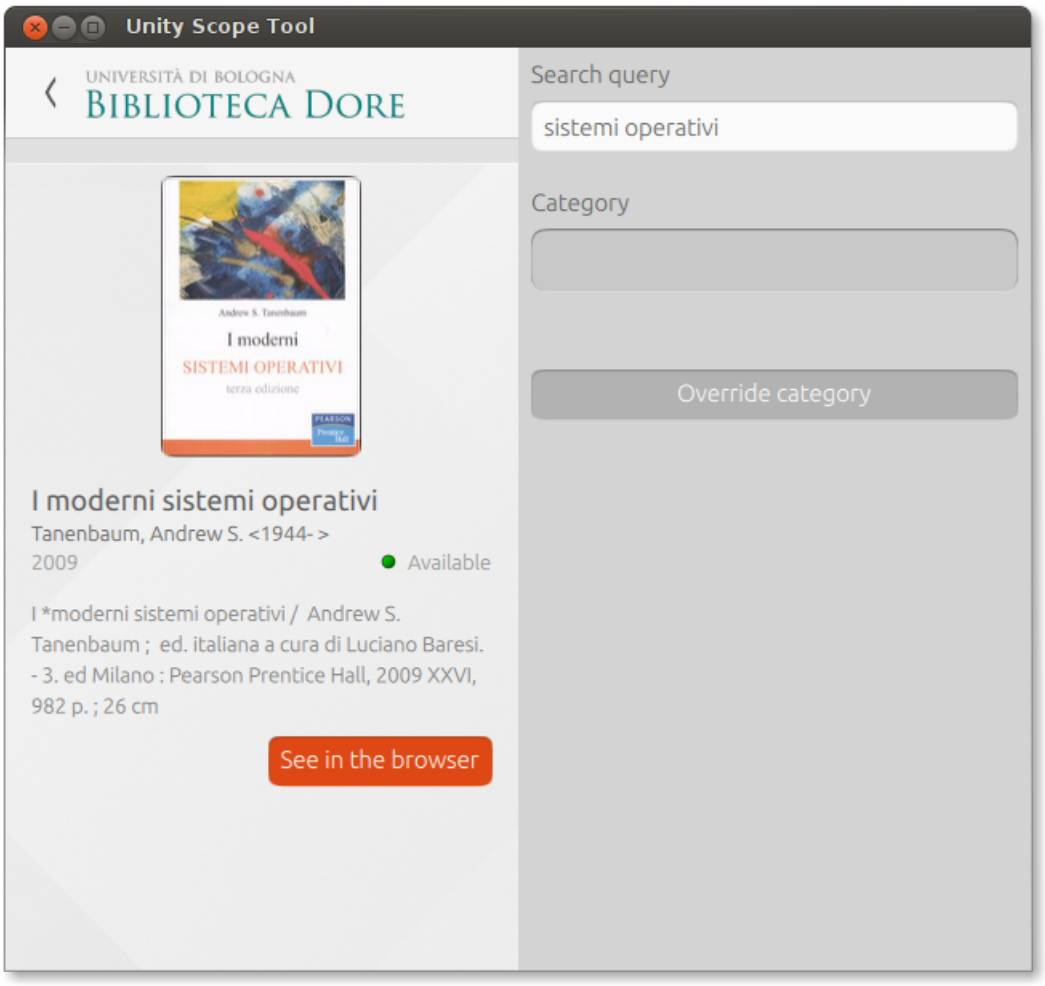


Figura 5.4: Anteprima dello Scope - Vista Libro

Capitolo 6

Conclusioni

Sebbene l'entusiasmo dietro lo sviluppo della piattaforma sia stato molto alto, superando persino la soglia psicologica dei 10 milioni di dollari nella campagna di crowfunding del 2013[1], Ubuntu Touch sembra essere destinato ad una stretta cerchia di utenti. Non vi sono attualmente casi di studio oggettivi che vedano questo sistema operativo coinvolto in realtà aziendali al di fuori di quella Canonical, l'azienda che tira le redini dello sviluppo di Ubuntu.

Tuttavia il progetto è relativamente giovane, il primo dispositivo ad essere venduto con Touch preinstallato ha visto la luce solo nel febbraio 2015 ed altri nuovi modelli sono in arrivo, è quindi forse troppo presto per trarre conclusioni di marketing. Inoltre, gli smartdevice odierni non sono ancora abbastanza potenti per sostituire completamente una workstation, tutto a discapito della ricerca della convergenza pura fra dispositivi che ha guidato la vision di Ubuntu negli ultimi anni. A tal proposito è utile sottolineare che la vera milestone fissata da Canonical è prevista per l'aprile del 2016 con l'uscita della versione 16.04, quando la nuova Unity (v8, riscritta completamente) sarà disponibile come interfaccia predefinita anche sui Desktop, come confermato da Micheal Hall, ingegnere presso Canonical, nell'ottobre 2014[2].

[1] Ubuntu Edge $32 million crowfunding campaign https://www.indiegogo.com/projects/ubuntu-edge

[2] Unity 8 default in Ubuntu Desktop 16.04 LTS http://mhall119.com/2014/10/unity-

Le applicazioni presentano interessanti punti di innovazione. Se da un lato si avvantaggiano dei decennali punti di forza delle distribuzioni Linux, come il nuovo gestore dei pacchetti Click che è basato su dpkg di Debian, dall'altro lato abbracciano le più nuove tecnologie integrandole completamente. Le applicazioni in Ubuntu sono infatti scritte utilizzando il linguaggio QML e le tecnologie web (HTML5 e JavaScript) di ultima generazione.

La scelta di questi linguaggi risulta fondamentale nell'attrarre nuovi sviluppatori nell'ecosistema Ubuntu. Una scelta necessaria e doverosa che abbraccia l'ondata di modernità degli ultimi anni. Secondo le statistiche di GitHub, il famoso container di Git che vanta circa 3,4 milioni di utenti, JavaScript è stato infatti il linguaggio di programmazione più utilizzato degli ultimi 24 mesi[3], mentre le librerie del framework Qt (su cui si interfaccia QML) spopolano ormai fra le aziende di tutto il mondo[4], come Panasonic Avionics, Bluescape, Adeneo e molte altre.

Ma il vero punto di forza per Ubuntu Touch sono gli Scope: non semplici applicazioni, ma nuovi motori di ricerca nel dispositivo. Semplici, leggeri e veloci da sviluppare, gli Scope rappresentano l'unica novità che Ubuntu introduce nella propria esperienza utente. Per capirli a fondo e per confermare la facilità e la velocità di sviluppo annoverata dagli ingegneri di Canonical, nel lavoro di questa tesi si è scelto di svilupparne uno interamente finalizzato alla ricerca di volumi disponibili presso la Biblioteca di Ingegneria "Gian Paolo Dore". Il lavoro, dopo un primo impatto brusco verso C++ (il linguaggio di programmazione degli Scope), è risultato stimolante e produttivo, in poco tempo si era già in possesso di un primo prototipo del software pronto per l'anteprima. Il codice finale, presente nel dispositivo di archiviazione allegato a questo documento, è stato rilasciato sotto licenza libera (GPL v3) ed è in fase di pubblicazione negli archivi ufficiali di Ubuntu.

8-desktop/

[3]Explore the complexity of the universe of programming languages used across the repositories hosted on GitHub http://githut.info/

[4]Leading companies in over 70 industries use Qt to power millions of devices and applications. http://www.qt.io/qt-in-use/

L'utilizzo di Ubuntu SDK, l'IDE predefinito per lo sviluppo in Ubuntu, è risultato infine fluido e intuitivo, seppur pecchi ancora della mancanza di qualche feature fondamentale, come una suite di refactory per velocizzare il processo di produzione del software in caso di cambiamenti strutturali. Tuttavia l'impressione è sommariamente positiva, specialmente per la gestione degli emulatori di dispositivi mobile, sui quali viene eseguito Ubuntu Touch per il testing e il debug delle proprie applicazioni. Una positività però non sempre riscontrata nell'utilizzo della documentazione ufficiale, che è a tratti datata e con informazioni obsolete, e a tratti incompleta e discordante con la vera implementazione del software e delle API. Durante la stesura del documento è stato infatti necessario più volte ricorrere alla lettura del codice sorgente per avere un'informazione sia dettagliata sia veritiera delle proprietà descritte dai manuali di Ubuntu. A tal proposito, sono stati riportati numerosi bug per la segnalazione di incongruenze fra documentazione e codice, come ad esempio i numeri #1424720[5] e #1424723[6], che descrivono l'errata implementazione di una classe e della relativa documentazione che ne suppone la correttezza.

Lo sviluppo è risultato tuttavia piacevole, ho avuto modo di conoscere persone molto interessanti della famosa comunità di Ubuntu, e di entrare in contatto diretto con gli ingegneri di Canonical, entrambi sempre disponibili a rispondere a qualsiasi domanda o dubbio gli venga posto nei classici canali di comunicazione, come ad esempio via IRC o via email. È sicuramente questa la vera forza, non solo di Ubuntu, ma dell'intero mondo dell'Open Source: una filosofia che accomuna gente da tutte le parti del mondo, con culture diverse, fusi orari diversi, storie diverse. Gente che si riunisce in comunità virtuali e che, unite dal solo interesse di fare informatica, dà vita ogni giorno a nuovi entusiasmanti progetti, regalandoci righe di codice libero, aperto a chiunque, pronto per la lettura e la condivisione, destinato ad ingrandire prima la conoscenza del singolo e poi, per diretta conseguenza, quella globale,

[5]Add emblem to PreviewHeader widget https://bugs.launchpad.net/bugs/1424720
[6]Wrong doc for PreviewHeader widget https://bugs.launchpad.net/bugs/1424723

perché in fondo "Io sono ciò che sono per merito di ciò che siamo tutti"[7].

[7]Significato della parola africana Ubuntu http://it.wikipedia.org/wiki/Ubuntu_(filosofia)

Elenco delle figure