

POLITECNICO DI BARI

Corso di Laurea in Ingegneria dell'Automazione

Tesi di Laurea Magistrale in Internet of Things

Piattaforme di Crowdsourcing in sistemi IoT eterogenei

Dipartimento di Ingegneria Elettrica e dell'Informazione



Relatori:

prof. Luigi Alfredo Grieco
prof. Giuseppe Piro

Candidato:
Antonio Brandi

17 Ottobre 2019

*Solo coloro che sono abbastanza folli da pensare di cambiare il mondo,
alla fine lo cambiano per davvero*

Introduzione

In questo lavoro di tesi si intende esplorare le funzionalità e potenzialità offerte dalla rapida ascesa delle tecnologie disponibili nel mondo dell' Internet of Things (IoT).

A seguito di una dettagliata analisi dello stato dell'arte e dei campi di applicazione che maggiormente potrebbero beneficiare del rapido sviluppo delle tecnologie IoT, ci si concentrerà maggiormente sul settore della mobilità, con il fine ultimo di sviluppare un prototipo dimostrativo delle potenzialità offerte da una piattaforma di Crowdsourcing.

L'output di questo lavoro di tesi sarà quindi un prototipo di una Applicazione sviluppata per smartphone Android che consenta la raccolta dei dati provenienti dai sensori integrati nello smartphone stesso e la condivisione di questi dati in una piattaforma Cloud. Tale condivisione dei dati relativi alla navigazione stradale si rende necessaria in quanto, solo attraverso la condivisione e raccolta di una grossa mole di dati, possono essere adottate delle tecniche di analisi e predizione avanzate che potrebbero consentire una migliore qualità della navigazione ed una più pronta reazione delle autorità a particolari eventi.

I protocolli di condivisione dei dati raccolti dallo smartphone saranno uno dei punti salienti di questo lavoro di tesi in quanto saranno analizzate diverse tecnologie per una comunicazione Client/Server tra dispositivi eterogenei e che ulteriormente siano scalabili ed adatte alla condivisione di dati relativi al traffico.

Nella fattispecie, l'intento della fase di prototipazione non sarà quello di descrivere dettagliatamente un prodotto finito, ma sarà piuttosto quello di mostrare il flusso logico che ha portato ad intraprendere alcune scelte progettuali piuttosto che altre e, ulteriormente, quello di mostrare la logica che ha portato alla realizzazione della architettura hardware e software del prototipo.

Il vantaggio di questo approccio è quello di poter riutilizzare le stesse logiche progettuali intraprese in questo lavoro di tesi, anche per altri progetti legati al mondo dell'IoT che condividano le stesse necessità di raccolta decentralizzata dei dati e condivisione degli stessi su piattaforme Cloud.

Nel Capitolo 1, a seguito di una breve panoramica del mondo e dei numeri dell' Internet Of Things, saranno mostrate alcune delle tecnologie legate al mondo dell'IoT e che si intendono utilizzare per la realizzazione del prototipo.

Nel Capitolo 2, ci si soffermerà invece sulla spiegazione del significato del termine Crowdsourcing e lo si valuterà applicato al mondo dell'IoT corredato da esempi di applicazioni che si basano proprio su piattaforme di Crowdsourcing.

Nel Capitolo 3, ci si focalizzerà sulla fase di Progettazione di tutte le componenti del prototipo spiegando, per ciascuna fase, le ragioni che abbiano portato ad alcune scelte progettuali piuttosto che altre. Questo capitolo, seguirà quindi il processo logico della implementazione di un nuovo sistema o applicativo legato al mondo dell'IoT e quindi, piuttosto che mostrare nei dettagli l'implementazione di ciascuna scelta progettuale, se ne mostrerà solo la logica e le motivazioni che hanno influenzato la scelta. Sebbene in questo capitolo sono mostrate delle brevi guide per l'accesso o l'utilizzo di alcune componenti, queste sono di carattere generale e non sono in alcun modo limitate al solo prototipo in fase di progettazione.

Le componenti che sarà necessario progettare in questo capitolo saranno quindi:

- Protocolli di comunicazione di basso livello
- Standard di formattazione dei dati
- Architettura Software
- Piattaforme di Crowdsourcing per dispositivi IoT
- Architettura del Cloud Server
- Architettura del Cloud Database

Nel Capitolo 4, infine, sulla base delle scelte progettuali intraprese nel Capitolo 3, si andrà ad implementare il sistema. Questo capitolo, pertanto, è molto specifico per l'applicativo progettato in questo lavoro di tesi e risulta anche molto specifico in termini di linguaggio di programmazione utilizzato, piattaforma di sviluppo e piattaforme server. Il fine ultimo di questa fase sarà quello di mostrare la struttura implementativa del prototipo in funzione della quale poter fornire delle linee guida implementative e possibili sviluppi futuri.

Indice

Introduzione	IV
1 Introduzione all'Internet of Things	1
1.1 I Numeri dell'IoT	2
1.2 IoT Enabling Technologies	6
1.2.1 Il Protocollo IPv6	7
1.2.2 Molte applicazioni, diversi protocolli	9
1.2.3 L'Application Layer per i dispositivi IoT	9
1.3 Applicazioni IoT	12
1.3.1 Home	12
1.3.2 Healthcare	13
1.3.3 Agriculture	13
1.3.4 Cities	14
1.3.5 Energy	15
1.3.6 Retail	15
1.3.7 Automotive	16
2 Sistemi di Crowdsourcing	18
2.1 Applicazioni di Crowdsourcing	19
2.1.1 Google Maps	20
2.1.2 Linux e Wikipedia	21
2.2 Crowdsourcing ed Internet of Things	21
3 Progetto del Sistema	24
3.1 Idea Generale	24
3.2 La Raccolta dei Dati	26
3.2.1 Accesso ai Sensori di uno Smartphone	28
3.2.2 Parametri di Qualità dei Sensori	30
3.2.3 Filtraggio delle Letture dei Sensori	31
3.3 Il Protocollo DATEX	34
3.3.1 DATEX II Data Model	36
3.3.2 XML Schema	39
3.3.3 Exchange Mechanisms	39

3.3.4	Exchanged Data	40
3.3.5	Publication di Elementi Base	42
3.3.6	Utilizzo del DATEX II Data Model	43
3.4	IoT Cloud Platform	49
3.4.1	AWS IoT Core Overview	51
3.4.2	AWS IoT Core Development	54
3.5	Database Architecture	60
3.5.1	Database Relazionali e Non Relazionali	62
3.5.2	Query Geo-Referenziate	63
3.5.3	Amazon DynamoDB	65
4	Implementazione del Sistema	71
4.1	Applicazione Android	73
4.1.1	MapsActivity	74
4.1.2	SensorActivity	83
4.1.3	EventActivity	89
4.1.4	AWSActivity	91
4.2	AWS IoT Core	98
4.3	DynamoDB	100
5	Conclusioni	105
A	Utilizzo del Protocollo DATEX II	107
B	Implementazione del Protocollo DATEX II	117
	Riferimenti bibliografici	124

Elenco delle figure

1.1	Overview del paradigma IoT	1
1.2	Gli oggetti collegati ad Internet possono essere progettati per una grande varietà di applicazioni [4]	2
1.3	Number of IoT devices roadmap	3
1.4	Previsioni della rapida crescita sul numero di oggetti connessi ad Internet [37]	4
1.5	Impatto economico di applicazioni IoT nel 2025 fornito dal McKinsey Global Institute [14]	4
1.6	Campi di applicazione nei quali si stima il maggiore sviluppo reso possibile dall'IoT [37]	5
1.7	Evoluzione dello spazio destinato all'indirizzamento [13]	7
1.8	Confronto tra l'header nei protocolli IPv4 ed IPv6 [13]	8
1.9	Modello TCP-IP (destra) in confronto con il modello ISO-OSI (sinistra) [7]	10
1.10	Dispositivi con limitate capacità computazionali, RAM e ROM	11
1.11	Comunicazione attraverso il protocollo MQTT [17]	12
1.12	Dispositivi comunicano tra di loro nell'ecosistema casa	13
1.13	Esempi di soluzioni per l'ecosistema smart home	13
1.14	Soluzione per il monitoraggio dell'asma (sinistra) e per il monitoraggio del glucosio per pazienti affetti da diabete (destra)	14
1.15	Bosch Deepfield Connect Project	14
1.16	Il concept delle smart grid	15
1.17	Amazon Go store nella città di Seattle	16
1.18	The IoT Connected Car	16
2.1	Esempio dello sviluppo di un sistema IoT di analisi utilizzando MATLAB, Machine Learning e ThingSpeak [19]	19
2.2	Indicazione sul livello di traffico stradale nella città di Boston	20
3.1	Schema del prototipo. a) Raccolta dei dati attraverso un sistema eterogeneo di dispositivi. b) Condivisione dei dati in un Middleware IoT. c) Storing dei dati all'interno di Databases d) Analisi sui dati. d) Query Geo-Referenziata sulle condizioni di traffico.	25
3.2	Principio di funzionamento dei sensori attualmente in uso per la misura del livello di traffico	26

3.3	Crescita della disponibilità di sensori nell'equipaggiamento di uno smartphone fonte: CornerAlliance	27
3.4	Schema generale di un sistema di misura	29
3.5	Estratto dello stack utilizzato nel sistema operativo Android per rendere disponibili le letture dei sensori ad applicazioni e servizi	29
3.6	Differenza tra la definizione di Precisione e quella di Accuratezza di una misura	30
3.7	Principio con il quale nel sistema operativo Android è possibile recuperare le letture di un sensore ed utilizzarle in una applicazione. [18]	32
3.8	Sensore di accelerazione a bordo di uno smartphone	33
3.9	Panoramica del Mondo ITS e le relative sfide tecnologiche	35
3.10	Workflow di un processo di conversione automatico	39
3.11	Fasi del Processo Unificato	44
3.12	Distinzione effettuata dallo Standard DATEX II tra metodo di comunicazione (<i>Exchange</i>) e formato del Payload (<i>PayloadPublication</i>)	47
3.13	Formato del Payload	47
3.14	Formato della Publication	48
3.15	Implementazione della Publication	48
3.16	Schema della componente che si vuole implementare in questa sezione vista nel progetto completo	49
3.17	Capacità di interfacciamento della piattaforma IoT Core con gli altri servizi dell'ecosistema AWS	52
3.18	Esempi di utilizzo della piattaforma IoT Core combinata con altri servizi dell'ecosistema AWS	53
3.19	Struttura della comunicazione	53
3.20	Processo di autenticazione implementato in ogni punto della connessione . .	54
3.21	Primo accesso alla piattaforma IoT Core	55
3.22	Creazione di un nuovo oggetto nella pagina principale di AWS IoT Core. . .	56
3.23	Panorama della frammentazione nella distribuzione delle versioni del sistema operativo Android	57
3.24	Panoramica della schermata Test	58
3.25	Sottoscrizione al Topic EU_WEST_2	58
3.26	Pubblicazione nel Topic EU_WEST_2	59
3.27	Ricezione del messaggio	59
3.28	Esempio dell'utilizzo dell'SDK messo a disposizione da AWS in Android per lo sviluppo di Lambda Application	60
3.29	Esempio di un database relazionale	62
3.30	Esempio di un database non relazionale	63
3.31	Architettura di OpenGeoBase [32]	64
3.32	Range-Query ed approcci di suddivisione della griglia basata su tre livelli 0,1,2 [32]	64
3.33	Schema di una tabella in DynamoDB evidenziando i campi <i>partition key</i> e <i>sort key</i>	65

3.34 Suddivisione di un Database in DynamoDB in tabelle e suddivisione delle tabelle in Partitions	66
3.35 Visualizzazione della Regola sul database DynamoDB	67
3.36 Creazione di una nuova tabella	68
3.37 Creazione di un nuovo ruolo	69
3.38 Visualizzazione della nuova entry nella tabella	70
4.1 Diagramma dei casi d'uso elaborato sulla base dei requisiti funzionali del sistema Tabella 3.2	72
4.2 Schema delle Activity che compongono la applicazione Android	73
4.3 Moduli utilizzati da ciascuna Activity	74
4.4 Overview della nuova directory creata da Android Studio	74
4.5 Layout della Activity iniziale nell'applicazione Android	75
4.6 Lista Sensori con le relative misure	84
4.7 Layout della EventActivity	89
4.8 Layout della AWSActivity	91
4.9 Lista di tutti i topic supportati da AWS IoT Core suddivisi per area geografica.	99
4.10 Regola per la scrittura sul Database DynamoDB	99
A.1 Distinzione effettuata dallo Standard DATEX II tra metodo di comunicazione (<i>Exchange</i>) e formato del Payload (<i>PayloadPublication</i>)	107
A.2 Implementazione della Publication	108
A.3 Implementazione del' Exchange	108
A.4 Location reference	109
A.5 Elaborated Data Publication structure	109
A.6 Elaborated Data Implementation	110
A.7 Measured Data Structure	110
A.8 Measurement Site Record Structure	111
A.9 Basic Data Default Implementation	111
A.10 Humidity Implementation	112
A.11 Temperature Implementation	112
A.12 Visibility Implementation	113
A.13 Wind Implementation	113
A.14 Situation Publication Structure	114
A.15 Impact Implementation	114
A.16 Traffic Element Implementation	115
A.17 Validity Implementation	115
A.18 Level B extension to the DATEX II Standard	116

Elenco delle tabelle

1.1	Previsione dell'andamento globale del mercato di prodotti e servizi legati al mondo IoT [43]	5
1.2	Le metodologie di comunicazione [35]	6
3.1	Le metodologie di comunicazione [15]	35
3.2	Analisi dei requisiti funzionali seguendo il modello di sviluppo UP	46
3.3	Analisi dei requisiti che la IoT Cloud Platform deve supportare	50
3.4	Confronto tra Cloud Platforms	51
3.5	Analisi dei requisiti che la IoT Cloud Platform deve supportare	61

Listings

3.1	Pseudo-codice relativo all'implementazione di un filtro passa-basso	33
3.2	Esempio di un possibile messaggio in formato JSON da salvare nel database	69
4.1	Mostro a schermo il layout che ospiterà la mappa	76
4.2	Listener associato agli elementi del Menu	76
4.3	Accesso al GPS e connessione Internet	77
4.4	Utilizzo della chiave fornita da Google Maps per l'utilizzo delle Mappe . . .	78
4.5	Mapping delle tabelle ed attributi del Database DynamoDB che si vuole mappare	79
4.6	<struttura del Task Asincrono che dovrà interrogare il Database	80
4.7	Calcolo dei parametri con i quali inviare una query geo-referenziata	81
4.8	Composizione ed invio della query	81
4.9	Ricezione degli item che soddisfano la query	82
4.10	Visualizzazione di ciascun evento come marker sulla mappa	83
4.11	Creazione del riferimento al SensorManager che sarà condiviso tra tutti i sensori mostrati nella Activity	85
	code/sensor_2.java	85
4.12	Creazione e lancio del thread separato che leggerà i valori dall'accelerometro.	86
4.13	Registrazione del Listener sull'accelerometro.	86
4.14	Modulo Sensor Event Listener.	87
4.15	Funzione applicata all'output dell'accelerometro per il filtraggio dei dati con un filtro passa-basso.	87
4.16	Accesso al contenuto degli elementi della view nella SensorActivity.	87
4.17	Lettura delle caratteristiche salienti del sensore.	88
4.18	Gestione del click sul primo pulsante.	89
4.19	Utilizzo della libreria XStream per la conversione della publication in for- mato XML.	90
4.20	Utilizzo della libreria Gson per la conversione della Publication in formato JSON.	91
4.21	Variabili necessarie alla autenticazione del Client.	92
4.22	Generazione di un Client ID.	93
4.23	Utilizzo di AWS Cognito per l'accesso ad AWS IoT Core.	94
4.24	Connessione con l'AWS IoT Core.	96
4.25	Invio della Publication al Server.	97
4.26	Payload del messaggio in fromato JSON.	100

B.1	Implementazione della classe Publication che sarà il contenitore di tutto il pacchetto trasmesso.	117
B.2	Implementazione della classe Exchange che conterrà informazioni utili alla comunicazione	117
B.3	Implementazione della classe PayloadPublication che conterrà tutti i dati prodotti dalla applicazione	119
B.4	Esempio di output della applicazione in formato XML in rispetto dello standard DATEX II	121

Capitolo 1

Introduzione all'Internet of Things

Il panorama mondiale ha, nell'ultimo decennio, riposto particolare attenzione ed interesse verso le emergenti tecnologie nel campo dell'Internet of Things (IoT).

Questo perchè, il paradigma IoT consente oggi di realizzare la vision ed il concept che era già stato supposto e predetto come sviluppo della rete Internet. Ovvero adempiere allo scopo di collegare il mondo (Figura 1.1).

Infatti, se grazie ad Internet è stato possibile collegare le persone, metterle in relazione, fornire ad essi gli strumenti per interagire tra loro, indipendentemente dalla loro posizione geografica o razza o sesso o colore, allo stesso modo, il naturale sviluppo di Internet non può che essere quello di mettere in relazione le cose. I dispositivi che siamo abituati ad utilizzare ogni giorno e che, seppur diversi ed eterogenei tra loro in termini di funzionalità, costo e posizione geografica, sono in grado di comunicare tra loro, scambiarsi dati ed informazioni al fine di cambiare definitivamente il nostro modo di lavorare, acquistare, consumare e quindi di vivere.

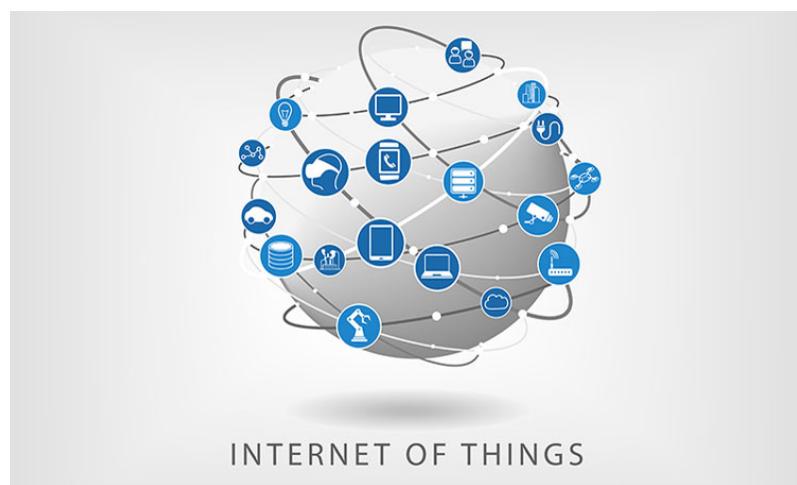


Figura 1.1: Overview del paradigma IoT

Per rendere ancora più evidente il livello innovativo introdotto dal paradigma IoT, è necessario prima effettuare una chiara distinzione tra Internet ed il World Wide Web, spesso confusi. Con il termine Internet ci si riferisce al livello fisico dello stack protocollare realizzato da switch, routers e altri strumenti che consentono lo scambio di dati da un punto ad un altro. Per World Wide Web si intende piuttosto un livello applicativo dello stack protocollare situato al di sopra di Internet, con il ruolo di definire una interfaccia o uno standard che renda comprensibile World Wide le informazioni trasmesse attraverso Internet.

Questo per evidenziare il fatto che Internet, come noi oggi lo conosciamo ed utilizziamo sia pressochè la stessa tecnologia prototipata ed utilizzata per scopi scientifico/militari nel 1969 dal DARPA denominata ARPANET (Advanced Research Projects Agency Network) dalla quale, nel 1983 naque Internet.

Il paradigma IoT si propone quindi come la prima, vera, evoluzione di Internet con la potenzialità di cambiare radicalmente il modo di vivere, imparare e lavorare. [11]

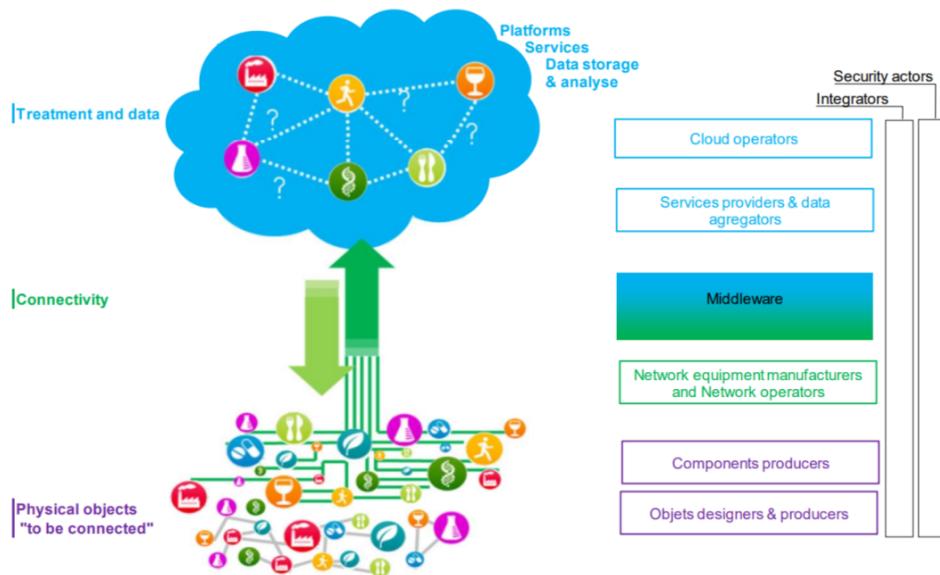


Figura 1.2: Gli oggetti collegati ad Internet possono essere progettati per una grande varietà di applicazioni [4]

1.1 I Numeri dell’IoT

Alla luce di quanto detto, si può quindi riassumere il significato del termine Internet of Things come la connessione di un numero crescente di dispositivi ed oggetti che comunicano tra di loro sfruttando la rete Internet. [26]

Ci si aspetta, ed i numeri attuali lo dimostrano, che il mondo dell’IoT abbia una crescita

esponenziale, connettendo miliardi di dispositivi in un tempo relativamente breve. Questo, ovviamente non può che avere un enorme impatto dal punto di vista tecnologico ed economico.

Dal punto di vista **tecnologico**, va affrontata la sfida di creare delle infrastrutture scalabili che possano supportare il numero crescente di dispositivi e la crescente mole di dati tra di essi trasmessa. Va inoltre notato come tali stime non tengano in conto anche dei

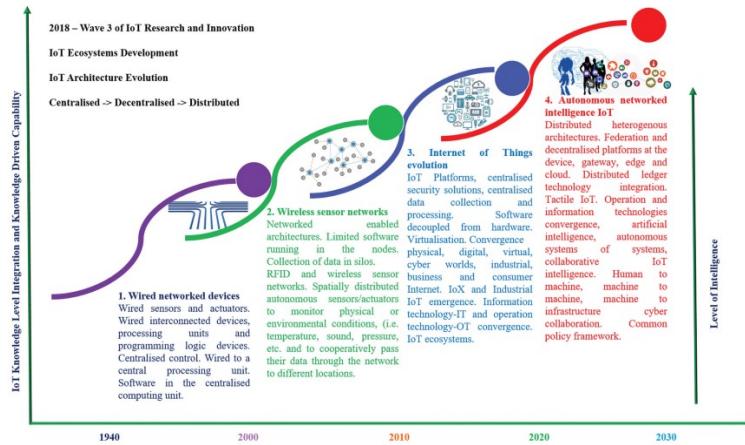


Figura 1.3: Number of IoT devices roadmap

vantaggi di una rapida ascesa di Internet ma che si basino solo sui dati noti al momento della stima.

La sfida tecnologica è dovuta al fatto che Internet è una tecnologia che risale allo scorso secolo che non fosse stata progettata per supportare un così crescente numero di dispositivi connessi alla rete.

Internet, infatti, fa largo uso del protocollo IP per la assegnazione di indirizzi pubblici univoci a ciascun device (o meglio connessione fisica alla rete) . Nella sua versione IPv4, ciascun indirizzo è formato da 32 bit, limitando così il numero massimo di indirizzi univoci a:

$$2^{32} = 4.294.967.296 \cong 4,3 \cdot 10^9 \quad (1.1)$$

Ma va tenuto presente che non tutti sono utilizzabili in quanto alcuni di questi siano riservati per specifici utilizzi. Tale protocollo quindi poco si adatta alla frenetica crescita del numero di dispositivo collegati ad Internet, motivo per il quale, già nel 1998 viene proposta una nuova versione di questo protocollo denominata IPv6 che sarà analizzata in dettaglio nella sezione 1.2. Il primo, evidente vantaggio di questa nuova versione, sta nel fatto che IPv6 riserva 128 bit per la codifica di ciascun indirizzo, portando così il numero massimo di indirizzi univoci assegnabili a [2]:

$$2^{128} \cong 3,4 \cdot 10^{38} \quad (1.2)$$

Questo inevitabile sviluppo del protocollo IP è completamente giustificato dalle stime sulla rapida crescita del numero di oggetti che vengono connessi alla rete Internet.

Dal punto di vista **economico**, una simile ascesa del numero di dispositivi connessi in

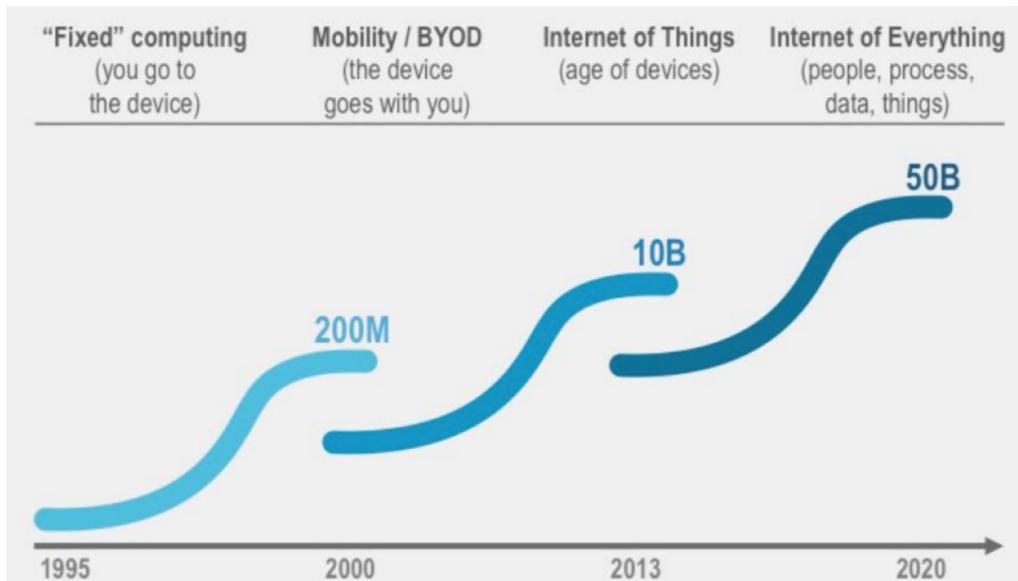


Figura 1.4: Previsioni della rapida crescita sul numero di oggetti connessi ad Internet [37]

rete, in un campo così ampio di applicazione, ha un tale impatto da giustificare gli investimenti su infrastrutture di telecomunicazione e sui plant industriali. Un simile andamento

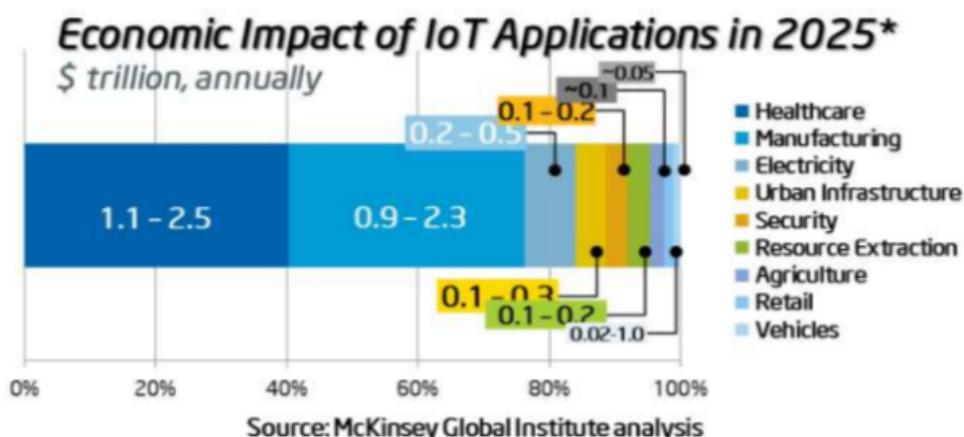


Figura 1.5: Impatto economico di applicazioni IoT nel 2025 fornito dal McKinsey Global Institute [14]

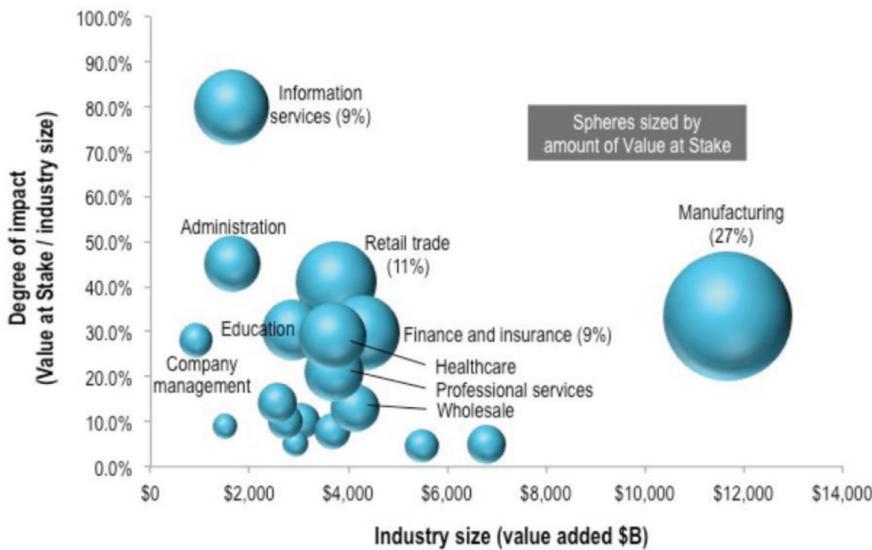


Figura 1.6: Campi di applicazione nei quali si stima il maggiore sviluppo reso possibile dall’IoT [37]

del mercato legato al mondo IoT è stato descritto anche in [43] dove si evidenzia come l’andamento del mercato globale, fortemente influenzato dal crescente numero di dispositivi connessi alla rete, possa attrarre una grossa percentuale di investitori e di valore economico aggiunto per le imprese.

S.N.	IoT Global
1.	Il mercato IoT crescerà da 15.4 miliardi di dispositivi nel 2015 a 30.7 miliardi di dispositivi nel 2020 e 75.4 miliardi nel 2025
2.	Nel periodo 2016-2021, le spese globali su prodotti e servizi legati al mondo IoT raggiungeranno i \$120-\$253 miliardi attraendo il 16% CAGR
3.	IoT crescerà da \$10 a \$15 trilioni di GDP globale nei prossimi 20 anni
4.	Nel 2020 la guida autonoma e dispositivi IoT cresceranno globalmente

Tabella 1.1: Previsione dell’andamento globale del mercato di prodotti e servizi legati al mondo IoT [43]

1.2 IoT Enabling Technologies

L’idea di una rete Internet che consentisse la comunicazione a livello globale tra persone o tra persone e cose o tra cose era già da molto tempo una visione condivisa di quello che sarebbe potuto essere lo sviluppo di Internet.

Tale visione è oggi possibile grazie alla diffusione capillare in qualsiasi strumento di piccoli, economici e potenti dispositivi di calcolo.

Tuttavia, affinchè una simile rivoluzione tecnologia si paventasse, era necessario superare i tradizionali protocolli di comunicazione tipici della rete Internet (IPv4, HTTP) e creare conseguentemente delle nuove tecnologie per la comunicazione che fossero ottimizzate per le nuove esigenze (IPv6, MQTT, CoAP, etc.). Va infatti ricordato ancora una volta che la vision dell’IoT sia quella di abilitare una comunicazione tra qualsiasi tipo di dispositivo che abbia al suo interno una unità di calcolo. Questo significa stabilire una comunicazione tra dispositivi eterogenei, dispositivi che assolvono diversi compiti ed adempiono a diverse necessità e che sono molto spesso anche dotati di una diversa e limitata potenza di elaborazione.

Di qui la necessità di nuovi protocolli e tecnologie di comunicazione che non solo garantissero la trasmissione sicura di una grossa mole di dati tra un grande numero di dispositivi, ma che fossero anche dei protocolli leggeri dal punto di vista computazionale ed ovviamente economici, così da poter essere integrati in qualsiasi dispositivo, da quelli destinati all’industria a quelli destinati al mercato consumer.

Data la complessità della sfida tecnologica da fronteggiare e considerata l’eterogeneità del

Tipologia di Comunicazione	Descrizione
M2M	Machine to Machine communication. Una macchina comunica con un’altra macchina per accumulare informazioni e scambiare dati.
H2H	Human to Human communication. Comunicazione tra uomini mediante gesti o mediante il linguaggio
D2D	Device to Device communication. Comunicazione tra due dispositivi che evitano infrastrutture di routing della comunicazione intermedie. Ciascun dispositivo comunica direttamente con un altro dispositivo che sia da esso direttamente raggiungibile senza intermediari

Tabella 1.2: Le metodologie di comunicazione [35]

problema e delle necessità nei diversi campi applicativi, sono stati sviluppate e proposte diverse tipologie di tecnologie e protocolli di comunicazione.

1.2.1 Il Protocollo IPv6

Come già anticipato nella sezione 1.1, il primo enabler necessario per l’avvento dell’Internet of Things è proprio quello che garantisca che ogni dispositivo, oggetto, sensore o persona possa connettersi alla rete Internet e scambiare dati con altri dispositivi. Per far sì che questo sia possibile, è quindi necessario dotare ciascun dispositivo di un indirizzo che identifichi univocamente quell’oggetto nella rete Internet. Questo compito, attualmente, è affidato perlopiù al protocollo IPv4 che identifica un elemento in rete attraverso un indirizzo IP pubblico univoco composto da 32 bit.

Visto il limitato numero di indirizzi IP che il protocollo IPv4 è in grado di fornire (alcuni dei quali non utilizzabili), già nel passato, si sono dovuti apportare alcuni accorgimenti e modifiche al protocollo. Già negli anni ottanta, infatti, si è giunti ad una saturazione di IPv4 che non era più in grado di adempiere al suo compito di fornire un indirizzo univoco, vista la crescita esponenziale avuta dalle reti LAN. A tal scopo, sono state escogitate altre tecniche (come il mascheramento) per continuare ad utilizzare il vecchio protocollo IPv4 nonostante la sua limitata capacità.

Sebbene queste tecniche ci abbiano consentito e ci stanno consentendo di continuare ad utilizzare il protocollo IPv4, nonostante la capillare espansione della rete Internet, quando si parla di Internet of Things o addirittura di Internet of Everything, questi stratagemmi non sono più sufficienti. Dalle stime sulla crescita esponenziale del numero di oggetti connessi ad Internet, si intuisce subito che sia necessaria una evoluzione del protocollo IPv4 già proposta nel 1998, ovvero il protocollo IPv6. Conseguentemente ad una modifica

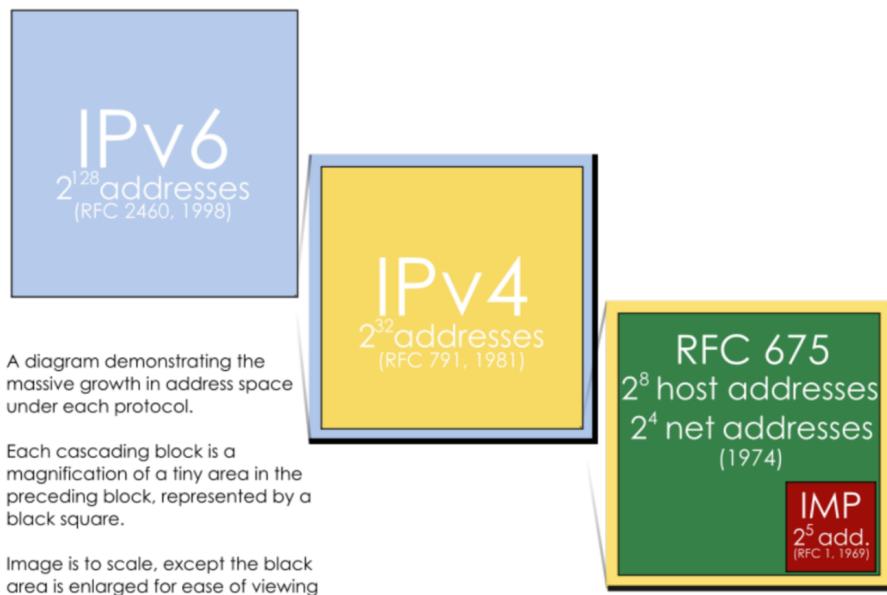


Figura 1.7: Evoluzione dello spazio destinato all’indirizzamento [13]

alla dimensione dello spazio di indirizzamento, è avvenuta anche una modifica per quanto riguarda l’header di un pacchetto trasmesso attraverso Internet in formato IPv6.

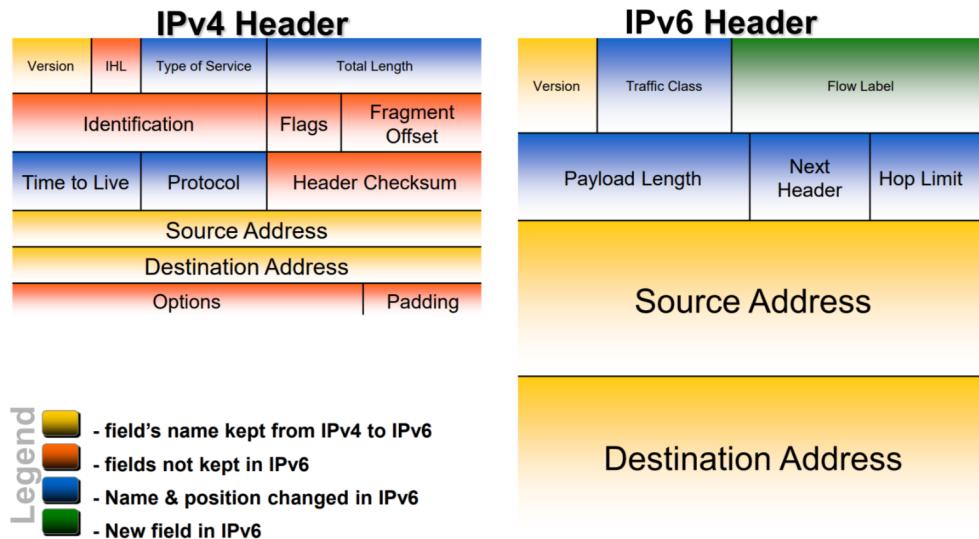


Figura 1.8: Confronto tra l’header nei protocolli IPv4 ed IPv6 [13]

Tra le principali caratteristiche del protocollo IPv6 rispetto all’IPv4 ci sono: [13] [2]

- Spazio di indirizzamento più grande (128 bit)
- Nuovo formato dell’header che minimizza l’overhead
- Infrastruttura di routing e schema di indirizzamento progettati appositamente per essere efficienti e gerarchici
- Supporto built-in per la sicurezza (IPSec) [12]
- Supporto per la QoS garantita da nuovi campi nell’header

I vantaggi di un protocollo (IPv6) progettato appositamente per soddisfare le nuove richieste della rete Internet sono subito evidenti:

- ✓ Maggiore disponibilità di indirizzi IP univoci
- ✓ Numero inferiore di campi all’interno dell’header
- ✓ Minore capacità computazione richiesta per la lettura dell’header

1.2.2 Molte applicazioni, diversi protocolli

Così come il protocollo IP diviene obsoleto con l’avvento dell’era dell’Internet of Things, rendendo necessario un upgrade a nuovi standard (IPv6), anche le infrastrutture di rete ed i protocolli di più basso livello lo diventano. Al fine di rendere possibile la connessione persistente ed onnipresente richiesta da molte delle applicazioni IoT, è necessario che siano aggiunte molte più features e funzionalità alle attuali tecnologie a banda larga. Le evoluzioni del 4G e le emergenti reti 5G saranno pertanto caratterizzate dall’interoperabilità e dall’integrazione tra più reti di accesso radio. [10]

Tuttavia, considerato il grande campo di applicazione ed il grande numero di dispositivi legati al mondo dell’IoT, sarebbe riduttivo limitare i protocolli di comunicazione di basso livello alle sole evoluzioni del 4G e del 5G. Una grande varietà di tecnologie di comunicazione, infatti, ha preso piede, rispecchiando così la grande varietà di domini applicativi, dispositivi e requirements legati all’IoT. Tra le più rilevanti nel campo della comunicazione domestica o locale emergono Bluetooth Low Energy [16] and Zigbee [25]. Mentre altri protocolli come WiFi, LowPower Wide Area Networks (LPWA) [30] e le comunicazioni cellulari come 3GPP , 4G o 5G [10] hanno uno scope più ampio, abilitando la comunicazione tra dispositivi anche molto distanti tra loro.

Tutti questi protocolli di comunicazione e le infrastrutture che li supportano, condividono alcune caratteristiche che sono comunemente necessarie a tutti gli oggetti connessi in rete:

- Scalabilità
- Bassa capacità computazionale richiesta
- Economicità
- Bassi consumi energetici
- Supporto dell’ecosistema IP

1.2.3 L’Application Layer per i dispositivi IoT

A valle di tutte le considerazioni effettuate sugli enablers necessari al mondo dell’Internet of Things per instaurare una comunicazione a basso livello tra oggetti, è bene anche considerare che, al pari dei protocolli e tecnologie di basso livello, anche i protocolli di comunicazione di alto livello (application layer) diventano obsoleti ed inadatti all’IoT.

Uno dei principali motivi per i quali Internet come lo conosciamo ed utilizziamo oggi ha avuto un capillare sviluppo a livello globale ed ha consentito la massiccia nascita di applicazioni e servizi ad esso legati, è certamente dovuto al World Wide Web. Come già anticipato, infatti, la rete Internet nata per scopi militari/scientifici, ha come unico obiettivo quello di trasmettere una informazione, da un punto ad un altro, anche molto distanti. Tuttavia, che cosa venga trasmesso attraverso Internet, cioè quali dati e cosa essi significano e come sono codificati, è fuori dallo scope di Internet. In riferimento alla Figura 1.9, questi compiti sono svolti da livelli superiori allo stack protocollare TCP-IP che si occupano di codificare determinate informazioni o dati, passarli ad i livelli inferiori (Internet)

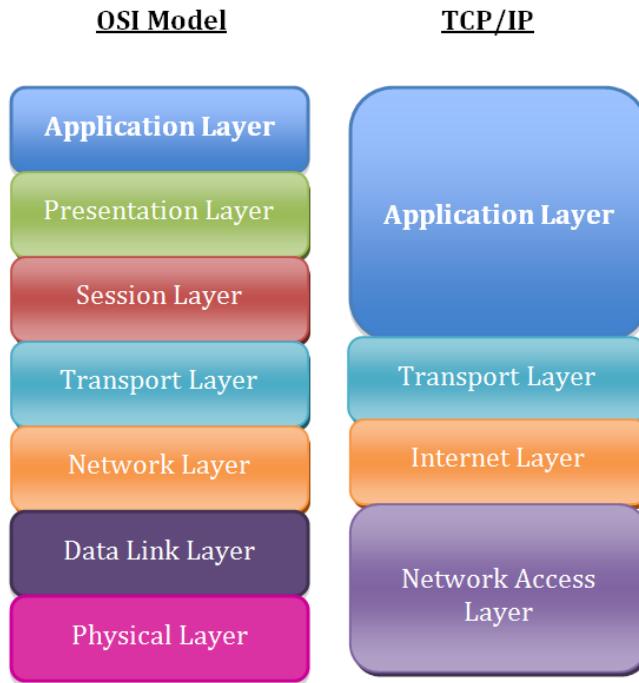


Figura 1.9: Modello TCP-IP (destra) in confronto con il modello ISO-OSI (sinistra) [7]

e poi trasmetterli. Alla base del World Wide Web e quindi del comune utilizzo di Internet è presente il protocollo HTTP o HTTPS. Questo protocollo appartiene all’ Application Layer di Figura 1.9 ed è utilizzato come principale sistema per la trasmissione di informazioni sul Web attraverso una architettura Client-Server che, a livello più basso (Transport Layer) utilizza il protocollo TCP. Questo protocollo, unitamente ad HTML, URIs e REST, hanno portato all’enorme successo del World Wide Web. [13] Per quanto il protocollo HTTP sia oggi diffuso ed utilizzato e si potrebbe esportare anche nei dispositivi legati al mondo dell’Internet of Things, vanno considerati alcuni svantaggi e limitazioni al suo utilizzo.

- × HTTP utilizza a livello di trasporto il protocollo TCP il quale risulterebbe troppo pesante
- × HTTP utilizza i protocolli SSL/TLS per garantire elevati standard di sicurezza i quali risulterebbero troppo pesanti

A livello applicativo vengono pertanto progettati dei nuovi protocolli che risultino più leggeri nell’elaborazione in dispositivi dalle capacità limitate, ma che al contempo ereditino i vantaggi, le funzionalità e la compatibilità con lo standard HTTP. Tra i più famosi CoAP, AMQP e MQTT.

In riferimento al prototipo sviluppato in questo lavoro di tesi, a valle delle scelte progettuali

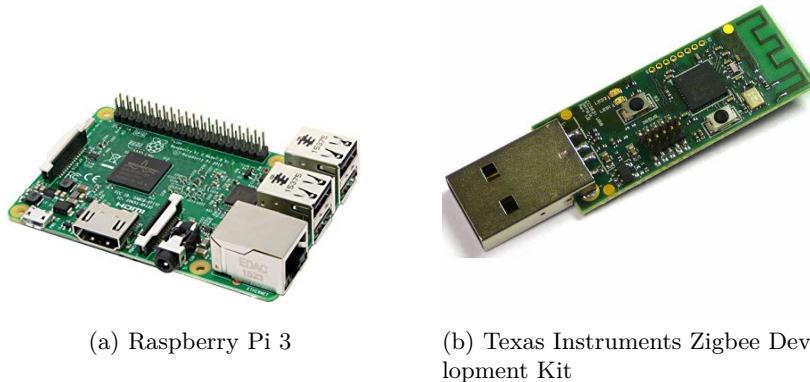


Figura 1.10: Dispositivi con limitate capacità computazionali, RAM e ROM

mostrate nel Capitolo 3, si è ritenuto opportuno utilizzare il protocollo MQTT per il livello applicativo e pertanto si intende adesso analizzarlo più nel dettaglio.

Il Protocollo MQTT

Per la condivisione di dati attraverso Internet, il protocollo MQTT sta diventando uno dei più popolari. Questo anche grazie al fatto che sia un protocollo leggero e che quindi bene si adatti ai sistemi ed alle necessità del mondo dell’Internet of Things, ovvero condividere una grossa mole di dati in real-time. Il protocollo MQTT, realizzato nel 1999 da Dr. Andy Stanford-Clark e Arlen Nipper, nasce come un protocollo open source di messaggistica con architettura publish-subscribe. La sua progettazione è stata proprio pensata per funzionare in dispositivi limitati, reti inaffidabili o ad elevata latenza e che fosse anche facile da implementare. [17] [13] Gli attori principali che compongono questo protocollo sono pertanto:

- MQTT Broker
- Publisher
- Subscriber

Per lo scambio di dati attraverso il protocollo MQTT, è predisposto un controllore centrale che viene utilizzato per distribuire i messaggi.

Questo è detto **MQTT Broker** ed ha il compito di inoltrare, filtrare ed assegnare le priorità alle publish request che il broker riceve dai publisher dirette ad i subscriber.

I **Publisher** sono coloro che si occupano di generare i dati. Possono infatti essere sensori, generatori di dati o sistemi integrati, i quali, per comunicare attraverso MQTT, devono specificare due elementi principali: messaggio da inviare e topic. Solo in questo modo, attraverso il topic di un messaggio, il Broker potrà ricevere correttamente i messaggi dai

publisher e decidere quale Subscriber debba ricevere il messaggio.

I **Subscriber** infine, non sono altro che dispositivi o oggetti o altri Publisher, che si sottoscrivono ad uno o più topic. Ovvero che richiedono al Broker di ricevere tutti i messaggi ad esso inviatogli che siano appartenenti ad un certo topic. I Subscribers possono quindi ricevere i soli messaggi che abbiano lo stesso topic (o gli stessi topic) di quello ai quali si sono sottoscritti. Dalla Figura 1.11 si evince come, se il Publisher invii un messaggio al

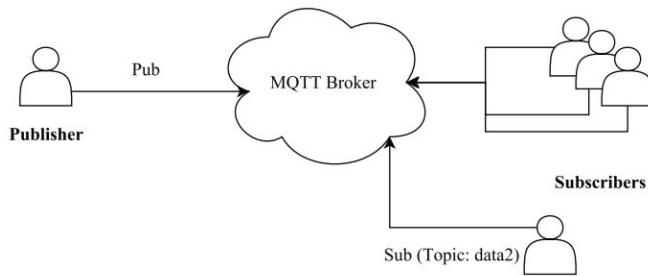


Figura 1.11: Comunicazione attraverso il protocollo MQTT [17]

Broker caratterizzato dal topic : *data2*, allora questo sarà ricevuto dal solo Subscriber in basso, quello cioè in 'ascolto' sul topic *data2*.

Il protocollo MQTT, infine, grazie alla sua leggerezza rispetto ad HTTP, consuma meno potenza per ore di operazione rispetto ad HTTP ed inoltre, in un’intervallo di un’ora, i messaggi scambiati attraverso il protocollo MQTT sono dieci volte superiori rispetto a quelli che sarebbero scambiati nello stesso intervallo di tempo con HTTP.[22] [9]

1.3 Applicazioni IoT

Quello di cui si è parlato nelle precedenti sezioni non va considerato come una pura previsione o aveniristica visione del mondo, bensì si tratta di un movimento attualmente in corso, di una rivoluzione di Internet che intacca inevitabilmente il mercato consumer e quello industriale. Di seguito una panoramica di quelli che sono attualmente i campi applicativi nei quali l’IoT sta avendo maggiore sviluppo grazie anche alla nascita di soluzioni e servizi industriali e non.

1.3.1 Home

Il termine Smart Home, indica l’integrazione di un numero anche abbastanza elevato di dispositivi all’interno dell’ecosistema casa. Dispositivi che originariamente sono disconnessi ed indipendenti tra loro, iniziano a comunicare al fine di offrire esperienze e servizi all’utente.

In questo modo, dispositivi quali luci, termostati, sistemi di allarme possono essere attivati da remoto attraverso uno smartphone o attraverso un comando vocale o addirittura possono imparare le nostre abitudini, comunicare tra di loro, per decidere quali comportamenti intraprendere senza alcun intervento umano.



Figura 1.12: Dispositivi comunicano tra di loro nell’ecosistema casa



(a) Google Home



(b) Google Nest

Figura 1.13: Esempi di soluzioni per l’ecosistema smart home

1.3.2 Healthcare

Uno degli ambiti nei quali il mondo dell’IoT potrebbe avere un impatto straordinario sulle persone è proprio quello dell’healthcare, rivoluzionando di fatto l’intera concezione legate a questo mondo. Il fine delle applicazioni IoT in ambito medico è quello di migliorare la vita delle persone, migliorando e monitorando il loro stato di salute attraverso l’utilizzo di dispositivi indossabili ed interconnessi. In questo modo, tali dispositivi avranno il compito di raccogliere dati che saranno fondamentali per una corretta creazione di un profilo dello stato di salute individuale e la conseguente creazione di piani personalizzati di cura.

1.3.3 Agriculture

Data la continua crescita della popolazione mondiale, di pari passo incrementa anche la domanda di cibo e beni primari. Per fronteggiare a queste necessità, attraverso l’IoT è possibile dotare gli agricoltori di tecniche avanzate per incrementare la produzione e migliorare la qualità. Gli agricoltori sono così in grado di raccogliere utili informazioni da



Figura 1.14: Soluzione per il monitoraggio dell’asma (sinistra) e per il monitoraggio del glucosio per pazienti affetti da diabete (destra)

sensori disposti nei campi che possano aiutarli nelle scalte da adottare nella coltivazione e controllare l’utilizzo di acqua e fertilizzanti.



Figura 1.15: Bosch Deepfield Connect Project

1.3.4 Cities

Il campo delle Smart Cities è talmente vasto ed offre talmente tanti spunti che sarebbe riduttivo ricondurlo a pochi esempi applicativi. L’idea alla base dell’utilizzo di dispositivi IoT all’interno di una città è quello di automatizzare ed ottimizzare servizi quali i trasporti pubblici, inquinamento, sicurezza e regolamentare il traffico. In questo modo, attraverso la distribuzione di una rete di sensori interconnessi tra loro, i cittadini possono essere in grado di trovare parcheggio, rilevare in anticipo ed eventualmente evitare aree trafficate, rilevare infrazioni e pericoli.

1.3.5 Energy

Il termine Smart Grid è diventato presto popolare in tutto il mondo, questo perchè il suo scopo è ambizioso ed è quello di raccogliere dati relativi alla richiesta e produzione energetica in maniera automatica, analizzare i comportamenti ed il consumo energetico per migliorare l’efficienza con la quale i fornitori distribuiscono l’energia e la rendono disponibile ai consumatori. In questo modo è possibile ridurre i costi e l’impatto ambientale legati alla produzione dell’energia.

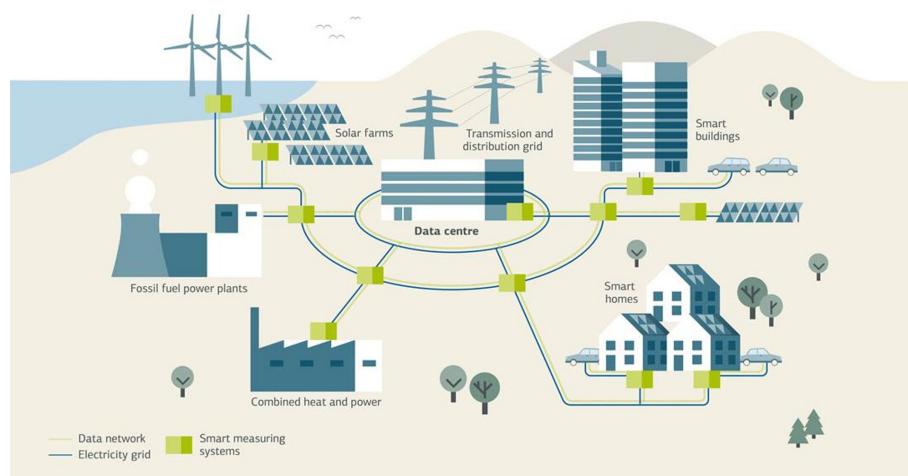


Figura 1.16: Il concept delle smart grid

1.3.6 Retail

Il nostro modo di acquistare, scegliere e desiderare nuovi prodotti è stato già completamente rivoluzionato dalla possibilità di acquistare un capo di abbigliamento oppure l’ultima novità tecnologia oppure del pane fresco stando comodamente seduti in spiaggia o in ufficio e riceverlo entro poche ore. Sono inoltre già presenti dispositivi IoT che effettuano ordini al nostro posto quando un certo prodotto si esaurisce o sta per farlo, in modo tale da non lasciarci mai senza. In questo modo, frigoriferi o dispense intelligenti acquistano per noi quello che ci serve. A questa rivoluzione del modo di comprare online che è già ampiamente consolidata, si sta avvicinando una nuova rivoluzione del modo di comprare anche dai negozi fisici. I negozi fisici infatti si trasformano e non hanno più casse dove pagare o commessi a rifornire gli scaffali, il tutto sostituito da una grande quantità di sensori, sistemi di visione ed in generale dispositivi IoT che sono in grado di capire cosa inseriamo nel nostro carrello, in quale quantità ed infine addebitare il tutto sul nostro smartphone senza alcuna cassa ne coda. In questa rivoluzione del modo di comprare e di vendere, gli smartphone sono il mezzo grazie al quale venditori e consumatori restano in contatto anche una volta fuori dal negozio. Inoltre essi, unitamente ad altri sensori e dispositivi



Figura 1.17: Amazon Go store nella città di Seattle

interconnessi, rappresentano un modo per seguire i movimenti dei clienti all’interno dei negozi, in modo da poter addattare il layout dello store disponendo alcuni prodotti in zone a più elevato traffico.

1.3.7 Automotive

Nel campo della mobilità, l’IoT ha già offerto varie soluzioni finalizzate ad offrire al passeggero un maggiore confort, intrattenimento e sicurezza. Un esempio ne sono i consolidati sistemi di aiuto alla guida quali Cruise Control, Frenata Assistita, Rilevamento dei Pedoni, Rilevamento di Corsia etc. Il prossimo step nella integrazione del mondo IoT con



Figura 1.18: The IoT Connected Car

quello Automotive è quello di stabilire una comunicazione tra veicoli, finalizzata ad offrire nuovi servizi. Tale comunicazione potrà poi essere anche uno dei maggiori enabler per la guida autonoma di veicoli i quali, non soltanto saranno in grado di ‘vedersi’ ma anche di ‘parlarsi’.

Questo è proprio il fulcro di questo lavoro di tesi che si concentrerà quindi sulla progettazione e prototipazione di un servizio IoT che, abilitando una comunicazione tra diversi veicoli, possa offrire ad ogni conducente un servizio in real time sul monitoraggio del traffico e, più in generale, di segnalazione di tutti gli eventi che potrebbero occorrere e potrebbero mettere a rischio la vita delle persone o anche soltanto che potrebbero consentire di raggiungere in un minor tempo la propria destinazione. Sarà infatti progettato e prototipato nei capitoli seguenti, un sistema di condivisione dei dati stradali attraverso una piattaforma di Crowdsourcing, termine che sarà spiegato nel dettaglio nel Capitolo 2. Tali dati stradali, provenienti da diversi utenti, in diverse aree geografiche, potranno essere poi obiettivo di una analisi dei dati con tecniche di Machine Learning per la scoperta di pattern, consigli sulla navigazione, sulla manutenzione ed ampiamento delle strade. Tale analisi di dati, sebbene nel Capitolo 2 e Capitolo 3 venga sviluppata una interfaccia che renda possibile l’immediato utilizzo di tecniche di Machine Learning sui dati raccolti, sarà fuori dallo scopo di questo lavoro di tesi.

Capitolo 2

Sistemi di Crowdsourcing

Con il termine Crowdsourcing ci si riferisce alla *possibilità di utilizzare i contributi indipendenti di una "folla" per uno scopo, senza che questi siano organizzati a priori in flussi di lavoro* [40].

L'evoluzione dell'umanità è stata sempre legata al linguaggio o più in generale alla comunicazione. Attraverso la comunicazione, sia che essa avvenga attraverso i gesti o il linguaggio o Internet, è stato possibile condividere la conoscenza acquisita dagli esseri umani. Ogni uomo ha potuto rendere disponibile la propria conoscenza e la propria esperienza dappri-ma ai suoi vicini ed ora, attraverso Internet, a tutto il mondo.

L'Internet of Things non è altro che l'ennesimo strumento per lo scambio di dati e quindi per la condivisione della conoscenza acquisita. Solo che in questo caso, la conoscenza non viene più acquisita e condivisa tra i soli uomini ma anche tra uomini e macchine e tra macchine e macchine.

Di pari passo con la crescita del numero di dispositivi che raccolgono dati, sono anche cresciute le scienze ed i servizi che utilizzano ed analizzano questi dati per trarne da questi delle informazioni. Data la mole dei dati sui quali si opera, questi dati non potrebbero essere analizzati se non che da algoritmi. D'altro canto, questi algoritmi o tecniche di Machine Learning sui dati, fanno un uso massiccio di dati ed inoltre, diventano più accurati all'aumentare del numero di dati che ricevono in pasto. Si crea quindi una doppia relazione tra i sensori e dispositivi IoT connessi alla rete che raccolgono dati di ogni sorta e strumenti di analisi dei dati che, analizzando questi dati, effettuano delle previsioni ed offrono dei servizi i quali, diventano sempre più completi ed accurati quanto più numerosi ed accurati siano i dati a disposizione. Per soddisfare questa fame di dati e quella che verrà in futuro, per quanto la raccolta dati possa essere automatizzata e distribuita in un numero infinitamente alto di dispositivi, sono necessarie delle risorse ingenti per la distribuzione, manutenzione e creazione di infrastrutture di comunicazione per fare in modo che questi dati siano disponibili ed utilizzabili. Di qui l'idea del Crowdsourcing di esternalizzare un lavoro o parte di un lavoro ad un gruppo di persone o nel nostro caso, ad un gruppo di cose e persone. L'obiettivo è quello di sfruttare una più grande forza lavoro per il raggiungimento di un certo scopo, ovvero quello della raccolta massiccia di dati. [29] Ricerche dimostrano che ricorrere all'utilizzo del Crowdsourcing, anche al di fuori del mondo IoT,

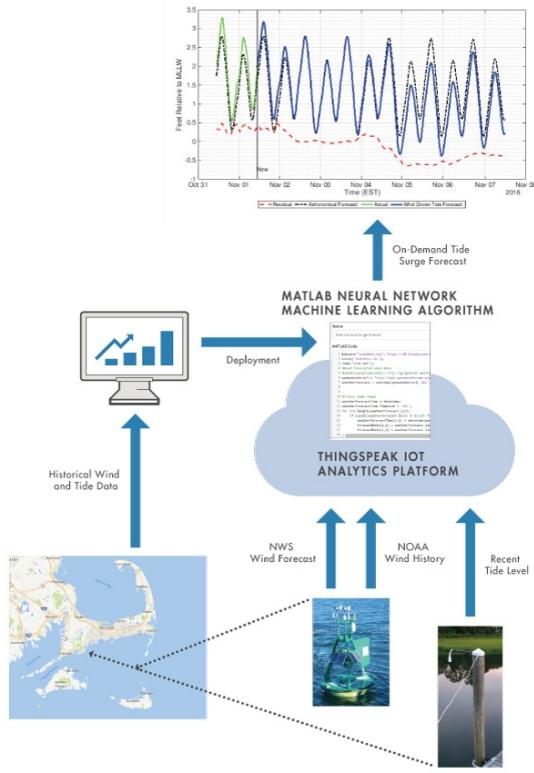


Figura 2.1: Esempio dello sviluppo di un sistema IoT di analisi utilizzando MATLAB, Machine Learning e ThingSpeak [19]

ed in particolare degli approcci di Crowdsourcing legati ai cittadini, possano risultare una fonte affidabile, scalabile e sostenibile per la raccolta di grosse moli di dati. [29] Per chiarire definitivamente il concetto ed i vantaggi del Crowdsourcing, vengono mostrate nella sezione 2.1 alcune tra le applicazioni più comuni e famose che sfruttano appunto tutta la potenza del Crowdsourcing.

2.1 Applicazioni di Crowdsourcing

Attualmente, sono molteplici le applicazioni che sfuttano la potenza messa a disposizione da una '*forza lavoro*' praticamente gratuita ed inesauribile. Queste applicazioni sono talmente comuni ed ottimizzate da non farci nemmeno sospettare di essere noi, con i nostri utilizzi, ad essere la '*forza lavoro*' che produce miliardi di dati e che, molto spesso, quelli stessi dati, dopo opportune analisi, li consuma anche. Negli esempi contenuti nei paragrafi successivi, sarà subito lampante come queste applicazioni di Crowdsourcing si differenzino principalmente in un aspetto: **chi genera i dati**. Sono infatti proposte applicazioni nelle

quali a generare i dati sono gli umani, condividendo ciò che essi vedono o conoscono. Oppure, ci sono applicazioni nelle quali i dati sono generati dalle macchine, a volte in maniera completamente trasparente all'uomo che spesso ne è addirittura inconsapevole.

2.1.1 Google Maps

Google Maps ® è il celebre servizio offerto da Google che ci consente di orientarci in nuove città, scoprire nuovi posti che potrebbero interessarci, aiutarci a trovare la strada migliore per tornare a casa e guidarci nella navigazione. Nella fattispecie, il servizio di navigazione offerto da Google Maps ® , si è rivelato ben presto nettamente superiore alle soluzioni di navigazione precedentemente in uso. Questo tutto grazie all'IoT e nella fattispecie ad un sistema di Crowdsourcing tra i più potenti ed avanzati. La feature aggiuntiva di Google Maps ® e di altri servizi simili (Apple Maps ®) sta nel fatto che sono in grado, sulla base della condizione del traffico stimata in real time, di suggerirci la strada migliore per arrivare a casa, in qualsiasi parte del mondo essa sia, nel minor tempo possibile. Per realizzare una simile infrastruttura, sarebbero necessarie centinaia di sensori, sistemi di visione ed antenne per la comunicazione in real time dello stato sul traffico per ogni km di strada, in tutto il mondo. Senza azzardare calcoli, la cifra per la progettazione, sviluppo e manutenzione di un simile impianto sarebbe spropositata persino per Google. Pertanto Google ha deciso di sfruttare il dispositivo IoT più comune, più utilizzato e più diffuso: lo smartphone, lo stesso smartphone che utilizza i servizi offerti da Google Maps ® è utilizzato da Google per ottenere delle stime in real time sul traffico. Di fatto, ogni giorno, miliardi di dispositivi si muovono su tutte le strade del mondo. Questi sono, a tutti gli effetti, dei dispositivi IoT in quanto dotati di sensori, comunicazione attraverso Internet ed una sovrabbondante capacità di calcolo.

Non c'è da stupirsi, infatti, se una rete così grande di sensori possa fornire informazioni sufficienti a Google per consigliarci in maniera estremamente accurata di seguire un percorso piuttosto che un altro per tornare a casa evitando così rallentamenti o incidenti. La

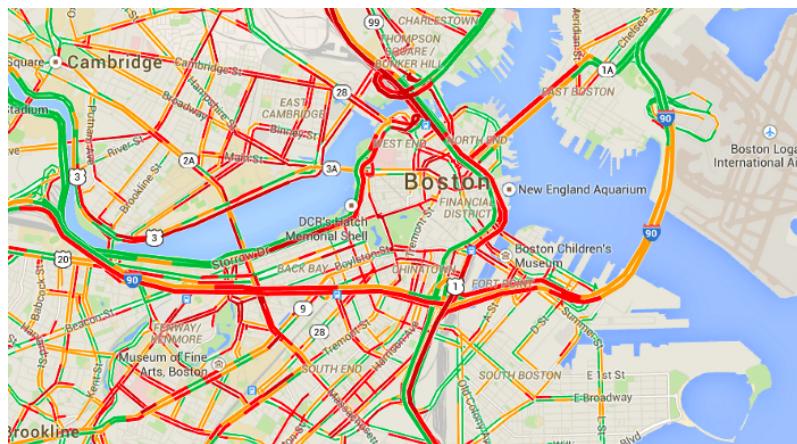


Figura 2.2: Indicazione sul livello di traffico stradale nella città di Boston

capacità di combinare la velocità di un dispositivo con la velocità di altri dispositivi sulle strade, utilizzando migliaia di smartphone in movimento in una area geografica ad un dato periodo di tempo, ha consentito ai servizi offerti da Google di fornire una panoramica affidabile delle condizioni di traffico in real time. Tutto questo ottenuto senza sborsare l'enorme cifra necessaria alla distribuzione di una quantità spropositata di sensori sulla rete stradale di tutto il mondo, ma utilizzando una delle più grandi piattaforme di Crowdsourcing completamente invisibile a coloro che la utilizzano e completamente invisibile a coloro che contribuiscono alla creazione di dati. [28]

2.1.2 Linux e Wikipedia

Un esempio di Crowdsourcing completamente differente da quello mostrato in precedenza, è quello che ha consentito la realizzazione di piattaforme come Wikipedia e del sistema operativo Linux. In questo caso, infatti, l'utente non solo è consapevole di contribuire alla realizzazione di un progetto distribuito, ma anche deve mettere a disposizione attivamente le proprie competenze per contribuire al progetto.

In Wikipedia, ogni utente registrato, è spronato a dare il proprio contributo per l'accrescimento della più grande enciclopedia pubblica e gratuita. La '*forza lavoro*' in questo caso è consapevole del ruolo svolto, della sua importanza e dell'impatto che esso ha sulla community di utilizzatori.

Allo stesso modo, il sistema operativo Linux, viene sviluppato e manutenuto da una community di sviluppatori, il cui contributo è quello di mettere a disposizione le proprie competenze informatiche per sviluppare e quindi rilasciare il Sistema Operativo in forma Open Source.

In entrambi i casi, il sistema Crowdsourcing è stato indispensabile alla creazione del prodotto finito. Se pensiamo a Wikipedia, in essa sono contenute più di 320 milioni di pagine, tradotte in 280 lingue, uno sforzo immenso se questo lavoro fosse toccato ad una sola persona o ad una sola azienda. Invece, sfruttando il Crowdsourcing, si raccoglie tutta la conoscenza acquisita da qualsiasi uomo in qualsiasi parte del mondo, in qualsiasi istante di tempo. [41]

I numeri di Linux, sono altrettanto impressionanti, si stima infatti che al 2018, Linux fosse composto da oltre 25 milioni di righe di codice, scritte da più di 19 mila diversi contributori. (fonte: <https://linux.slashdot.org/>) Ancora una volta, questi numeri sono possibili grazie ad una community che collabora per il raggiungimento di un certo fine che, in questo caso, piuttosto che finalizzato alla raccolta dei dati, è finalizzato alla condivisione della conoscenza e delle competenze.

2.2 Crowdsourcing ed Internet of Things

Ora che sono chiare le potenzialità offerte dal Crowdsourcing, è necessario affrontare un altro tema che consentirà di avere una più chiara visione sul legame Crowdsourcing ed IoT dal quale sarà possibile trarre utili informazioni necessarie alla progettazione fatta nel Capitolo 3.

Sebbene evidenti i vantaggi delle piattaforme di Crowdsourcing negli esempi mostrati nella

sezione 2.1, vanno tuttavia effettuate alcune considerazioni ed accorgimenti da intraprendere quando si utilizzano simili strumenti.

Il mondo dell’Internet of Things, si adatta perfettamente a quella che è l’idea alla base del Crowdsourcing : *Esternalizzazione di una parte del lavoro ad un gruppo di persone, le quali, con contributi indipendenti e non organizzati lavorano per uno scopo comune.* Se ci si sforza ad estendere questa definizione non solo alle persone ma anche alle cose, è possibile vedere come questa nuova definizione del Crowdsourcing applicata agli oggetti, bene si adatti con il paradigma dell’Internet of Things. Se grazie agli strumenti, sensori, protocolli e sistemi integrati offerti dal paradigma IoT è possibile garantire che qualsiasi oggetto possa raccogliere dati e condividerli attraverso Internet, il Crowdsourcing ci suggerisce che tutti questi dati, raccolti come contributi indipendenti da un grande numero di dispositivi, possano essere raggruppati ed utilizzati insieme per il raggiungimento di un più grande scopo. [29] [42]

Nella sezione 2.1 è stata volutamente omessa una informazione. Sebbene sia vero che senza il Crowdsourcing sarebbe stato impossibile per Linux raggiungere le 25 milioni di righe di codice e per Wikipedia sarebbe stato impossibile raggiungere le 320 milioni di pagine, è anche vero che il contributo dei 19 mila sviluppatori di Linux e dei 4 milioni di utenti registrati in Wikipedia vada gestito e regolamentato. Si rende cioè necessario trovare una sorta di Protocollo o di Interfaccia comune che tutti i contributori, sebbene indipendenti, debbano seguire ed adottare per contribuire al progetto.

In Linux questa interfaccia risiede nel software Git per il source control. In questo modo ogni sviluppatore che contribuisce al progetto Linux può clonare¹ o fare un branch² del progetto esistente, aggiungere le proprie modifiche ed effettuare nuovamente un commit³ sul server così che tutti possano prendere visione delle modifiche apportate. Soltanto dopo aver verificato che le modifiche di un certo utente siano conformi al task richiesto, non introducano problemi o bug e non comportino un rischio per la sicurezza, ne viene effettuato il merge⁴ all’interno della directory principale.

Analogamente, in Wikipedia, è necessario verificare la veridicità e la correttezza dei contributi di ciascun utente. A tal proposito, Wikipedia si affida ad un’altra forma di Crowdsourcing, questa volta realizzata non solo da utenti registrati ma anche da coloro che usufruiscono del servizio che possono segnalare argomenti che ritengono sbagliati o non completi.

Questa necessità di realizzare una Interfaccia o un Protocollo comune a tutti i contributori di una piattaforma di Crowdsourcing si rende necessaria non solo nelle applicazioni dove i contributori sono umani, ma anche in quelle dove i contributori sono macchine.

Come evidenziato nel Capitolo 1, la forza dell’Internet of Things è quella di abilitare una comunicazione ed uno scambio dati tra dispositivi eterogenei. Con eterogenei si intende

¹Termine utilizzato per indicare l’ottenimento di una copia di un progetto globale in una propria cartella locale

²Termine che indica l’esecuzione di una diramazione di un certo progetto, in un certo stato

³Termine che indica la conferma dell’esecuzione di una modifica ad un dato progetto

⁴Termine che indica l’unione di due versioni dello stesso codice

una differente potenza di calcolo, dotazione sensoristica, funzionamento, costo e casa produttrice. Il risultato di questa eterogeneità è che non esiste un modo comune di identificare un particolare dispositivo all'interno di una piattaforma di Crowdsourcing, ovvero il tipo del dispositivo, il costruttore e quali dati sono prodotti non sono informazioni regolate da uno Standard comune. [29] Una difficoltà aggiuntiva se si realizza una piattaforma di Crowdsourcing con dispositivi IoT, sta nel fatto che sia necessario gestire e legittimare l'accesso e la produzione dei dati. Analogamente a quanto fatto da Wikipedia e Linux, è cioè necessario gestire i contributi dei singoli ed ulteriormente verificare che essi siano corretti e conformi. I problemi legati allo sviluppo di una piattaforma di Crowdsourcing legata a dispositivi IoT sono quindi:

1. **Interoperabilità Semantica** : L'IoT è una tecnologia '*industry driven*' nella quale ogni costruttore realizza la sua piattaforma IoT. In più, la maggior parte delle soluzioni IoT sono case-centric con la conseguente creazione di '*IoT Silos*' i quali, per comunicare tra loro e scambiarsi dati, hanno bisogno di una interoperabilità '*Inter-Silos*'. Si rende necessario che ogni Protocollo o Standard debba considerare la varietà dei dispositivi, il loro contesto di utilizzo ed i dati emessi. La sfida nel dominio IoT è dovuta alla presenza di una varietà di ontologie che trattano vari aspetti dei sensori e delle grandezze da misurare (diversa portata, granularità e generalità). Questo complica lo sviluppo di una ontologia formale o di un Protocollo di comunicazione comune a tutti i dispositivi.
2. **Condivisione dei Dati e Controllo degli Accessi** : Un'altra grande sfida per la ricerca nel campo dell'IoT è quella di garantire la privacy degli utenti e la protezione dei dati personali, specialmente in sistemi nei quali si effettua raccolta, gestione e condivisione dei dati. L'identificazione e la gestione di miliardi di dispositivi connessi tra loro ed il mantenimento di una comunicazione affidabile e sicura, rappresentano un punto critico da regolamentare. Questo richiede sia lo sviluppo di una policy che regolamenti la comunicazione ma anche richiede uno sviluppo tecnologico per la condivisione sicura ed affidabile con protocolli leggeri ed interoperabili.
3. **Elaborazione di Politiche Democratiche** : Alcuni domini specifici nei quali l'IoT si sta diffondendo come l'healthcare e le smart cities, sono campi di applicazione nei quali gli individui operano non soltanto come data providers, ma essi partecipano attivamente nella risoluzione di problemi, condivisione di soluzioni e formulazione di leggi e regolamenti. Si rende quindi necessario adattare il comportamento e i Protocolli o Standard di comunicazione alle particolari politiche democratiche ed ulteriormente al particolare campo di applicazione.

Capitolo 3

Progetto del Sistema

Nei capitoli precedenti è stata offerta una panoramica sugli strumenti che il paradigma IoT mette a disposizione e su come alcune compagnie utilizzino piattaforme di Crowdsourcing per fornire risorse e servizi altrimenti impensabili.

In questo capitolo, saranno trattate la progettazione e le scelte architettoniche che hanno portato alla prototipazione di un Software che sfrutti il paradigma IoT e tecniche di Crowdsourcing per offrire un servizio legato al mondo dell'Automotive. In questo capitolo, tuttavia, piuttosto che alla realizzazione di un prodotto finito, ci si focalizzerà sulle motivazioni che hanno portato ad alcune scelte progettuali piuttosto che ad altre. Tale fase di progettazione, essendo di carattere completamente generale, potrà essere utilizzata anche per lo sviluppo di altri sistemi in altri campi applicativi che facciano uso di tecniche di Crowdsourcing applicate a dispositivi IoT.

3.1 Idea Generale

Recentemente, notevole importanza viene posta ai cosiddetti Intelligent Transportation System (ITS) sia da parte delle amministrazioni pubbliche ed europee (2019/40/EU [8]) che nel mondo della ricerca. [6] In [8], infatti, viene definito un framework per le specifiche di sviluppo che rendano interoperabili le piattaforme ITS anche attraverso differenti organizzazioni. La necessità è quella di fornire informazioni sul traffico in real-time per il territorio Europeo.

Avendo a mente questa necessità, in questo lavoro di tesi si intende progettare e sviluppare un Software che, raccogliendo dati sulla navigazione da dispositivi eterogenei e, raggruppando questi dati in una piattaforma di Crowdsourcing, possa sottoporli ad analisi e ricavare così delle informazioni sul livello del traffico in una data area geografica. I dati raccolti possono essere di diversa natura, proprio perché diversa è la natura dei dispositivi che possono raccogliere ed inviare questi dati. Alcune informazioni possono riguardare:

- Dati legati alle condizioni del traffico
- Dati provenienti da vari sensori disposti sulla rete stradale (es. stazioni meteo)

- Dati provenienti da dispositivi che transitano temporaneamente sulle strade (Smart-phone, Tablet, Sistemi di Infotainment nelle automobili)
- Dati provenienti da videocamere a circuito chiuso (CCTV)

Dunque, proprio data l'eterogeneità di dispositivi che parteciperanno alla fase di raccolta dati, sarà necessario anche analizzare alcune tecniche e Protocolli che consentano ad una così eterogenea estrazione di dispositivi di poter raccogliere dati ed inviarli in un formato comune e quindi comprensibile. Infine, una volta creato e popolato in real time il dataset, questi dati saranno resi disponibili per delle query geo-referenziate, in modo tale che, una utenza privata o una organizzazione per il controllo del traffico, possa accedere ad i dati più recenti sul traffico di una certa area geografica.

Nella fattispecie, la fase di progettazione riguarderà:

- Raccolta dei Dati
- Formattazione dei Dati in una Interfaccia condivisa tra dispositivi eterogenei
- Geo-Referenziare i Dati
- Invio dei dati Geo-Referenziati ad un Middleware IoT
- Storing dei dati Geo-Referenziati in Database opportunamente progettati
- Gestione delle query Geo-Referenziate al Database

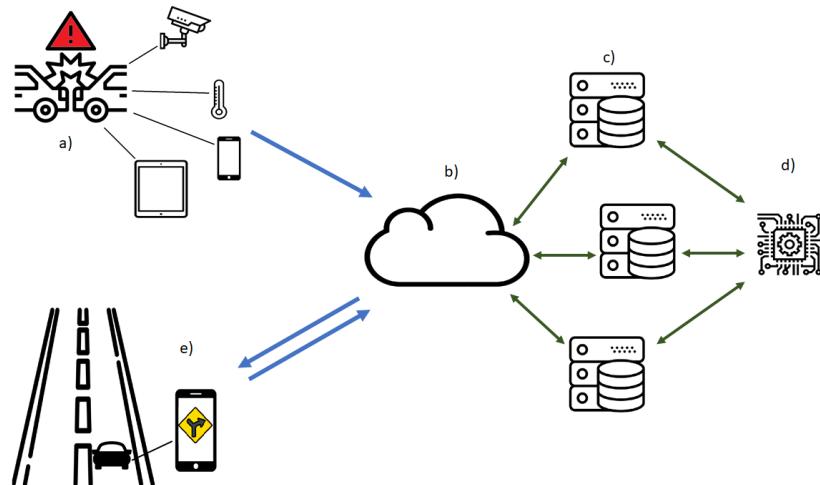


Figura 3.1: Schema del prototipo. a) Raccolta dei dati attraverso un sistema eterogeneo di dispositivi. b) Condivisione dei dati in un Middleware IoT. c) Storing dei dati all'interno di Databases d) Analisi sui dati. d) Query Geo-Referenziata sulle condizioni di traffico.

3.2 La Raccolta dei Dati

Il primo passo per l'implementazione di un sistema come quello proposto nella sezione 3.1 è quello dello studio ed analisi dei dati necessari e la conseguente raccolta degli stessi.

Lo sviluppo delle tecnologie legate al mondo dell'IoT ha prodotto tutto un movimento tecnologico per la creazione di nuovi dispositivi e per il miglioramento di quelli esistenti. Questo ha consentito lo sviluppo di competenze tecnologiche nella realizzazione di unità di calcolo, circuiti integrati, sensori e strumenti di comunicazione sempre più avanzati ed economici. Questo movimento ha consentito inoltre di incrementare la diffusione di componenti elettroniche la cui potenza e costo li rendono disponibili all'utilizzo in qualsiasi applicazione.

In riferimento allo sviluppo di un sistema come quello ideato nella sezione 3.1, si richiede che un numero molto elevato di sensori venga disposto sulla rete stradale in maniera capillare, così da poter avere dati affidabili ed in real-time sulla condizione del traffico in tutto il territorio. Alcuni di questi sensori sono già presenti ma sono in numero talmente basso che, con i dati da essi estratti, si potrebbero solamente ottenere informazioni parziali sul traffico su una porzione molto limitata del territorio. Le difficoltà principali

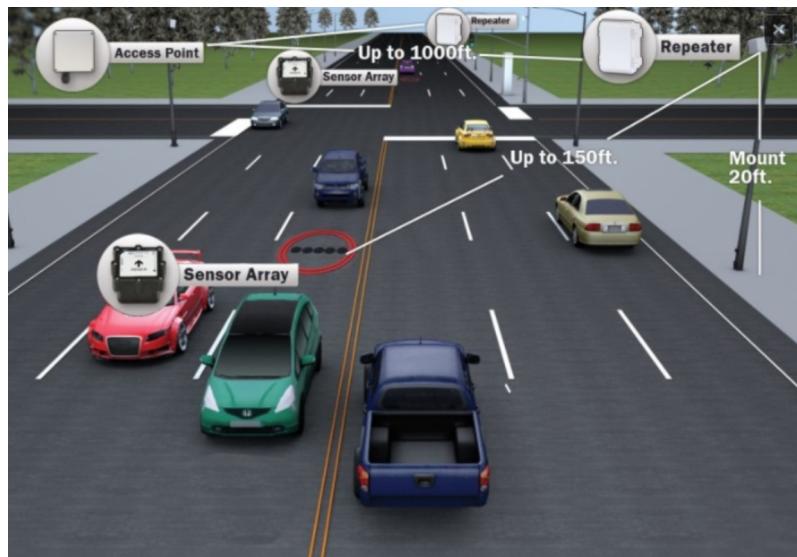


Figura 3.2: Principio di funzionamento dei sensori attualmente in uso per la misura del livello di traffico

nello sviluppo di una piattaforma per il controllo del traffico sono ovviamente quelle legate ai costi per la distribuzione di una rete di sensori che copra tutta la rete stradale. Pertanto, è necessario trovare altre soluzioni che possano risultare più sostenibili ed anche scalabili. Come evidenziato nel Capitolo 2, uno dei principali vantaggi di una piattaforma di Crowdsourcing è proprio quello di garantire che, attraverso il contributo non organizzato dei singoli, si possa raggiungere un obiettivo comune riducendo di fatto i costi.

Attualmente, infatti, la rete stradale è già ampiamente invasa da un numero estremamente elevato di sensori i quali sono diffusi in maniera capillare e senza costi aggiuntivi: gli Smartphone (o Tablet).

Gli Smartphone sono il più evidente esempio della diffusione di componenti elettroniche i quali, nelle loro evoluzioni hanno integrato al loro interno un numero sempre maggiore di sensori, diventando sempre più accurati e miniaturizzati. Sebbene il mercato degli

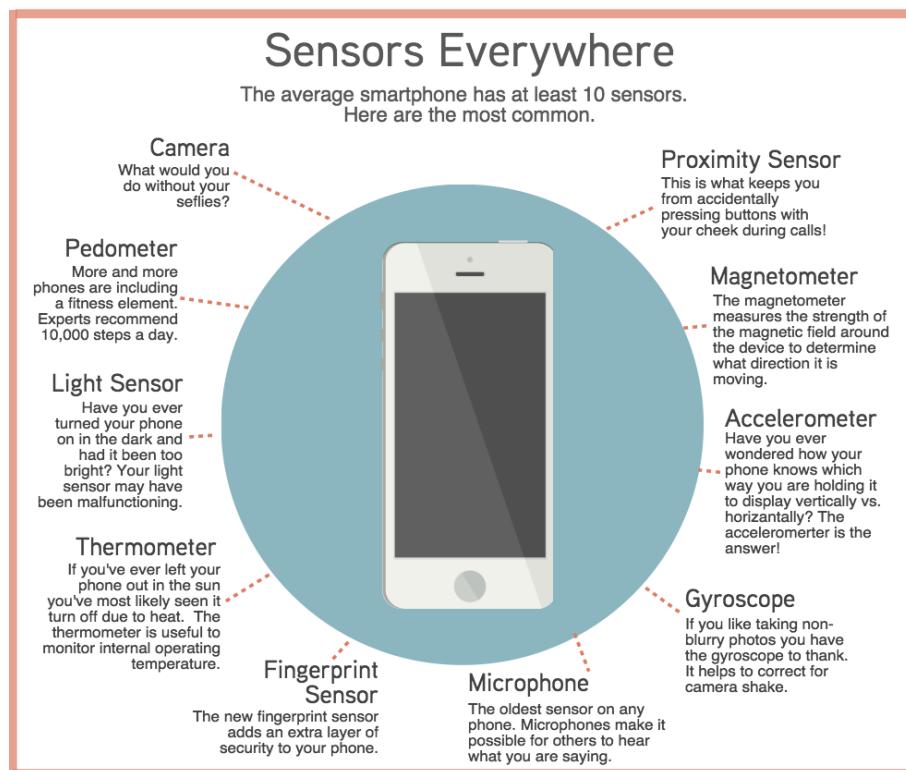


Figura 3.3: Crescita della disponibilità di sensori nell'equipaggiamento di uno smartphone
fonte: CornerAlliance

smartphone sia molto eterogeneo dal punto di vista della dotazione hardware e del sistema operativo, generalmente si avranno sempre a disposizione alcune tipologie di sensori che possiamo classificare in funzione della grandezza fisica che essi misurano:

- Termometro
- Accelerometro
- Giroscopio
- Microfono
- Camera

- GPS

Con l'aggiunta, in dispositivi più avanzati di:

- Pedometro
- Prossimità
- Impronte Digitali
- Magnetometro

Effettivamente, la sensoristica di cui è dotato un singolo smartphone non sarebbe in nessun modo in grado di stimare autonomamente il livello del traffico o l'eventuale presenza di incidenti. Tuttavia, una piattaforma che raccolga i dati di un gran numero di dispositivi e che utilizzi degli algoritmi di analisi dei dati, sarebbe in grado di effettuare delle stime sul livello del traffico in un' area geografica. Questo è quello che viene proposto in [27] dove viene analizzata una piattaforma per la raccolta dei dati relativi al traffico proveniente da un gran numero di smartphone e ne viene anche analizzato il rischio per la sicurezza di un utente che contribuisca ad una simile piattaforma dovendo condividere, tra gli altri, anche i dati relativi alla propria posizione.

Nel caso del prototipo ideato nella sezione 3.1, non si vuole soltanto realizzare una piattaforma che raccolga dati provenienti da smartphone, ma si vuole fare in modo che questa piattaforma sia adatta anche a raccogliere dati provenienti da infrastrutture stradali ed autostradali per il monitoraggio del traffico e per la comunicazione del livello di traffico. Si vuole cioè raccogliere in un'unica piattaforma i dati provenienti dai sensori stradali per il traffico come quello mostrato in Figura 3.2, quelli provenienti da stazioni meteo, smartphone, tablet, computer di bordo di automobili, stazioni di monitoraggio del traffico e così via. Sebbene nella fase di prototipazione ci si focalizzerà principalmente sulla raccolta dati da smartphone e della conseguente trasmissione degli stessi secondo un predefinito formato, tutte le scelte progettuali sono pensate per essere compatibili anche per la raccolta dati dai diversi dispositivi precedentemente elencati.

3.2.1 Accesso ai Sensori di uno Smartphone

L'aumento del numero dei sensori, l'aumento della loro accuratezza ed affidabilità, ha portato alla nascita di servizi che utilizzassero tutte o alcune delle letture fornite dai sensori per offrire dei servizi all'utenza.

Alcuni di questi servizi consentono ad esempio di monitorare la qualità del proprio sonno, il numero di Kcal bruciate ogni giorno, regolare in modo automatico la luminosità e la gamma di colori del proprio schermo per salvaguardare gli occhi dell'utilizzatore ed una serie di altri infiniti servizi, tutti possibili sulla base dei dati provenienti da uno o più sensori.

Questi servizi o applicazioni non sono altro che l'ultimo step di una struttura verticale nella quale, ai primi livelli si trovano effettivamente le componenti hardware del sensore che producono una uscita elettrica. Si rende necessario poi filtrare, amplificare e condizionare

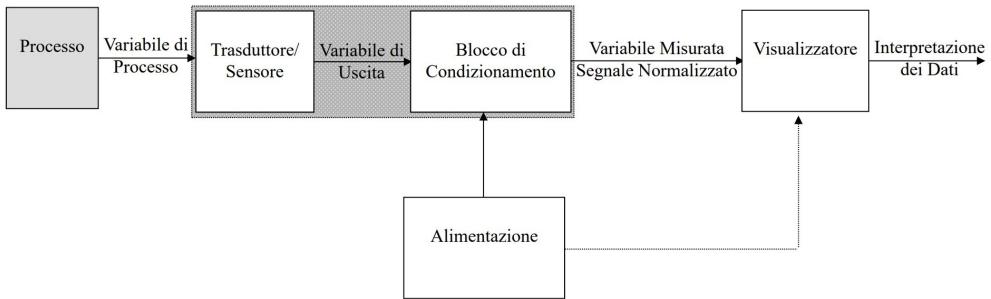


Figura 3.4: Schema generale di un sistema di misura

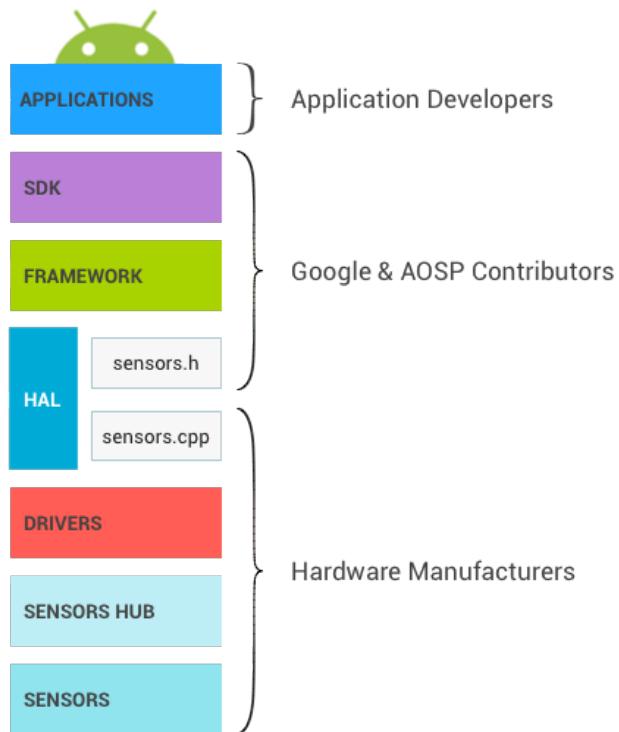


Figura 3.5: Estratto dello stack utilizzato nel sistema operativo Android per rendere disponibili le letture dei sensori ad applicazioni e servizi

questo segnale elettrico al fine di renderlo comprensibile e trasmissibile come in Figura 3.4. La variabile fisica o variabile di processo che si intende misurare viene quindi trasdotta o convertita in un'altra forma (generalmente elettrica) che è più semplice da trasmettere ed elaborare. Questa variabile di uscita prodotta dal blocco sensore e trasduttore non è tuttavia ancora adatta ad essere utilizzata. Questa subirà quindi un nuovo processo di

condizionamento nel quale verrà filtrata, amplificata e linearizzata. Di qui il sistema operativo si occupa di interpretare ed eventualmente tradurre in un diverso formato le letture dei sensori in modo da renderle disponibili attraverso una interfaccia alle applicazioni e servizi dei livelli superiori che ne fanno richiesta. Tali applicazioni o service provider saranno quindi solo i consumatori finali di questi dati dai quali estraranno le informazioni da mostrare all’utente finale che usufruisce del servizio. Come evidente, nello stack utilizzato dal sistema operativo per smartphone Android Figura 3.5 viene offerto agli sviluppatori che realizzano servizi ed applicazioni una interfaccia diretta e standardizzata per l’utilizzo delle letture provenienti dai sensori. A prescindere dal sistema operativo utilizzato e dal tipo di sensori di cui è dotato lo smartphone, sarà offerto agli sviluppatori una interfaccia chiamata SDK ovvero Software Development Kit che indica un insieme di strumenti per lo sviluppo e la documentazione di software. In questo modo, coloro che si occupano di fornire servizi basati sulle letture di sensori, non dovranno preoccuparsi di elaborare, tradurre e condizionare l’uscita del sensore, ma solo di utilizzarla. [18] [3]

3.2.2 Parametri di Qualità dei Sensori

Ai sensori è generalmente richiesto di funzionare in ambienti ostili all’uomo e spesso ostili anche all’elettronica. Inoltre, un sensore sarà ritenuto tanto migliore quanto più esso riuscirà ad essere sensibile ad una sola grandezza ed insensibile a tutte le altre grandezze o rumore che possono disturbarne la misura. [5] [34]

Si può quindi sintetizzare che la **qualità di un sensore** sia determinata dalla sua sensibilità alla grandezza che si vuole misurare e dalla sua insensibilità a tutte le altre grandezze fisiche ambientali. Essendo interessati alla valutazione delle prestazioni di un sensore in modo da poterne valutare la sua affidabilità e la qualità delle sue letture, va considerato che l’accuratezza del dispositivo può variare con altre grandezze di influenza (temperatura, umidità, ecc.) e che spesso, si perviene alla misura di una grandezza fisica di interesse in maniera indiretta, introducendo così una propagazione dell’errore. Una prima valutazione

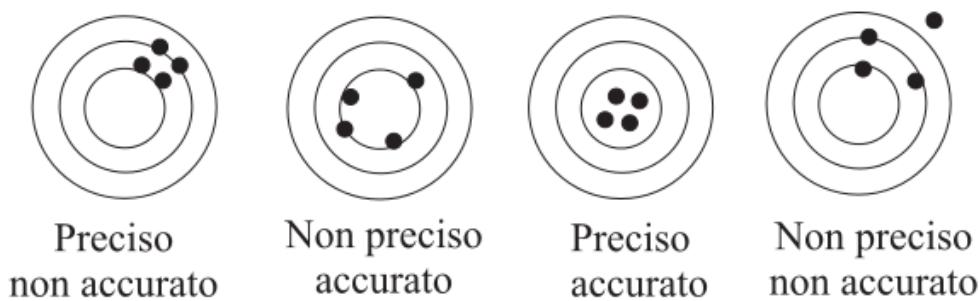


Figura 3.6: Differenza tra la definizione di Precisione e quella di Accuratezza di una misura

sulla qualità di un sensore può essere effettuata attraverso due indici: **responsivity** e **detectivity**.

La **responsivity** o **sensitivity** rappresenta l'efficienza di trasduzione del sensore ed è normalmente espressa come il rapporto di due unità di misura, in genere differenti.

$$r = \frac{\partial S_o}{\partial S_i} \quad [S_o \setminus S_i] \quad (3.1)$$

La **detectivity** rappresenta la capacità di un sensore di rendere disponibile un segnale dipendente dalla grandezza di ingresso e di tenerlo distinto dal rumore da esso stesso prodotto. Il suo valore dipende dal *noise floor* ed è legato alla soglia o minimo segnale rilevabile. Si misura come il reciproco dell'unità di misura dell'ingresso.

$$d = \frac{S_o \setminus N_o}{S_i} = \frac{r}{N_o} \quad [S_i^{-1}] \quad (3.2)$$

Queste informazioni, saranno utili al fine dello sviluppo del prototipo in quanto, dal momento che la piattaforma ideata nel sezione 3.1 si occuperà di raccogliere i dati provenienti da un numero molto elevato di dispositivi eterogenei, questo significa anche che le letture dei sensori provenienti dai diversi sensori saranno qualitativamente eterogenee.

Pertanto, un primo requisito, sarà quello di valutare i parametri di qualità relativi alle letture dei sensori di ciascun dispositivo ed eseguire in questo modo un primo filtraggio dei dati in modo da escludere dall'analisi quelle letture affette da un grado di incertezza troppo elevato o non compatibile con l'analisi effettuata. Nel caso degli smartphone e tablet, queste informazioni sono messe a disposizione all'interno del SDK.

3.2.3 Filtraggio delle Letture dei Sensori

Nella Figura 3.4 è possibile notare come nel sistema di misura sia presente una componente che si occupi di filtrare, amplificare e linearizzare l'uscita fisica del sensore: il Blocco di Condizionamento. Questo blocco è generalmente di tipo software ed è realizzato ad-hoc per il particolare sensore e per il particolare utilizzo. Il suo scopo è quello di portare il segnale in uscita dal sensore in un formato compatibile con il dispositivo di elaborazione che lo segue riducendo al massimo gli effetti negativi di carico. [5] Allo stesso modo, anche per quanto riguarda i sensori di cui sono dotati gli Smartphone, è necessario che le relative uscite vengano condizionate in maniera opportuna in funzione della particolare applicazione.

Prima però è necessario effettuare un approfondimento sui sensori che possono equipaggiare uno smartphone (e che sono necessari alla predizione delle condizioni stradali). All'interno degli smartphone, infatti, è possibile incontrare due diverse tipologie di sensori:

- Sensori Hardware
- Sensori Software

Questa distinzione, sebbene non evidente all’utente finale, produrrà diverse tipologie di dati che dovranno quindi essere trattati diversamente.

I **Sensori Hardware** sono componenti fisiche che sono effettivamente montate all’interno di uno smartphone. Pertanto, le letture di questi sensori provengono dalla misura di una certa grandezza fisica attraverso un sistema di misura come quello mostrato in Figura 3.4. Un esempio ne sono l’accelerometro, il magnetometro ed il sensore per l’intensità luminosa. I **Sensori Software** non sono sensori fisici composti cioè da componenti elettroniche che trasducano una certa grandezza fisica. Questi sensori, emulano il comportamento di un sensore hardware ma le loro uscite sono spesso derivate da uno o più sensori hardware, opportunamente rielaborate via software. Un esempio ne sono sensore di orientamento, vettore di rotazione, pedometro.

Sebbene questa distinzione sia completamente invisibile a coloro che utilizzano questi dati e sia anche invisibile a coloro che sviluppano applicazioni e servizi sfruttando l’SDK messo a disposizione dal sistema operativo, è necessario tenerla a mente perché i dati in uscita dalle diverse tipologie di sensori dovranno essere trattati diversamente. I dati in uscita dai sensori software saranno infatti direttamente utilizzabili in quanto hanno già subito un processo di condizionamento proprio perché, per definizione, sono derivati da altri sensori hardware. Si rende invece necessario sviluppare un sistema di filtraggio ed eventualmente linearizzazione per le letture dei sensori hardware.

Il problema dei sensori hardware è che la frequenza con la quale le uscite variano il loro valore è così elevata che, se mappassimo tutte queste piccole variazioni (Rumore), i valori oscillerebbero notevolmente. Più in generale, le letture provenienti dai sensori sono ricevute con una diversa frequenza ma, i picchi da essa raggiunti, sono troppo elevati. Il processo

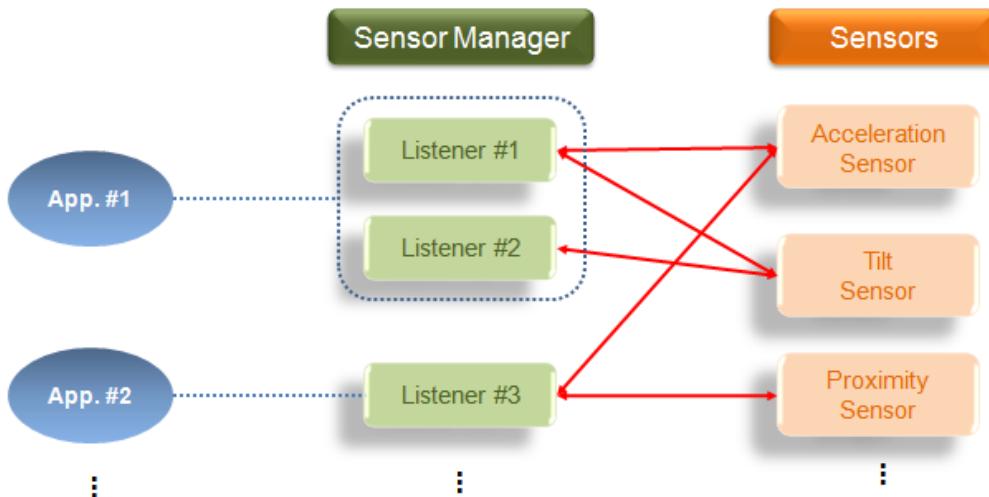


Figura 3.7: Princípio con il quale nel sistema operativo Android è possibile recuperare le letture di un sensore ed utilizzarle in una applicazione. [18]

che viene seguito nel sistema operativo Android per rendere disponibile la lettura di un

sensore ad una applicazione o servizio che ne faccia richiesta è quello di instanziare un SensorListener il cui ruolo è appunto quello di restare in ascolto sull'uscita di un dato sensore e di notificare la applicazione nonappena venga rilevata una variazione del valore misurato dal sensore. La frequenza con la quale questo processo avviene dipende dalla velocità del sistema operativo e dalla capacità computazionale del dispositivo. Questo significa che dispositivi con una maggiore capacità di calcolo interrogheranno un numero maggiore di volte il sensore e riceveranno un numero maggiore di risposte affette, inevitabilmente, da rumore. Questo significa che è necessario utilizzare i soli valori necessari ed utili e filtrare il rumore.

La soluzione è l'applicazione di un filtro passa-basso sulle letture provenienti dai sensori in modo che esso consenta il passaggio alle sole basse frequenze del segnale e ne attenui il valore alle frequenze superiori a quella di taglio.

Si prenda ad esempio l'accelerometro di uno smartphone le cui letture indicano l'accelerazione del dispositivo rilevata in un predefinito sistema di riferimento. A causa di un

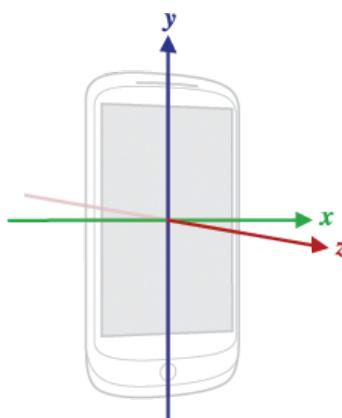


Figura 3.8: Sensore di accelerazione a bordo di uno smartphone

disturbo esterno come fattori ambientali o jerks o vibrazioni, viene aggiunto alla lettura del sensore una gran quantità di rumore. Queste componenti ad alta frequenza del segnale, provocano un'ampia oscillazione dei valori letti dal sensore. Questi disturbi, potrebbero essere tollerabili nella maggior parte delle applicazioni ma se si vogliono ottenere delle migliori letture è necessario progettare e sviluppare un filtro passa-basso che filtri a livello software le uscite di sensori hardware. In questo modo, è possibile filtrare le componenti di rumore ad alta frequenza utilizzando un opportuno valore di soglia.

```

1 for i from 1 to n
2     y[i] := y[i-1] + a*(x[i] - y[i-1])
3 end

```

Listing 3.1: Pseudo-codice relativo all'implementazione di un filtro passa-basso

Il Listing 3.1 corrisponde ad una implementazione di un filtro passa-basso che prenda l’uscita y di un sensore e la filtri con un filtro passa-basso avente a come coefficiente di soglia. La scelta del parametro a dipende da quanto si intende dimensionare la frequenza di taglio del filtro. Quanto più essa sarà alta, tanto più rumore entrerà nel filtro viceversa, quanto più sarà bassa tanto più lento sarà il segnale. In [18] viene proposta una soluzione intermedia con la scelta del parametro $a = 0.25$.

3.3 Il Protocollo DATEX

La raccolta dei dati è solo una piccola porzione del problema. Per ottenere il massimo da questi dati è necessario condividerli in modo da sfruttare le potenzialità delle piattaforme di Crowdsourcing nelle quali, le analisi effettuate su un gran numero di dati potrebbero fornire delle deduzioni e previsioni molto più accurate e significative. Si rende quindi necessaria una comunicazione tra i dispositivi che si occupano della raccolta dati.

Diventa quindi necessario abilitare la comunicazione tra tutti i diversi dispositivi che si occupano della raccolta dati considerando la loro eterogeneità in termini di:

- Capacità computazionale
- Sensoristica a disposizione
- Grandezze fisiche misurate
- Qualità delle letture dei sensori
- Costruttore del dispositivo

DATEX (da Data Exchange) è il protocollo che abilita la comunicazione tra dispositivi eterogenei per la condivisione di dati legati al traffico stradale ed autostradale. Questo protocollo fa in modo che differenti dispositivi possano comunicare tra loro e con un Middleware, definendo cioè uno Standard per la comunicazione condiviso tra tutti i partecipanti alla comunicazione.

Nella fattispecie, DATEX è il linguaggio utilizzato nella Comunità Europea per lo scambio di informazioni legate al traffico in modo da consegnare informazioni comprensibili all’utenza finale. DATEX è infatti stato progettato e sviluppato come un sistema per lo scambio di dati dalla Comunità Europea per la standardizzazione dell’interfaccia di comunicazione tra i gestori di strade ed autostrade, i produttori dei dati in strade ed autostrade e gli utilizzatori dei servizi sul traffico. [39]

Una prima versione del Protocollo DATEX I è stata iniziata già nel 1990 mossa dalla necessità di un meccanismo per lo scambio di dati tra centri per il monitoraggio del traffico e gli operatori stradali. Ben presto, tuttavia, dato lo sviluppo tecnologico del XXI secolo, è sorta la necessità di integrare in questo standard anche i fornitori di servizi. Per questo motivo, viene sviluppato lo standard DATEX II con lo scopo di distribuire le informazioni sul traffico indipendentemente dalla lingua e dall’aspetto. In questo modo, viene eliminata la probabilità che nascano delle incomprensioni o errori di traduzione.

Lo Standard DATEX II non è stato pensato per fornire delle rigide specifiche di comuni-

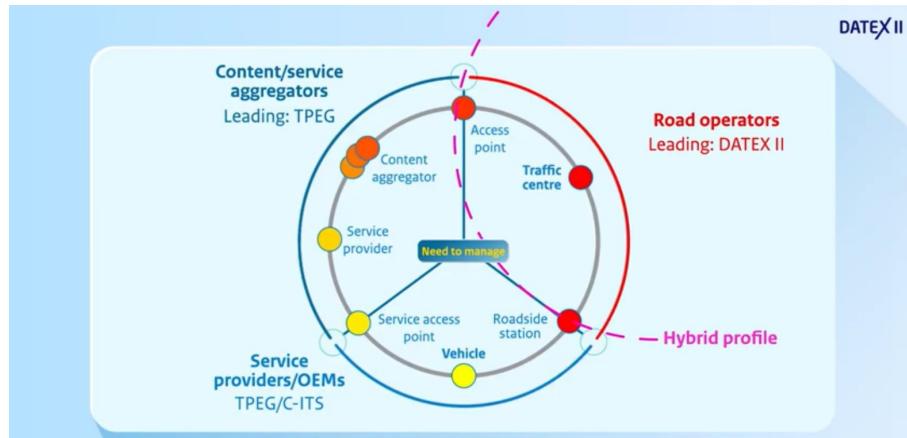


Figura 3.9: Panoramica del Mondo ITS e le relative sfide tecnologiche

Type	Remarks
Accidents Obstruction Activities	public events, police operations etc.
Abnormal Traffic Driving conditions Poor road infrastructure Network Management Sign information Roadside Assistance Car parking information Transit information Service disruption	Queues Weather, Road surface Closures, Restrictions VMS settings Number of vacant spaces Rail and Ferry connections Motorway service area
Measured Traffic Data Measured weather/pollution	Travel times, Flow data Wind, precipitation, visibility, air quality
Traffic View	Combination of event informations and URLs for CCTV cameras

Tabella 3.1: Le metodologie di comunicazione [15]

cazione ma piuttosto come un Protocollo che ammettesse una certa libertà di scelta e che fosse in grado di evolversi per consentire, in futuro, lo scambio di informazioni aggiuntive. Il dominio di dati che possono essere scambiati attraverso DATEX II riguarda tutti i dati relativi al traffico ed alle informazioni sul viaggio per strade urbane ed inter-urbane. Spaziando tra eventi legati al traffico come incidenti e lavori stradali ad eventi legati al flusso

di traffico come permanenza sulle strade e durata del viaggio.

Risultano quindi evidenti le motivazioni che hanno portato alla scelta dello standard DATEX II nella progettazione del prototipo ideato in sezione 3.1. Rispetto ad altri protocolli di comunicazione, infatti, DATEX II presenta delle caratteristiche che ben si adattano allo scopo:

1. Sviluppato, Supportato ed Utilizzato dalla Comunità Europea
2. Progettato per lo scambio di dati relativi al traffico
3. Supporta lo scambio dati tra dispositivi eterogenei
4. Indipendente dalla tecnologia utilizzata per lo scambio dati
5. Flessibile per adattarsi a nuove necessità e dati da scambiare
6. Geo-Referenzia i dati scambiati tra i vari dispositivi
7. Documentazione completa per gli sviluppatori che vogliono integrare il protocollo nelle loro applicazioni
8. Documentazione completa per gli sviluppatori che vogliono ampliare il protocollo in funzione di nuove necessità

Progettando quindi il sistema della sezione 3.1 in modo che supporti il protocollo DATEX II, si fa in modo che il sistema possa essere aperto a ricevere non solo i dati provenienti da altri smartphone o tablet sui quali è installato lo stesso sistema, ma anche da Centri per il controllo del traffico, infrastrutture stradali e centri di informazione su tutta la rete stradale Europea, avendo così a disposizione un più ampio data-source sulla base del quale effettuare previsioni, analisi ed inviare comunicazioni agli utenti.

3.3.1 DATEX II Data Model

Il principale vantaggio del Protocollo DATEX II è la sua flessibilità che lo rende a prova di futuro. Questa caratteristica viene fornita allo standard da un ricco ed ampiamente sviluppato modello dei dati in formato UML (Unified Modeling Language). Questo consente allo Standard DATEX II di avere un modello dei dati che sia indipendente dalle sue implementazioni e che quindi possa essere mappato diversamente in diverse implementazioni. Attualmente, questo modello dei dati è mappato in informazioni codificate attraverso XML (eXtensible Markup Language) con la creazione di un XML schema. XML e XML schema sono tra i sistemi per la definizione dei modelli di dati più ampiamente utilizzati. Questa scelta garantisce che l'interfaccia DATEX II sia facilmente integrabile in quante più implementazioni ed applicazioni possibili.

In più, DATEX II implementa un alto livello di flessibilità attraverso una opzione di estensibilità del modello dei dati. Questo consente ad utenti singoli di scambiare dati attraverso il modello dei dati DATEX II, ma che sia stato adattato alle loro particolari necessità e preferenze, rimanendo comunque conforme allo standard DATEX II. Questo viene reso

possibile offrendo agli utenti la possibilità di ampliare il modello dei dati di DATEX II, seguendo una definita procedura, in modo che si mantenga l'interoperabilità con altri dispositivi e centri di controllo per il traffico che utilizzino una qualsiasi altra implementazione conforme a DATEX II. [15]

Lo sviluppo di DATEX II si è focalizzato sulla creazione di un documento di riferimento nel quale viene posta particolare enfasi alla distinzione tra Data (contenuto) e Meccanismo di Scambio. La descrizione tecnica fornisce una chiara distinzione tra Platform Independent Modelling (PIM) e Platform Specific Modelling (PSM). PIM si riferisce al dominio dei dati scambiati relativi al traffico mentre PSM si riferisce alle specifiche informazioni scambiate ed alla tecnologia di comunicazione utilizzata. Questa separazione tra i due domini produce una migliore comprensione dello standard rendendolo più facile da applicare.

Level A - Data Model

DATEX II viene sviluppato a valle di un periodo di studio ed osservazione sul tipo e numero di dati che sono scambiati dai dispositivi IoT nella Comunità Europea. Risultato di questa osservazione è il modello dei dati in formato UML di DATEX II anche noto con il termine "Level A Data Model". Sebbene ampiamente completo e documentato, potrebbero ancora esserci delle situazioni nelle quali l'interfaccia dati richiesta da un particolare utente possa non essere presente nel Data Dictionary di DATEX II in quanto, ad esempio, sono dati riferiti ad un contesto nazionale o locale.

In questo caso, questi utenti sono in grado di fornire una estensione al modello che includa i concetti mancanti, nota anche come "Level B Extension Model". Agli utenti viene quindi data la possibilità di aggiungere una limitata estensione al diagramma UML, seguendo predefinite istruzioni. Queste estensioni (Level B) devono comunque mantenere l'interoperabilità con lo standard DATEX II in modo che altri dispositivi che siano compatibili con DATEX II (ad esempio con il Level A), possano ancora mantenere una interoperabilità e che cioè siano in grado di processare i dati (o Publications) provenienti da un dispositivo che implementi un modello dei dati personalizzato (ad esempio di Level B), senza essere ovviamente in grado di processare i dati aggiunti dal particolare client.

Soltanto altri client specializzati, che cioè implementano tutto il modello dei dati modificato (di Level B) saranno in grado di processare tutto il contenuto, comprese le estensioni. Lo scopo principale per cui il Level B è progettato riguarda i soli casi in cui uno specifico utente implementi ed utilizzi il modello dei dati di Level A per la maggior parte delle comunicazioni ma che, in alcuni casi, non sia sufficiente a soddisfare le proprie necessità e risulta necessario ampliarlo.

Questo tuttavia non comprende il caso nel quale si vogliono definire dei concetti che siano completamente nuovi o completamente fuori dallo scope del Level A. In questo caso, gli utenti che dovessero incorrere in tali necessità hanno ancora la possibilità di ampliare il modello dei dati nel diagramma UML così da avere ancora un certo livello di interoperabilità. Questi modelli così modificati sono chiamati "Level C Extensions". Le implementazioni che forniscono dati (o Publications) di Level C non sono generalmente in grado di essere interoparabili con le implementazioni di Level A o Level B. .

Level B - Extension Mechanism

Sebbene il Level A sia già abbastanza ricco ed il suo modello dei dati abbastanza completo, non è escluso che nel tempo possano sorgere delle nuove necessità o applicazioni che vogliono aggiungere nuovi concetti ed attributi al modello esistente. Al fine di rendere il Protocollo DATEX II adatto a sopravvivere alle innovazioni nel tempo, è stato progettato un meccanismo formale attraverso il quale il modello dei dati di Level A può essere esteso. È necessario che queste nuove applicazioni che estendono il Level A, chiamate Level B, mantengano una compatibilità con il DATEX II nella sua versione base (Level A). Questo consente alle nuove implementazioni dello standard che arricchiscono il Level A con l'aggiunta di nuovi concetti o attributi di restare comunque compatibili con lo standard nella sua versione base e quindi di interpretare correttamente le publications. Fornitori o Client che volessero utilizzare completamente le informazioni definite in queste estensioni al Level A devono invece implementare il modello dei dati definito nell'estensione stessa Level B e non il solo modello base Level A.

Presso www.datex2.eu è disponibile un processo di registrazione di nuove estensioni di tipo B allo standard DATEX II. Qualora queste estensioni dovessero raggiungere ampio consenso, saranno integrate direttamente nella versione base dello standard.

Level C - Extensions

Dopo aver analizzato le possibilità offerte dalle versioni A e B dello standard DATEX II e dalla loro intercompatibilità, alcuni utenti potrebbero ancora ritenere necessario un ampliamento ancora più profondo dello standard con l'aggiunta di nuove definizioni e di nuovi scope. Queste modifiche potrebbero anche rivelarsi troppo distanti dal modello dei dati di Level A e quindi non implementabili con una sola estensione del Level A in Level B. Per questo motivo, viene anche sviluppato il concetto di Level C le cui implementazioni sono considerate non conformi allo standard e quindi non interoperabili con i Level A, B. Nonostante questa incompatibilità, le implementazioni di Level C devono quantomeno utilizzare le regole di modelling e protocolli di scambio dati che siano comuni a DATEX II. Questo potrebbe consentire in futuro uno scambio di idee e portare alla creazione di un modello dei dati unificato.

Data Dictionary

Lo standard DATEX fornisce le definizioni di quelli che sono i concetti e gli attributi utilizzati nel modello dei dati in modo da aiutare gli sviluppatori nell'utilizzo ed implementazione dello Standard. Queste definizioni sono raccolte in un diagramma UML per gli addetti ai lavori ed in un file Excel che documenti tutte le definizioni in una forma più comprensibile. In questo modo, è possibile istantaneamente valutare se lo standard comprenda già tutti gli attributi necessari alla particolare istanza o se sia necessario aggiungerne di nuovi.

3.3.2 XML Schema

Il formato XML sta vedendo una notevole crescita del suo utilizzo per lo scambio di informazioni. Una componente fondamentale di questo approccio sono gli XML Schemas. Basandosi su un modello dei dati e su un dizionario dei dati, gli XML Schemas sono progettati per adattarsi ad una particolare area applicativa. Questi schemi sono degli strumenti per capire il contenuto dei dati che sono scambiati.

Il modello UML viene inizialmente esportato in un file XMI ed in seguito, attraverso un tool di conversione, viene trasformato in un XML Schemas. Questi XML Schemas sono utilizzati per lo sviluppo software. XMI (XML Metadata Interchange) è invece uno standard OMG per lo scambio di metadati in formato XML. L'uso più comune che se ne fa di XMI è proprio quello di interscambio di modelli UML.

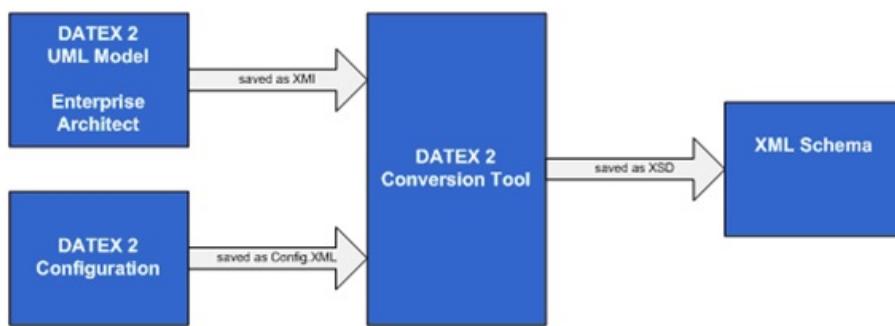


Figura 3.10: Workflow di un processo di conversione automatico

3.3.3 Exchange Mechanisms

Ci si focalizza ora su quali metodologie sono supportate dal protocollo DATEX II per lo scambio dei dati tra i diversi dispositivi che partecipano alla piattaforma.

DATEX II offre metodologie di push e pull per lo scambio di informazioni.

La modalità **push** consente al fornitore di inviare le informazioni al client. Questa operazione può essere effettuata periodicamente ad intervalli di tempo regolari oppure all'occorrenza quando il fornitore dovesse ritenerlo opportuno.

La modalità **pull** consente al client di richiedere dati ad un fornitore.

Una descrizione di alto livello di come avvenga la comunicazione tra fornitore e consumatore dei dati, senza considerare i dettagli tecnici, può essere sintetizzata in tre operazioni principali:

1. **Publisher Push on Occurrence:** La comunicazione e l'invio dei dati viene cominciata dal publisher ogni volta che i dati cambiano
2. **Publisher Push Periodic:** La comunicazione e l'invio dei dati viene cominciata dal publisher ad intervalli di tempo regolari

3. **Client Pull:** La comunicazione viene cominciata dal Client che si aspetta l'invio dei dati richiesti come risposta dal Publisher

Le metodologie di comunicazione viste finora possono essere sia online che offline. Nel Platform Specific Modelling (PSM), la sezione relativa allo scambio dei dati è stata progettata per essere indipendente dal contenuto che viene scambiato, ovvero dal payload. In questo modo, le informazioni relative allo scambio dei dati possono essere studiate indipendentemente dal modello dei dati in formato UML di DATEX II, garantendo così la separazione dei due campi.

Questa separazione logica ed implementativa, non solo rende più facile la comprensione e l'applicazione del protocollo stesso, ma rende anche possibile riutilizzare una delle due sezioni qualora l'altra dovesse cambiare ed evolversi nel tempo per adattarsi a nuove necessità. Ad esempio, potrà variare nel tempo il modello dei dati dello standard DATEX II perché potrebbero essere proposte delle nuove estensioni dello standard, tuttavia, non è necessario che vari anche il meccanismo di scambio dati in quanto indipendente dal payload.

3.3.4 Exchanged Data

Prima di analizzare nel dettaglio come sia fatto il diagramma UML relativo allo Standard DATEX II ed implementarlo nel sistema che prototiperà l'idea della sezione 3.1, si analizzi più nel dettaglio i dati relativi al traffico che sono supportati dallo standard.

Questi dati che vengono scambiati, sono composti da elementi base che sono disponibili all'interno di una pubblicazione o alternativamente potremmo dire che tra i dispositivi viene scambiata una publication composta da elementi o attributi base.

Elementi Base

Una Publication in DATEX II è composta da elementi base che possiamo raggruppare in macro-categorie:

1. Eventi relativi a condizioni stradali e traffico ("Traffic Elements")
2. Azioni dell'Operatore
3. Incidenti
4. Dati misurati o elaborati (tempo di viaggio, velocità del traffico stimata, intensità del traffico stimata, rilevazioni metereologiche, ...)
5. Messaggi mostrati sui Variable Message Signs (VMS)
6. Informazioni di Servizio (assenza di corsie di emergenza, ritardi su treni, ...)

In più, sono anche presenti elementi come: Zone di Posteggio, Posizioni dei pannelli VMS e Posizioni dei siti di misura. Queste informazioni, sebbene non siano dei dati elementari, sono comunque necessarie al Client al fine di comprendere ed utilizzare correttamente le informazioni degli elementi base.

Eventi relativi a condizioni stradali e traffico

Questi eventi riguardano tutti situazioni che non sono inizializzate da un operatore del traffico ma che lo obbligano ad intraprendere una decisione su come affrontare l'evento. Possono essere classificati in:

- Traffico Anormale (lunghe code, rallentamenti)
- Incidenti
- Attività (eventi pubblici, disturbi al normale funzionamento)
- Condizioni Stradali. Possono essere o meno legate a condizioni atmosferiche o all'ambiente
 - Ostruzioni
 - presenza di animali
 - presenza di veicoli in panne
 - ostruzioni dovute dall'ambiente (alberi)
 - ostruzioni dovute a infrastrutture (caduta di cavi)
 - ostruzioni dovute a persone
- Incidenti ad infrastrutture o sistemi stradali (VMS guasti)

Azioni dell'Operatore

Riguardano eventi che sono inizializzati da un operatore del traffico e possono essere classificati in:

- Manutenzione (chiusura di strade, traffico alternato)
- Lavori stradali (limiti temporanei, deviazioni)
- Assistenza a bordo strada

Incidenti

Questa sezione contiene in particolar modo gli eventi relativi alla disponibilità di corsie, rallentamenti ed eventuali deviazioni.

Dati Misurati o Elaborati

Questo set di dati può essere derivato da input diretti di stazioni per il traffico o strumentazione di uno specifico sito di misura, i quali dati possono essere ricevuti periodicamente o possono essere inviati in funzione delle condizioni di traffico in una specifica località.

- Valori di Traffico misurati

- flusso
 - velocità
 - progresso
 - concentrazione dei veicoli in coda
- Stato del traffico
 - flusso libero, intenso, congestionato, sconosciuto
 - Tempo di viaggio
 - tempo stimato
 - tempo di viaggio nominale atteso
 - Condizioni Atmosferiche
 - precipitazioni, vento, temperatura, inquinamento, condizioni del manto stradale e visibilità.
 - previsioni metereologiche

Messaggi mostrati sui VMS

Questo set di dati include diversi messaggi che possono essere mostrati sui differenti VMS (Variable Message Signs) in funzione della tecnologia che utilizzano, immagini e testo. Possono anche includere informazioni sulla posizione dei dispositivi VMS e sul loro status.

Informazioni di Servizio

Riguardano informazioni su un servizio che potrebbe influenzare il comportamento dei guidatori e quindi le caratteristiche del traffico.

- Informazioni sul Transito dei veicoli
 - informazioni circa altri mezzi di trasporto (treni, tram, voli)
- Indisponibilità di servizi (aree di sosta chiuse, aree di rifornimento chiuse)
- Indisponibilità di servizi stradali (chiamate di emergenza fuori uso)

3.3.5 Publication di Elementi Base

Gli elementi base mostrati in precedenza, possono essere scambiati individualmente tra dispositivi che implementano lo standard DATEX II o possono anche essere scambiati in gruppi di elementi. Questi scambi di gruppi di elementi base prendono il nome di Publication e, in funzione dei dati trasmessi, possono esserci cinque tipi fondamentali di Publication:

1. **Situation Publication:** Usata per Elementi legati al traffico, azioni dell'operatore, incidenti o informazioni di servizio
2. **Elaborated Data Publication:** Usata per lo stato del traffico, per i tempi di viaggio e per valori misurati di traffico e condizioni atmosferiche
3. **Measured Data Publication:** Usata per Misure del traffico e rilevamenti atmosferici, stato del traffico e tempo di viaggio
4. **Parking Publication:** Usata per le informazioni su parcheggi ed aree di sosta
5. **VMS Publication:** Usata per i messaggi da mostrare sui VMS e sulle informazioni relative agli VMS.

3.3.6 Utilizzo del DATEX II Data Model

Nelle sezioni precedenti, servendosi della documentazione ufficiale messa a disposizione dallo Standard DATEX (<https://datex2.eu/>), si sono evidenziati i vantaggi nell'adozione del protocollo ed anche le sue peculiarità implementative. Data la flessibilità offerta dal protocollo, risulta necessaria ora una fase di analisi su quelle che siano le caratteristiche del protocollo che è necessario implementare all'interno del sistema. Già da una prima analisi sull'idea del prototipo da realizzare, risulta subito evidente che sia superfluo implementare il protocollo DATEX II nella sua interezza in quanto, non tutte le sue componenti saranno necessarie al sistema.

Il punto di partenza di questa analisi è quello di adottare il Processo Unificato (UP) con il quale produrre la documentazione necessaria allo sviluppo del software relativo al sistema che si vuole implementare definito nella sezione 3.1. A tal proposito, è bene introdurre brevemente il Processo Unificato ed in generale gli strumenti messi a disposizione dall'Ingegneria del Software che saranno utilizzati nei capitoli seguenti.

Il Processo Unificato

Il Processo Unificato (UP) è uno dei modelli messi a disposizione dall'Ingegneria del Software che fornisce una sequenza di fasi da seguire per la progettazione e realizzazione di Software dalle grandi dimensioni. Il Processo Unificato e gli altri modelli per lo sviluppo di Software definiscono la lista di operazioni da seguire al fine di ottenere un prodotto software composto dal suo eseguibile, più la documentazione ad esso associato. Risulta infatti necessario, per software dalle grosse dimensioni, documentare ogni singola fase dello sviluppo, comprese quelle di progettazione.

Nello sviluppo di questo particolare sistema, la scelta è ricaduta sul Processo Unificato in quanto questo modello sia particolarmente flessibile ed adatto alla modellazione visuale attraverso l'Unified Modelling Language (UML) che è la stessa tipologia di modellazione utilizzata dallo standard DATEX II. In questo modo, sarà più semplice integrare il Software che si intende sviluppare in questo lavoro di tesi con il Protocollo DATEX II.

Il Processo Unificato si compone di diverse fasi, nelle quali ci si focalizza su vari aspetti del processo software e tra le quali si può ciclicamente iterare per implementare nuove

funzionalità del Software:

1. **Ideazione:** Fornisce una visione approssimativa del progetto. Stime sul progetto di carattere economico e relative alle tempistiche.
2. **Elaborazione:** Viene raffinata la visione del progetto, vengono identificati la maggior parte dei requisiti e sono realizzate delle stime più realistiche del progetto.
3. **Costruzione:** Fase di implementazione iterativa degli elementi rimanenti e preparazione al rilascio.
4. **Transizione:** Beta test e Rilascio agli utenti.

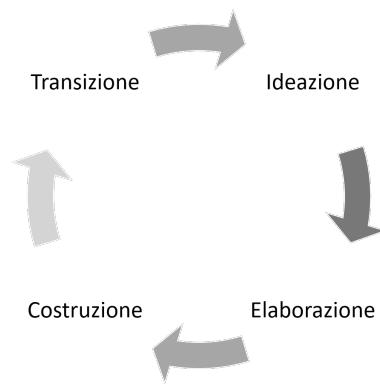


Figura 3.11: Fasi del Processo Unificato

Nella fattispecie, si produrrà in questo lavoro di tesi una parte della documentazione prevista dal Processo Unificato per la realizzazione del Software che prototiperà l'idea descritta nella sezione 3.1 che sarà di supporto alla progettazione e sviluppo del Software ed alla sua comprensione ed il suo adattamento a nuove applicazioni. [24], [36]

Requisiti Funzionali

Uno dei documenti che è necessario produrre se si vuole intraprendere l'utilizzo del Processo Unificato è l'Analisi dei Requisiti Funzionali. I Requisiti di un Software possono infatti essere suddivisi in:

- **Funzionali:** che riguardano cioè le funzionalità che si vuole implementare all'interno di un Software. **COSA**
- **Non Funzionali:** che riguardano cioè le performances di funzionamento del Software. **COME**

Per avere una panoramica sull’interfaccia del Protocollo DATEX II che dovrà essere implementata nel sistema, si analizzino ora i requisiti funzionali del sistema descritto nella sezione 3.1, che faciliteranno lo sviluppo Software e documentaremo i passi e le scelte progettuali effettuate durante il processo.

In accordo con il modello UP, il documento che verrà redatto in questa sezione non si intende completo ma è solo una prima visione generale del progetto.

Requisito	Descrizione
R1	Il sistema dovrà essere implementato su smartphone e tablet Android, garantendo un supporto anche alle versioni pregresse del sistema operativo
R2	Il sistema dovrà richiedere l’accesso alle letture dei sensori ed alle relative specifiche tecniche. Si dovrà inoltre poter accedere alla posizione del dispositivo ed alla rete Internet per la comunicazione con API esterne e con il Middleware.
R3	Ove necessario, il sistema deve rielaborare le letture provenienti dai sensori per eseguire un filtraggio.
R4	Il sistema deve essere in grado di supportare le letture provenienti da tutti i sensori attualmente integrati negli smartphone.
R5	Il sistema deve offrire un metodo rapido ed efficiente per l’aggiunta di nuovi sensori per le future evoluzioni degli smartphone.
R6	Il sistema dovrà verificare le specifiche tecniche di tutti i sensori integrati nel dispositivo ed utilizzare le letture dei soli ritenuti affidabili, accurati e conformi alle necessità dell’algoritmo.
R7	Deve essere data la possibilità a coloro che usano il sistema di selezionare le letture dei sensori che si intende condividere con la piattaforma di Crowdsourcing.
R8	Deve essere data la possibilità a coloro che usano il sistema di inviare le letture dei sensori che sono stati selezionati manualmente.
R9	Deve essere data la possibilità a coloro che usano il sistema di segnalare l’occorrenza di particolari eventi lungo la rete stradale (incidenti, ostruzioni, rallentamenti, condizioni climatiche).
R10	All’avviamento, il sistema deve consentire di visualizzare su una mappa gli eventi sul traffico condivisi da altri dispositivi.
R11	Il sistema deve consentire di selezionare l’area di interesse nella quale visualizzare gli eventi sul traffico condivisi da altri dispositivi.
R12	Il sistema deve interrogare il Database all’interno del quale sono salvati tutti gli eventi per recuperare quelli necessari.
R13	Deve essere implementato un Database che contenga le segnalazioni sul traffico inviate dai vari dispositivi in maniera geo-referenziata.
R14	Il Database deve consentire l’elaborazione di query geo-referenziate da parte dell’utenza.

R15	Il Database deve essere progettato per ottimizzare la risposta alle query geo-referenziate.
R16	Il sistema deve prevedere l'utilizzo di un Middleware IoT che riceva i dati da smartphone e possa condividerli con dei Database per future analisi di dati e query degli stessi.
R17	Il sistema deve stabilire una connessione di tipo MQTT o HTTP con un Middleware per la raccolta dei dati.
R18	Il sistema deve stabilire una comunicazione di tipo publish/subscriber tra dispositivi Smartphone e Middleware IoT.
R19	Una volta stabilita la comunicazione con il Middleware, il dispositivo Android deve inviare le letture dei sensori di cui è dotato a cadenza periodica oppure al click di un bottone da parte dell'utente.
R20	Il sistema dovrà prevedere l'implementazione del protocollo DATEX II per le sole componenti ritenute necessarie.
R21	Il sistema dovrà formattare ed interpretare i dati in formato XML in accordo con lo standard DATEX II.
R22	Il sistema dovrà inviare i dati formattati secondo il protocollo DATEX II in formato JSON al middleware IoT.
R23	Il sistema dovrà consentire la registrazione di un nuovo utente attraverso i suoi dati personali ed associare di conseguenza un token al particolare utente per l'accesso alle prestazioni di API esterne, del Middleware IoT e dell'accesso al Database.

Tabella 3.2: Analisi dei requisiti funzionali seguendo il modello di sviluppo UP

Diagramma delle Classi

I requisiti visti nella sezione precedente vanno ora confrontati con ciò che viene offerto dallo standard DATEX II in modo da implementare la sola interfaccia necessaria a soddisfarne i requisiti identificati del sistema. La Commissione responsabile dello sviluppo dell'interfaccia DATEX II ha messo a disposizione il modello delle classi implementato nello standard al quale fare riferimento per le implementazioni di DATEX II. http://d2docs.ndwcloud.nu/_static/umlmodel/v3.0/index.htm

Il Diagramma delle Classi qui presentato è un altro documento previsto dal modello di sviluppo del Processo Unificato per la documentazione dello sviluppo software. In questo documento, sono contenute le classi e le relazioni tra di esse e serve ad illustrare la struttura del software. Tale documento, tuttavia, serve soltanto ad offrire una panoramica statica delle classi che compongono il Software, senza considerare cioè le singole implementazioni delle stesse.

Si intende pertanto in questa sezione, analizzando il diagramma delle classi dello standard DATEX II, implementare il diagramma delle classi (o una parte di esso) del sistema che implementi lo standard per adempiere ai requisiti di Tabella 3.2.

Essendo la definizione di un protocollo di comunicazione DATEX II definisce anche il

significato di tutto ciò che viene trasmesso all'interno del Payload di un pacchetto scambiato tra due dispositivi che partecipano alla comunicazione. La definizione del payload Figura 3.14, può estendere a sua volta anche le definizioni di nuove classi o attributi Figura 3.13, purchè rispettino particolari vincoli come indicato nella sottosezione 3.3.1 Dalla

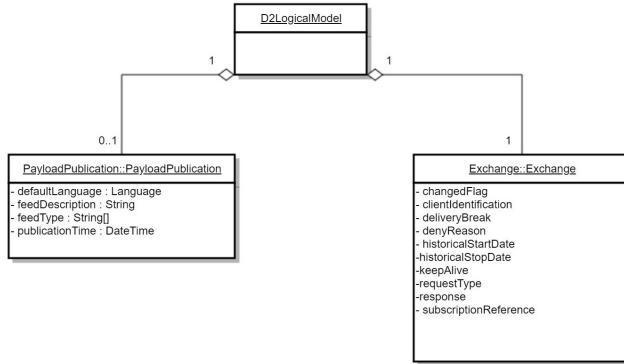


Figura 3.12: Distinzione effettuata dallo Standard DATEX II tra metodo di comunicazione (*Exchange*) e formato del Payload (*PayloadPublication*)

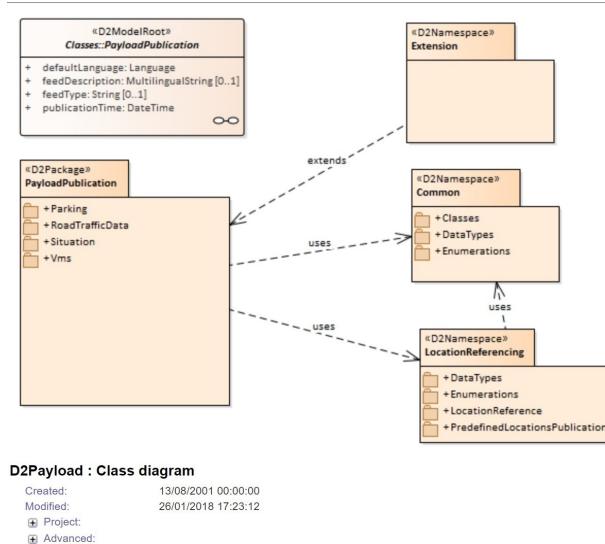


Figura 3.13: Formato del Payload

Figura 3.14 è evidente che la classe PayloadPublication sia solo una classe astratta o una interfaccia di classe che viene implementata da altre classi che definiscono tipologie di classi specializzate. Infatti, a conferma di quanto mostrato nella sottosezione 3.3.5, le Publications possono essere di varie tipologie specializzate a seconda dei dati che trattano. In

3 – Progetto del Sistema

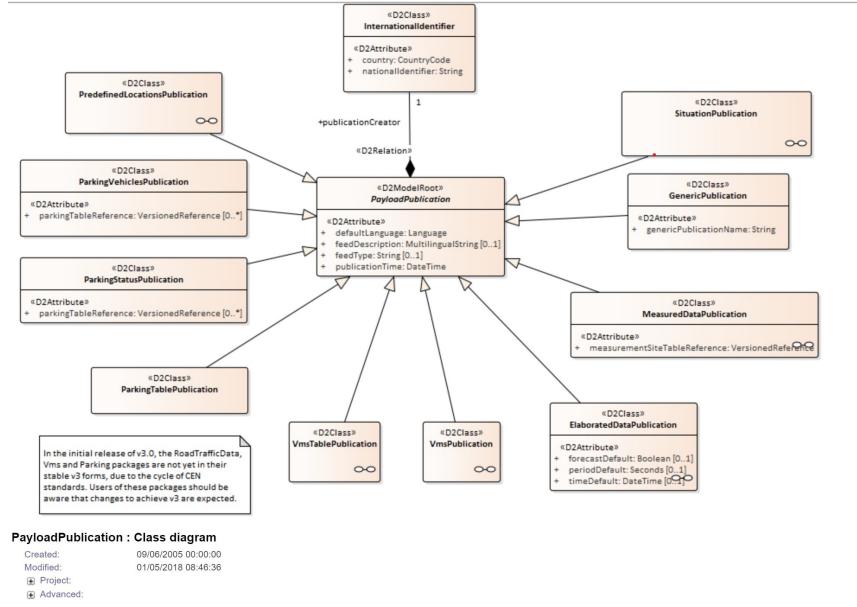


Figura 3.14: Formato della Publication

funzione dei Requisiti Funzionali in Tabella 3.2 e dell’idea del sistema nella sezione 3.1, è possibile già filtrare ad alto livello le sole tipologie di publication che è necessario implementare nel sistema. Un approccio del tutto simile è stato seguito quindi per tutte

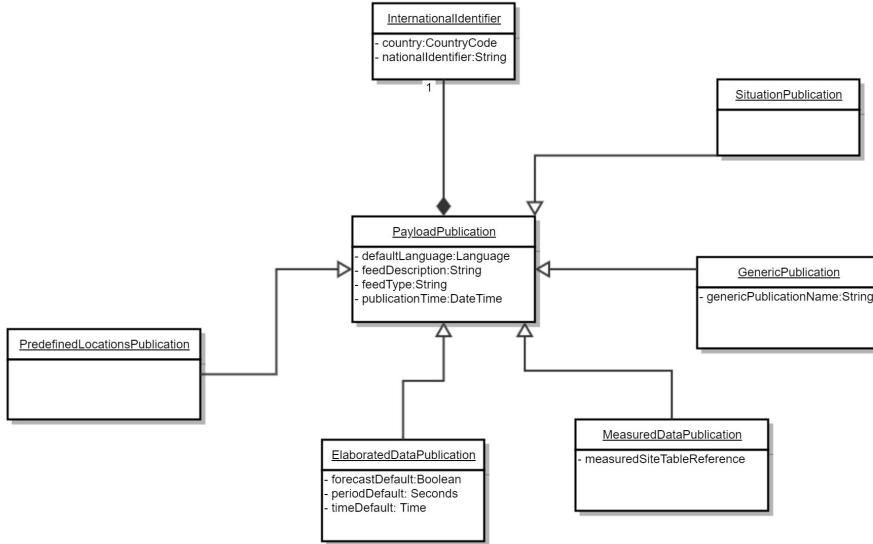


Figura 3.15: Implementazione della Publication

le tipologie di Publication che si è ritenuto opportuno implementare. Ovvero, sulla base del diagramma delle classi del protocollo DATEX II, sono state filtrate le sole classi ed attributi necessari a soddisfare i requisiti di Tabella 3.2. Il risultato di questo processo è mostrato nell' Appendice A.

3.4 IoT Cloud Platform

Nella sezione precedente è stato mostrato nei dettagli come procedere all'implementazione del Protocollo DATEX II in un qualsiasi linguaggio di programmazione orientato agli oggetti. In questo modo, si è giunti ad una fase della progettazione del software nella quale i dispositivi che eseguono il sistema, riescono a comunicare tra loro seguendo un Protocollo comune. Sebbene non sia ancora stato definito completamente il contenuto della comunicazione tra due dispositivi (che sarà compito del Capitolo 4), è necessario ora progettare una struttura che sia in grado di ricevere questo messaggio, indipendentemente da cosa esso contenga e di trattarlo opportunamente.

E' cioè necessario individuare una Piattaforma Cloud che sia in grado di ricevere i dati provenienti da dispositivi IoT e che sia in grado di conservarli all'interno di opportuni Database, la cui struttura sarà descritta nella sezione 3.5.

Dal momento che il panorama attuale offre numerosi servizi che possano assolvere a questo

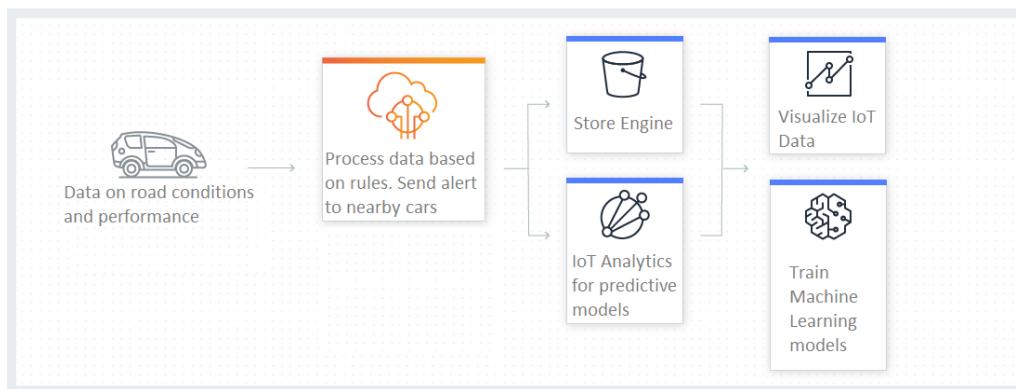


Figura 3.16: Schema della componente che si vuole implementare in questa sezione vista nel progetto completo

compito, si valutino ora i requisiti che il Middleware IoT deve avere per l'implementazione del sistema descritto nella sezione 3.1 e, sulla base di questi requisiti, si effettuerà un confronto tra le possibili soluzioni.

I requisiti mostrati nella Tabella 3.3 vanno ad aggiungersi a quelli elencati nella Tabella 3.2 concludendo in questo modo la redazione dell' Analisi dei Requisiti prevista nel Processo Unificato.

Requisito	Descrizione
M1	Il Middleware IoT dovrà prevedere un sistema per la registrazione di nuovi utenti
M2	Il Middleware IoT dovrà consentire l'accesso ai soli utenti registrati, instaurando con essi una connessione sicura
M3	Il Middleware IoT dovrà consentire di assegnare diversi ruoli a diversi utenti con diverse permission
M4	Il Middleware IoT deve trattare diversamente i dati provenienti da diverse località geografiche
M5	Il Middleware IoT deve stabilire una comunicazione con un database di tipo SQL/NoSQL per il salvataggio dei dati
M6	Il Middleware IoT dovrà stabilire una comunicazione di tipo publisher/subscriber con uno o più dispositivi IoT
M7	Il Middleware IoT dovrà essere altamente scalabile per supportare un numero sempre crescente di dispositivi
M8	Il Middleware IoT dovrà consentire la comunicazione con sistemi per la visualizzazione ed analisi dei dati
M9	Il Middleware IoT dovrà implementare il protocollo DATEX II
M10	Il Middleware IoT dovrà supportare diversi protocolli per la comunicazione (MQTT, HTTP, AMQP, etc) per garantire la comunicazione con dispositivi eterogenei

Tabella 3.3: Analisi dei requisiti che la IoT Cloud Platform deve supportare

Le piattaforme che sono state individuate come possibili Cloud Platforms adatte allo scopo sono:

- **AWS IoT Core:** <https://aws.amazon.com/it/iot-core/>
- **Azure IoT Lab:** <https://docs.microsoft.com/en-us/azure/iot-hub/about-iot-hub>
- **FIWARE:** <https://www.fiware.org/about-us/>

Le soluzioni identificate, non rappresentano assolutamente l'insieme di tutte le soluzioni possibili per le Cloud Platform, ma solo di quelle ritenute più adatte allo scopo della progettazione del sistema.

Di seguito viene proposto un confronto tra le diverse soluzioni elencate sulla base dei requisiti di Tabella 3.3. Le informazioni sulla base delle quali è stato effettuato questo confronto sono state raccolte dai siti Web di ciascuna soluzione e si intendono riferiti alla data della seduta di laurea alla quale questo elaborato si riferisce.

Si noti come, la soluzione proposta da Fiware sia particolarmente interessante in quanto offre una piattaforma open source che quindi può essere adattata alle particolari necessità. Tale soluzione, può essere ritenuta infatti molto promettente per quelle compagnie che vogliono progettare su misura la Cloud Platform ed adattarla alle proprie necessità in

Requisito	AWS IoT Core	Azure IoT Lab	Fiware
M1	✓	✓	✓
M2	✓	✓	✓
M3	✓	✓	✗
M4	✓	✗	✓
M5	✓	✓	Not Implemented
M6	✓	✓	Not Implemented
M7	✓	✓	✓
M8	✓	✓	Not Implemented
M9	✓	✓	✓
M10	✓	✓	✓

Tabella 3.4: Confronto tra Cloud Platforms

modo da avere dei migliori rendimenti prestazionali. Infatti, una soluzione personalizzata, potrebbe offrire prestazioni migliori di soluzioni generali come quelle offerte da AWS ed Azure.

Tuttavia, per il dominio di questo lavoro di tesi, la soluzione ideale è quella di sfruttare piattaforme cloud generiche per la raccolta dei dati. In questo modo, si godrà di un ecosistema di servizi ampiamente documentato e ben interfacciato. Infatti, ecosistemi come AWS ed Azure offrono diversi servizi e diverse applicazioni anche al di fuori del dominio dell’IoT. Queste possono essere ad esempio relative ad analisi dei dati, raccolta e storage dei dati, servizi di autenticazione e piattaforme di calcolo e machine learning. Tutti questi servizi, sono facilmente utilizzabili e possono facilmente comunicare tra di loro scambiandosi in maniera automatica i dati di cui necessitano. In piattaforme open source, invece, è necessario sviluppare ed interfacciare ciascuno di questi servizi, richiedendo di fatto una mole di lavoro estremamente maggiore.

Questa scelta è anche da considerarsi adatta per coloro i quali si interfacciano per la prima volta con una piattaforma cloud in quanto numerosi sono i corsi e le documentazioni offerte da AWS ed Azure.

Considerato infine che le funzioni offerte da AWS ben soddisfano i requisiti della Tabela 3.3, si è scelto di utilizzare questa piattaforma come Middleware IoT che si occuperà della raccolta, condivisione e salvataggio dei dati.

3.4.1 AWS IoT Core Overview

Amazon Web Services IoT Core è una piattaforma cloud che consente a dispositivi connessi di interagire tra loro e con altre applicazioni nel cloud. Inoltre, IoT Core, si interfaccia con tutti gli altri servizi offerti da AWS per il cloud computing and analysis. Ad esempio sarà semplice interfacciare i dati raccolti da IoT core con altri servizi quali:

- **Amazon S3** per la memorizzazione dei dati in Database SQL
- **Amazon DynamoDB** per la memorizzazione dei dati in Database NoSQL

- **Amazon Kinesis ed Amazon QuickSight** per la visualizzazione ed analisi dei dati
- **TensorFlow in AWS** per l'elaborazione dei dati raccolti con tecniche di Machine Learning

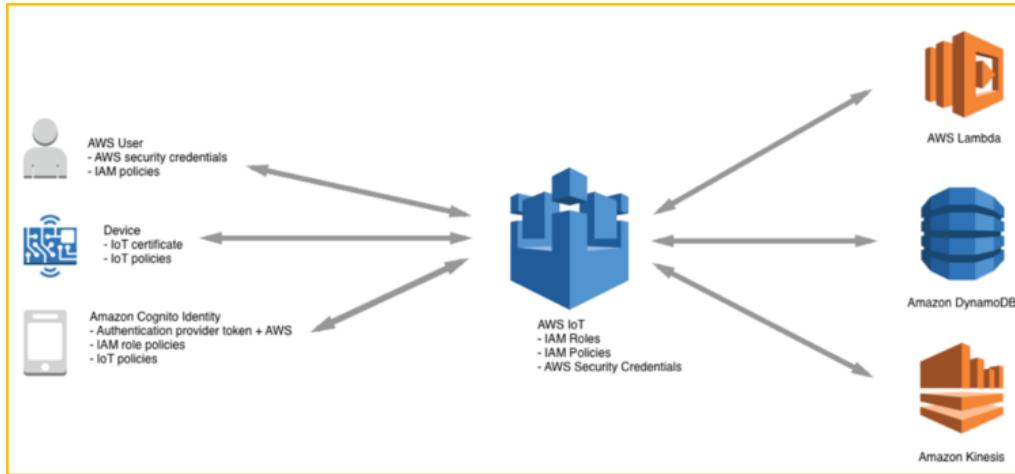


Figura 3.17: Capacità di interfacciamento della piattaforma IoT Core con gli altri servizi dell'ecosistema AWS

Alcune tra le comuni applicazioni IoT raccolgono ed elaborano dati di telemetria dai dispositivi oppure permettono agli utenti di controllare un dispositivo in remoto. I dispositivi segnalano il proprio stato pubblicando messaggi, in formato JSON, in argomenti MQTT. Ogni argomento MQTT ha un nome gerarchico che identifica il dispositivo il cui stato è in fase di aggiornamento. Quando vengono pubblicati in un argomento MQTT, i messaggi vengono inviati al broker di messaggi MQTT AWS IoT, responsabile dell'invio di tutti i messaggi pubblicati in un argomento MQTT a tutti i client che hanno sottoscritto l'argomento.

La comunicazione tra un dispositivo e AWS IoT è protetta dai certificati X.509.

AWS IoT è in grado di generare un certificato per l'accesso in maniera automatica oppure creato manualmente. In entrambi i casi, il certificato deve essere registrato e attivato su AWS IoT e quindi copiato nel dispositivo. Quando il dispositivo comunica con AWS IoT, presenta il certificato a AWS IoT come credenziale.

Authentication

AWS IoT Core offre autenticazione reciproca e crittografia in tutti i punti di connessione, perciò non si verificherà mai alcuno scambio di dati tra dispositivi e AWS IoT Core senza accertamenti di identità. AWS IoT Core supporta il metodo di autenticazione di AWS (denominato "SigV4"), l'autenticazione basata su certificato X.509 e l'autenticazione

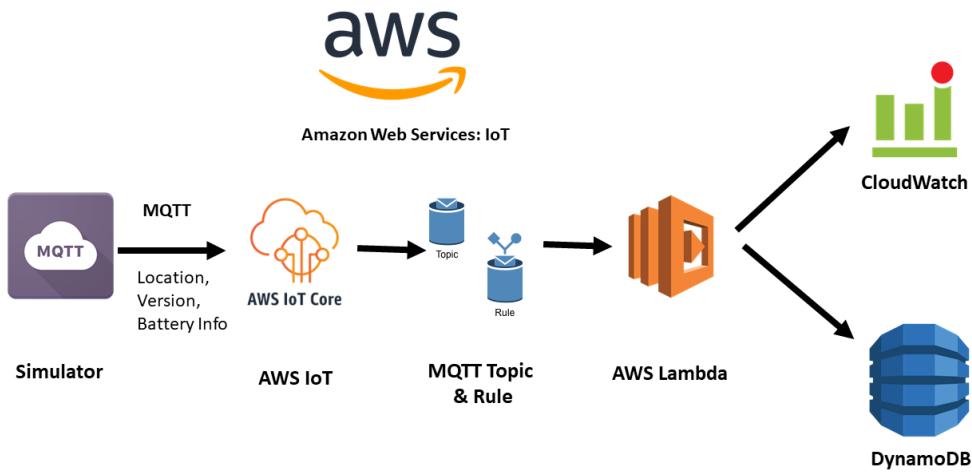


Figura 3.18: Esempi di utilizzo della piattaforma IoT Core combinata con altri servizi dell'ecosistema AWS

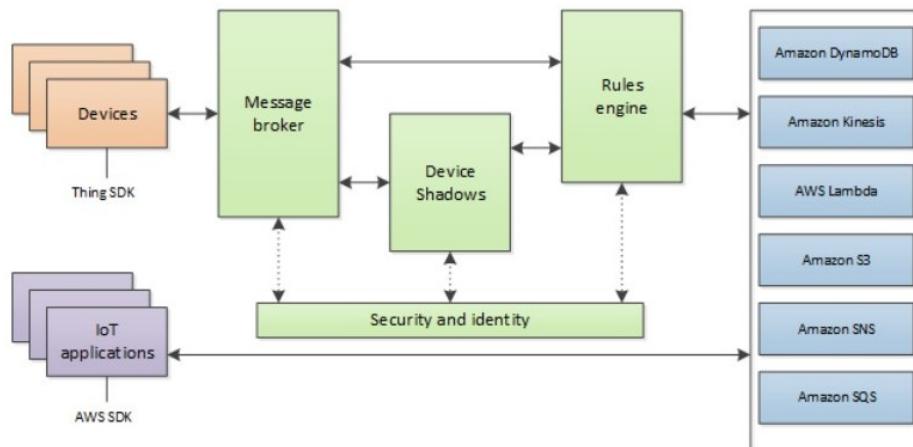


Figura 3.19: Struttura della comunicazione

basata su token creati dal cliente (mediante autorizzazioni ad hoc). Le connessioni che impiegano il protocollo HTTP possono usare tutti i metodi disponibili, mentre le connessioni MQTT usano l'autenticazione basata su certificato e le connessioni WebSockets possono usare SigV4 o autorizzazioni ad hoc. È possibile mappare policy per ciascun certificato per autorizzare l'accesso di dispositivi e applicazioni in qualsiasi momento

senza toccare il dispositivo.

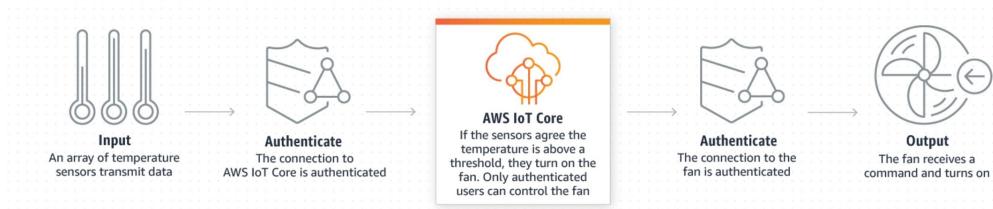


Figura 3.20: Processo di autenticazione implementato in ogni punto della connessione

È possibile creare, distribuire e gestire certificati e policy per i dispositivi tramite la console o l'API. I certificati dei dispositivi possono essere assegnati, attivati e associati con le policy IoT corrispondenti configurate tramite AWS IoT Core. In questo modo sarà possibile revocare istantaneamente l'accesso per singoli dispositivi. AWS IoT Core supporta inoltre le connessioni dalle app per dispositivi mobili degli utenti tramite Amazon Cognito, che crea un identificatore univoco e recupera credenziali temporanee di AWS con privilegi limitati. AWS IoT Core fornisce inoltre credenziali AWS temporanee quando un dispositivo si autentica con un certificato X.509, per facilitare al dispositivo l'accesso ad altri servizi AWS, ad esempio DynamoDB o S3.

Shadow dei Dispositivi

Con AWS IoT Core è possibile creare versioni virtuali e persistenti dei dispositivi, chiamate "shadow" dei dispositivi (copia digitale del dispositivo), che includono l'ultimo stato noto del dispositivo, consentendo ad applicazioni e altri dispositivi di leggerne i messaggi e interagire con esso. La shadow dei dispositivi conserva l'ultimo stato noto e lo stato futuro impostato per ciascun dispositivo anche quando è offline. Per recuperare l'ultimo stato noto di un dispositivo o impostare uno stato futuro, è possibile usare l'API o il motore di regole.

La shadow dei dispositivi semplifica la creazione di applicazioni che interagiscono con i dispositivi perché fornisce sempre API REST disponibili. Inoltre, le applicazioni possono impostare uno stato futuro per un dispositivo indipendentemente dal suo stato corrente. AWS IoT Core confronterà lo stato futuro con l'ultimo stato noto e imporrà al dispositivo di aggiornarsi.

3.4.2 AWS IoT Core Development

Dopo aver visto le main features della cloud platform AWS IoT Core, si progettino ora gli step implementativi da seguire al fine di integrare correttamente il sistema progettato nelle precedenti sezioni con la cloud platform. A tal scopo, sarà necessario seguire la documentazione messa a disposizione da AWS IoT Core per ottenere delle linee guida più dettagliate sui passi da seguire https://docs.aws.amazon.com/it_it/iot/latest/developerguide/iot-dg.pdf.

Gli step implementativi progettati in questa sezione si intendono riferiti ad una qualsiasi implementazione della cloud platform in un qualsiasi linguaggio di programmazione e per un qualsiasi progetto. I dettagli implementativi relativi al software sviluppato in questo lavoro di tesi, saranno invece mostrati nel Capitolo 4.

1. Accesso alla Console AWS IoT Core e creazione di un nuovo account
2. Registrazione e Configurazione di un nuovo dispositivo come copia digitale (o shadow) di un dispositivo reale
3. Configurazione della comunicazione con il protocollo MQTT
4. Utilizzo dell' SDK messa a disposizione da AWS per l'abilitazione dei dispositivi IoT alla comunicazione con il Broker centrale.

Accesso alla Console AWS IoT Core

Al fine di poter utilizzare i servizi offerti da AWS è necessario prima seguire un processo di registrazione di un nuovo account presso <https://aws.amazon.com/it/> e quindi seguire la procedura guidata per la Creazione di un Nuovo Account.
Una volta completata la procedura di registrazione è quindi possibile accedere alla piattaforma IoT Core ed iniziare a creare un nuovo dispositivo.

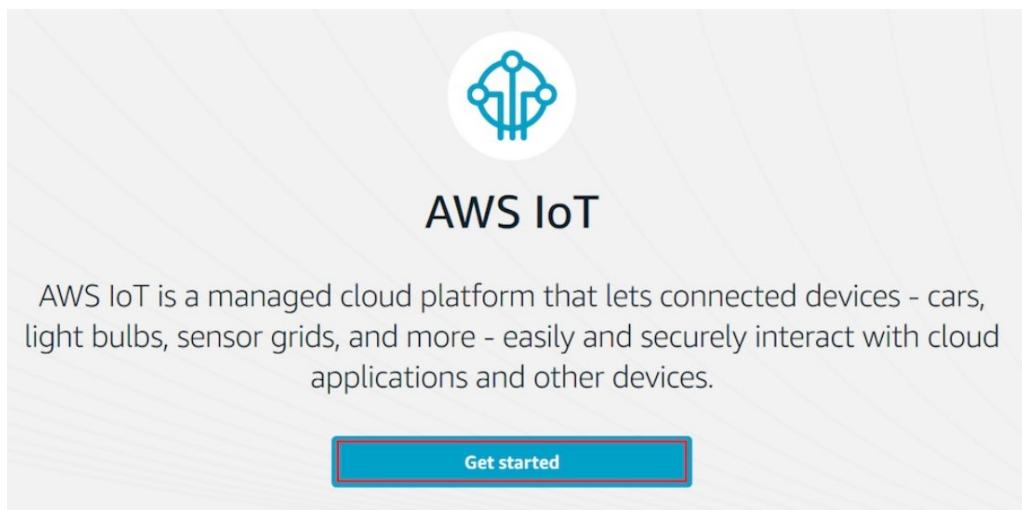


Figura 3.21: Primo accesso alla piattaforma IoT Core

Registrazione e Configurazione di un nuovo dispositivo

I dispositivi connessi a AWS IoT sono rappresentati da oggetti IoT nel registro AWS IoT. Questo registro permette di tenere traccia di un record di tutti i dispositivi registrati nell'account AWS IoT in modo da poter assegnare a ciascun dispositivo diversi ruoli e quindi diverse permission.

Si rende quindi necessario creare ora una copia digitale di ogni dispositivo abilitato alla comunicazione con AWS IoT Core. In realtà, per evitare di creare un numero infinitamente grande di dispositivi diversi, un'approccio più adatto è quello di creare delle macro-categorie di oggetti che possono collegarsi alla piattaforma cloud.

Volendo, in questo lavoro di tesi, abilitare la comunicazione tra i soli smartphone con sistema operativo Android ed il Middleware IoT, si crei un unico oggetto denominato **Android_Device** che si riferisca quindi a tutti i dispositivi Android che si collegano al Middleware. Allo stesso modo, è possibile creare una qualsiasi categoria di oggetti come ad esempio: **iOS_Device**, **Windows_Device**, **VMS_Device**, etc.

La procedura per la creazione di un nuovo oggetto è descritta nei dettagli in https://docs.aws.amazon.com/it_it/iot/latest/developerguide/iot-dg.pdf ed il risultato finale sarà la comparsa di un nuovo prototipo di oggetto *shadow* all'interno del IoT Core. Una volta creato, il nuovo dispositivo ha bisogno che vi siano assegnate delle perimis-

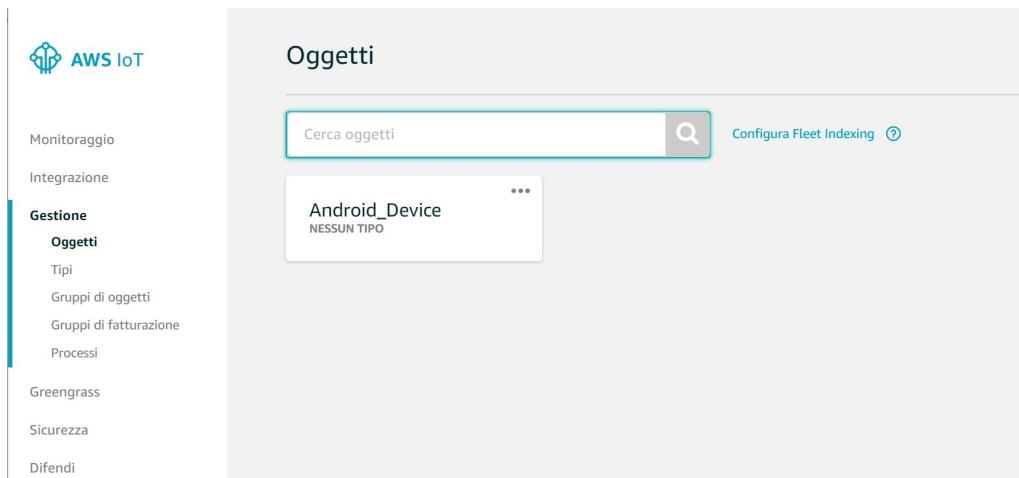


Figura 3.22: Creazione di un nuovo oggetto nella pagina principale di AWS IoT Core.

sion e quindi una Policy per poter comunicare con la Cloud Platform. AWS prevede infatti la possibilità di poter assegnare diverse Policy a diversi dispositivi che quindi avranno accesso a diverse componenti e servizi della Cloud Platform. Questo è fatto per garantire alti livelli di sicurezza e di gestione degli accessi a particolari servizi. Ad esempio, in questo modo potrebbero essere offerte alcune funzionalità a tutti i dispositivi, ed altre funzionalità premium ai soli dispositivi con un diverso piano tariffario.

Per lo scope di questo lavoro di tesi, la Policy che sarà assegnata al dispositivo appena creato (**Android_Device**), sarà quella che gli consentirà di eseguire tutte le operazioni

AWS IoT su tutte le risorse AWS IoT. Queste impostazioni sono adatte ad un ambiente di sviluppo per facilitare il debugging ma sono ovviamente eccessivamente permissive ed è quindi opportuno restringere l'ambito delle autorizzazioni a quelle richieste dal dispositivo in un ambiente di produzione.

Anche in questo caso, per evitare ridondanze, si rimanda alla procedura per la creazione ed assegnazione di una nuova Policy ad un oggetto shadow creato, si rimanda alla documentazione ufficiale.

Configurazione della Comunicazione

A questo punto della configurazione, è possibile testare la comunicazione tra il Client (oggetto appena creato ed al quale è stata associata una Policy) ed il Server (IoT Core).

AWS IoT Core, consente inoltre di stabilire la comunicazione Client / Server seguendo diversi protocolli di comunicazione tra i quali MQTT ed HTTP che possono essere interscambiati a piacere.

In riferimento al Capitolo 1 dove si era fatto un confronto tra i due protocolli e sui vantaggi dell'utilizzo di uno piuttosto che dell'altro, era evidente come il protocollo MQTT fosse più efficiente in termini di consumo energetico e nella capacità computazionale richiesta. Pertanto, vista la frammentazione del mondo Android in termini di versioni del sistema operativo implementate ed anche in termini di obsolescenza dei dispositivi, si ritiene più adatto l'utilizzo del protocollo MQTT per stabilire una comunicazione tra il Client Android ed il server AWS.

Si testi quindi ora la comunicazione di tipo Publish - Subscribe tra il Client ed il Server

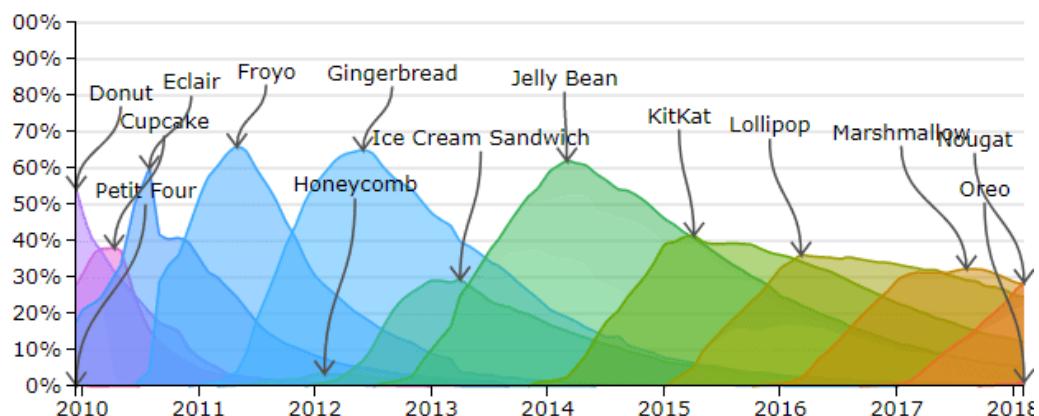


Figura 3.23: Panorama della frammentazione nella distribuzione delle versioni del sistema operativo Android

(sezione 1.2.3).

Un dispositivo pubblicherà dei messaggi all'interno di un Topic ed è possibile impostare il Server IoT Core in modo da essere in ascolto sullo stesso Topic, caratterizzato cioè dallo stesso nome.

3 – Progetto del Sistema

Nella sezione Test della console IoT Core, sono presenti due sottosezioni : **Pubblicare** e **Sottoscrivere**. Nella sezione **Sottoscrivere**, va specificato il nome del Topic al quale

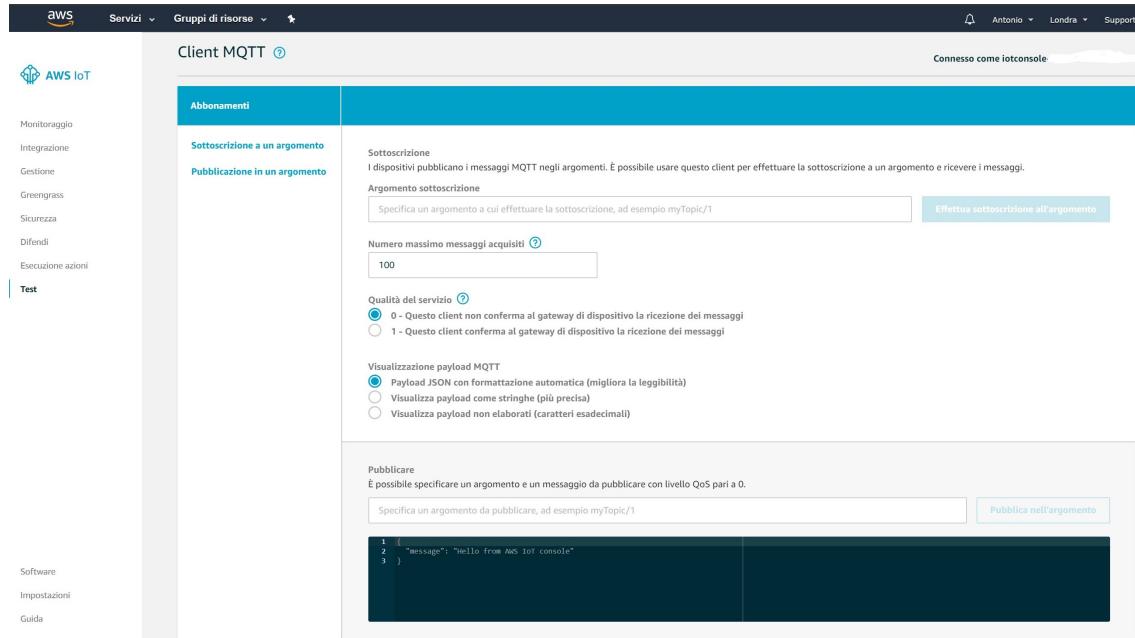


Figura 3.24: Panoramica della schermata Test

registrarsi o meglio sul quale restare in ascolto. Questo Topic sarà caratterizzato dallo stesso nome del Topic sul quale il dispositivo comunicherà o meglio pubblicherà. Infine, per



Figura 3.25: Sottoscrizione al Topic EU_WEST_2

simulare il comportamento di un dispositivo IoT che pubblichi un messaggio all'interno di un argomento, si pubblicherà un messaggio nella sezione **Pubblicare** che faccia riferimento allo stesso topic sul quale si è in ascolto. Dopo aver pubblicato il messaggio, nella sezione



Figura 3.26: Pubblicazione nel Topic **EU_WEST_2**

ne **Sottoscrivere** comparirà il messaggio appena ricevuto. In maniera del tutto analoga,

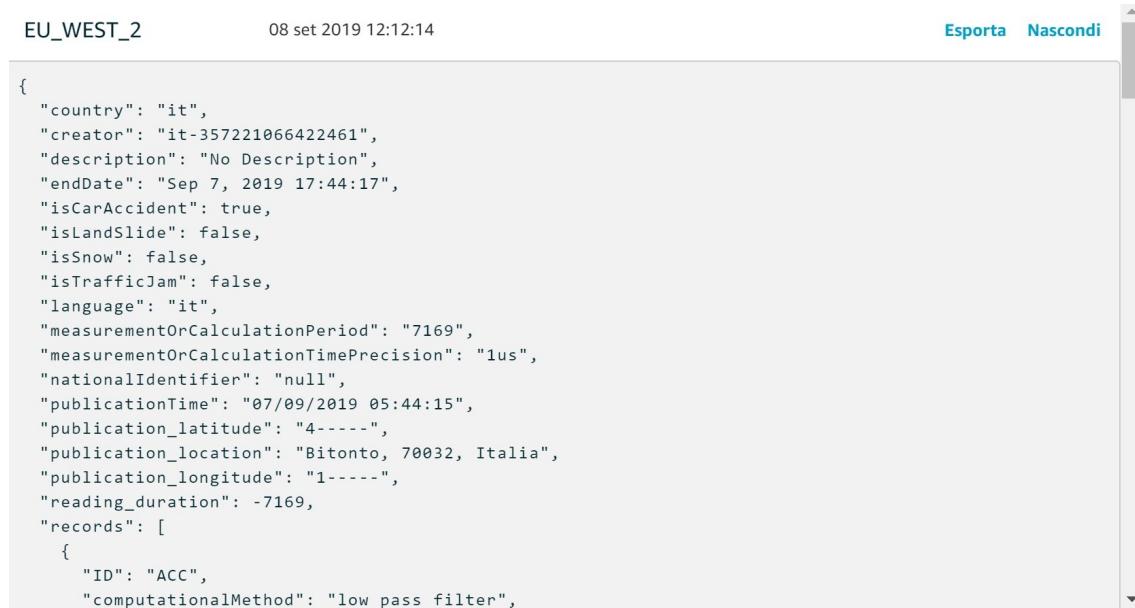


Figura 3.27: Ricezione del messaggio

avverrà anche la comunicazione tra il Server IoT Core ed il Client Android.

Si noti come in Figura 3.25, Figura 3.26 e Figura 3.27 si è utilizzato come nome per il topic della pubblicazione **EU_WEST_2**. In realtà avremmo potuto ottenere lo stesso risultato e quindi stabilire la comunicazione tra client e server anche utilizzando un qualsiasi

altro nome del topic, purchè il nome sul quale il Client pubblica i messaggi sia lo stesso sul quale il Server è in ascolto. Il motivo per cui sarà utilizzato il topic **EU_WEST_2** verrà spiegato nella sezione 4.2.

Utilizzo della SDK

Gli SDK (Software Development Kit) messi a disposizione dal team di sviluppo di AWS aiutano a connettere i dispositivi ad IoT Core in modo rapido e semplice. Gli SDK di AWS IoT includono librerie open source, linee guida per gli sviluppatori con esempi e guide alla portabilità, con cui è possibile creare soluzioni o prodotti IoT innovativi qualsiasi piattaforma hardware.

Gli SDK messi a disposizione dal team di AWS sono riferiti a specifiche architetture hardware e specifici linguaggi di programmazione e pertanto fuori dallo scope di questo capitolo che vuole invece fornire le linee guida generali di progettazione, indipendentemente dall'hardware e dal linguaggio utilizzato. Pertanto, nel Capitolo 4 saranno mostrati i dettagli implementativi dell' SDK per i dispositivi Android utilizzando il linguaggio di programmazione JAVA.

Qualora si vogliano ottenere i dettagli implementativi per altre piattaforme, si faccia riferimento alla documentazione ufficiale corredata anche da esempi applicativi (https://docs.aws.amazon.com/it_it/iot/latest/developerguide/iot-dg.pdf).

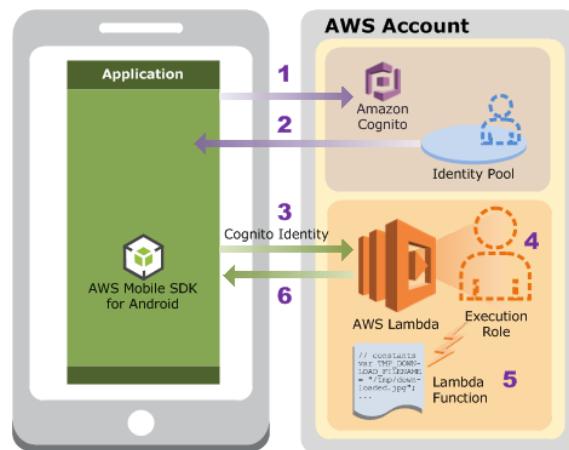


Figura 3.28: Esempio dell'utilizzo dell'SDK messo a disposizione da AWS in Android per lo sviluppo di Lambda Application

3.5 Database Architecture

In conclusione, l'ultima componente che è necessario progettare è il Database che si occuperà di raccogliere i dati provenienti da tutti i dispositivi IoT e transitanti attraverso la piattaforma cloud IoT Core. Il Database svolgerà un ruolo fondamentale in quanto non solo

sarà il sistema di raccolta di tutti i dati provenienti da i dispositivi ma esso si occuperà anche di esporre questi stessi dati ad altre applicazioni che gli analizzeranno ed elaboreranno per trarne delle considerazioni e previsioni. Ulteriormente, i Database esporranno i dati in essi contenuti anche ad altri dispositivi IoT che dovessero interrogarli per ottenere informazioni sullo stato del traffico o in generale di eventi stradali.

Dal momento che la piattaforma Cloud che è stata scelta nella sezione precedente appartiene all'ecosistema AWS, è opportuno utilizzare un servizio di storage dei dati che appartenga allo stesso ecosistema in modo da sfruttare l'ampia documentazione e la facilità di interfacciamento dei due sistemi appartenenti allo stesso ecosistema. In tal caso, AWS mette a disposizione due servizi per lo storage dei dati:

- **Amazon RDS:** Servizio per l'utilizzo di Database Relazionali (SQL)
- **Amazon DynamoDB:** Servizio per l'utilizzo di Database Non Relazionali (No-SQL)

La differenza sostanziale tra i due servizi consiste quindi nella architettura di Database nella quale si vogliono raccogliere i dati.

Si evidenzino quindi i requisiti che il Database deve soddisfare per il Software che si sta progettando in modo da vedere quali delle due servizi offerti da AWS sia più consono al progetto.

Requisito	Descrizione
D1	Il Database dovrà essere altamente scalabile per sopportare ad un numero di dispositivi e di entry nel database molto elevato
D2	Il Database dovrà essere altamente performante per rispondere in breve tempo alle query
D3	Il Database dovrà supportare l'invio di query geo-referenziate
D4	Il Database dovrà supportare e memorizzare diverse tipologie di dati provenienti da diversi dispositivi
D5	Il Database dovrà essere progettato considerando il suo futuro utilizzo per la Big Data Analysis

Tabella 3.5: Analisi dei requisiti che la IoT Cloud Platform deve supportare

Per capire quindi quale soluzione meglio si adatta ai requisiti del progetto in fase di sviluppo, è necessario effettuare una breve digressione sulle differenze tra le due architetture di un Database in modo da averne chiari i punti di forza ed i punti deboli di ciascuna architettura.

3.5.1 Database Relazionali e Non Relazionali

I Database Relazionali o **RDBMS** memorizzano i dati in tuple o record ovvero salvano i dati all'interno di tabelle. Ciascuna tabella ha un numero fisso di colonne e un numero variabile di righe (o record). Le colonne definiscono i dati degli attributi delle entità di una data tabella. Ad esempio, una tabella clienti avrà come colonne nome, cognome, indirizzo, telefono, e così via. Ogni riga nella tabella definisce un elemento effettivo costituito da valori per tutte le colonne.

Le tabelle hanno colonne speciali chiamate chiavi primarie (primary key), che contengono identificatori unici per ogni riga di una tabella. Le tabelle possono anche avere delle colonne chiamate chiavi esterne (foreign key), che contengono un riferimento chiave primaria di una riga su un'altra tabella (o anche se stessa). Questi collegamenti tra righe sono chiamate associazioni (o relazioni, dall'inglese relationships) e sono la base del modello relazionale di un database.

I database relazionali trovano spazio d'applicazione quando si ha a che fare con dati strutturati,

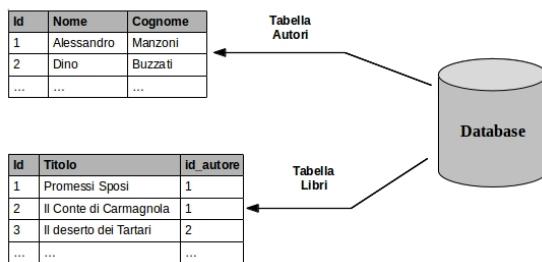


Figura 3.29: Esempio di un database relazionale

turati, ovvero quando è facile creare una rappresentazione su tabella semplice e lineare, con un numero ridotto di relazioni. Inoltre, gli SQL sono praticamente obbligatori ogni volta che dobbiamo garantire integrità sui dati e le proprietà ACID in senso lato.

I Database Non Relazionali o **NRDBMS** utilizzano molteplici modelli di dati per accedere e gestire i dati, quali documento, grafo, chiave-valore, in memoria e ricerca. Questi tipi di database sono ottimizzati specificatamente per applicazioni che necessitano di grandi volumi di dati, latenza bassa e modelli di dati flessibili, ottenuti snellendo alcuni dei criteri di coerenza dei dati degli altri database. I database NoSQL sono una soluzione ideale per molte applicazioni moderne, quali dispositivi mobili, Web e videogiochi che richiedono database flessibili, scalabili, con prestazioni elevate ed altamente funzionali per offrire un'esperienza utente eccezionale.

I NoSQL sono estremamente versatili e quindi indicati in quelle situazioni in cui dobbiamo modellare il polimorfismo. Se si hanno entità tra loro tutte simili e che grossomodo portano la stessa informazione, ma con piccole differenze l'una dall'altra, con NoSQL il problema si risolve qualche manciata di minuti. In SQL, invece, è necessaria una soluzione decisamente più complessa. NoSQL è quindi ideale nel caso di dati abbastanza scorrelati e

che tendono ad evolvere nel tempo; l'assenza di una struttura fissa permette di aggiungere nuovi tipi di dati (o modificare gli esistenti) senza dover mettere mano al database.

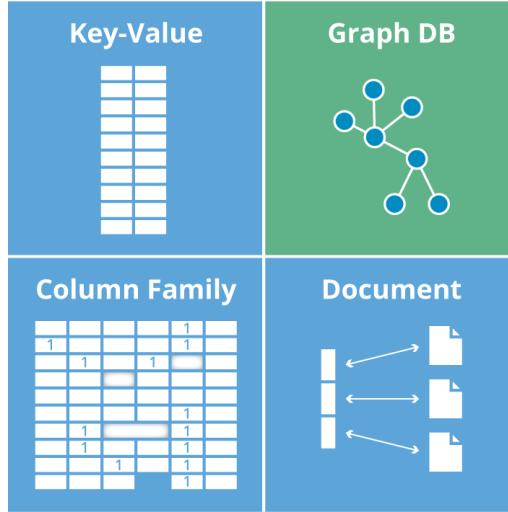


Figura 3.30: Esempio di un database non relazionale

3.5.2 Query Geo-Referenziate

Una delle prerogative principali che il Database deve supportare sono la possibilità di rispondere a query geo-referenziate. Questo significa che un utente debba essere in grado di interrogare il Database per ricevere i soli eventi avvenuti in una certa area geografica. Allo stesso modo, anche la analisi dei dati può essere effettuata in forma locale in modo da poter eventualmente fornire diverse previsioni e considerazioni per diverse aree geografiche in funzione delle prerogative di quella particolare area. Questo implica delle scelte progettuali che tengano conto di questa prerogativa in modo da ottimizzare questo aspetto nella costruzione dell'architettura del Database.

Notevoli sono gli sforzi della ricerca per lo studio delle migliori architetture di Database che garantissero ottime performance quando sottoposti a query geo-referenziate. Nei lavori [32] e [21] viene progettato uno Spatial Database di tipo ICN (Information Centric Networking) denominato OpenGeoBase. Attraverso l'architettura implementata, il Database può facilmente lavorare in maniera distribuita, usando anche differenti Database Engines in parallelo. In questo modo, si realizza una architetture multi-tenant nella quale diversi utenti e diversi processi possono concorrere indipendentemente all'uso del Database. Ulteriormente, in [32] e [20] si dimostra come una implementazione di una simile architettura di Database sia particolarmente adatta ad applicazioni di ITS (Intelligent Transportation System) in quanto consenta l'esecuzione di query geo-referenziate. OpenGeoBase memorizza i dati geo-referenziati in formato GeoJSON supportando così query riferite ad una certa area geografica. Nella fattispecie, le funzionalità implementate da OpenGeoBase sono:

- Routing-by-name per l'invio di query e l'inserimento di nuovi dati
- In-Networking caching per velocizzare le query e ridurre i tempi di calcolo del Database Engine
- Sicurezza di tipo Data-Centric per il supporto di applicazioni multi-tenant

Gli Spatial Database sono solitamente implementati come estensioni o plug-in di DBMS (Database Management System) generici. Tuttavia, in quanto Database Relazionali di tipo SQL, queste soluzioni sono affette da problemi di performances nel momento in cui le dimensioni del Database aumentano. Pertanto, per grandi raccolte di dati (Big Data), i Database NoSQL stanno sempre più prendendo piede rimpiazzando i Database Relazionali in quanto i Non Relazionali risultano più semplici da distribuire in diversi Server. Sebbene un DBMS Relazionale possa offrire un numero maggiore di strumenti per la gestione e creazione di query, in applicazioni dove sono richieste soltanto delle semplici operazioni di read/write su grosse moli di dati, i Database NoSQL sono ritenuti più adatti in quanto possono essere più facilmente distribuiti.

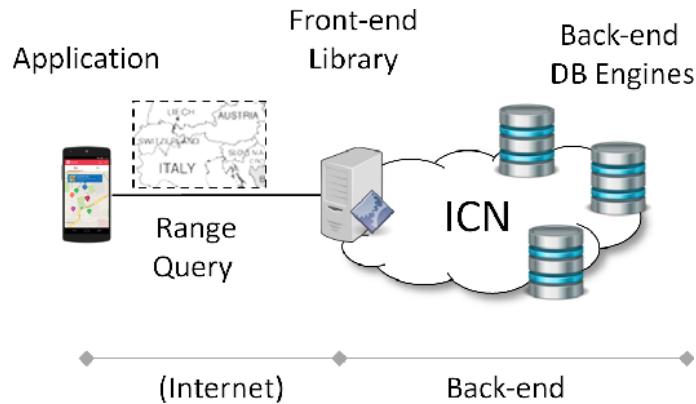


Figura 3.31: Architettura di OpenGeoBase [32]

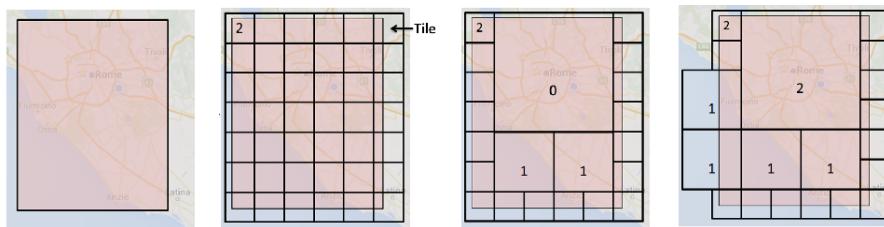


Figura 3.32: Range-Query ed approcci di suddivisione della griglia basata su tre livelli 0,1,2 [32]

Alla luce di quanto detto, è subito evidente che una simile architettura di Database proposta in [32] e [21] bene si adatti ai requisiti del sistema in fase di progettazione illustrati nella Tabella 3.5. Pertanto, alla luce del confronto tra Database Relazionali e Non, ed al fine di rendere il sistema in fase di sviluppo compatibile con l'implementazione di uno Spatial Database come OpenGeoBase, la scelta è ricaduta sull'utilizzo di un Database Non Relazionale e quindi NoSQL. Pertanto, dall'ecosistema AWS si è scelto di utilizzare Amazon DynamoDB.

3.5.3 Amazon DynamoDB

Amazon DynamoDB è il servizio di memorizzazione dei dati in Database Non Relazionali di tipo NoSQL. DynamoDB supporta i modelli di dati di tipo documento e di tipo chiave-valore ed è quindi adatto alla memorizzazione dei dati in formato JSON.

DynamoDB supporta due tipi di chiavi primarie:

- **Partition key:** Una semplice chiave primaria, composta da un attributo chiamato *partition key*. Gli attributi in DynamoDB sono simili in molti modi ai campi o alle colonne di altre arcitetture di Database
- **Partition key and Sort key:** Si riferiscono ad una chiave primaria composta da due attributi. Il primo attributo è la partition key ed il secondo attributo è chiamato *sort key*

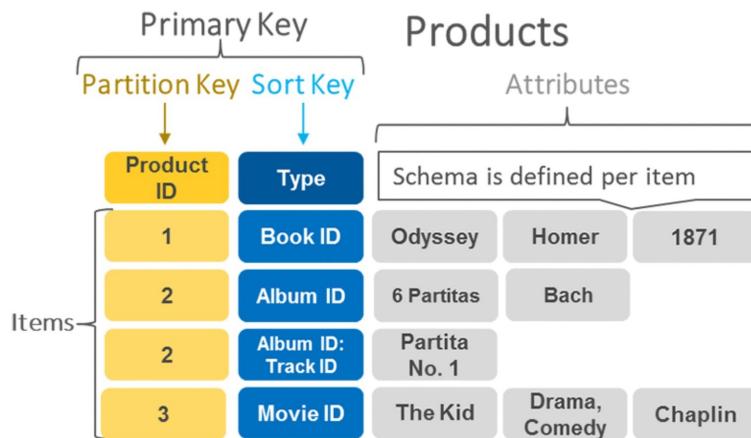


Figura 3.33: Schema di una tabella in DynamoDB evidenziando i campi *partition key* e *sort key*

DynamoDB memorizza i dati come gruppi di attributi, chiamati *items*. Gli *items* sono simili alle righe o tuple o record in altri Database. Ciascun *item* viene memorizzato e recuperato utilizzando la sua chiave primaria che deve quindi essere unica. Gli *item* sono

inoltre memorizzati in unità da 10GB chiamate partitions come mostrato in Figura 3.34. Pertanto, tutti gli *item* aventi lo stesso partition key sono memorizzati insieme, cioè nella stessa partition e sono ordinati in funzione del valore della loro sort key.

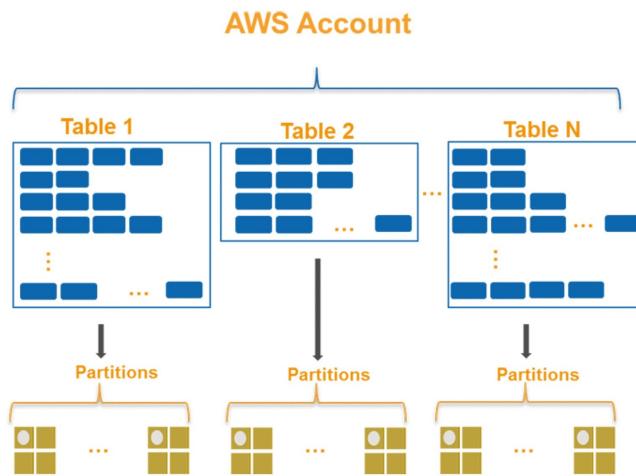


Figura 3.34: Suddivisione di un Database in DynamoDB in tabelle e suddivisione delle tabelle in Partitions

A seguito della overview mostrata su DynamoDB, si ha ora a disposizione uno schema implementativo di un Database con DynamoDB. Pertanto, ora si può procedere al collegamento tra la Cloud Platform IoT Core progettata in precedenza ed il Database DynamoDB. Questa comunicazione sarà necessaria per fare in modo che i dati ricevuti dalla Cloud Platform IoT Core siano automaticamente memorizzati all'interno delle tabelle del Database che saranno poi utilizzate per le query degli utenti e per l'analisi dei dati. Una volta stabilita la comunicazione tra IoT Core e DynamoDB, nel Capitolo 4 sarà mostrata una implementazione di questa comunicazione e come sia possibile interrogare DynamoDB da uno smartphone android per visualizzarne i dati.

Per stabilire il collegamento tra IoT Core e DynamoDB è necessario creare una **regola** in DynamoDB che consenta di recuperare informazioni da un messaggio MQTT in ingresso e di scriverle in una tabella DynamoDB. I passi da seguire sono quindi:

- Creazione di una nuova regola in AWS IoT Core
 - Creazione di una nuova tabella in DynamoDB
 - Creazione di un nuovo ruolo
 - Test della comunicazione

Per avere un maggiore livello di dettaglio sull'implementazione di ciascuna fase, si faccia riferimento alla documentazione ufficiale https://docs.aws.amazon.com/it_it/iot/latest/developerguide/iot-dg.pdf.

Creazione di una nuova Regola

Con la creazione di una nuova Regola si vuole automatizzare il processo di comunicazione tra la piattaforma IoT Core ed il Database DynamoDB. In questo modo, ogni qual volta IoT Core riceve una nuova entry da dispositivi IoT, dopo aver eventualmente verificato e validato i dati, può memorizzarli in un Database per future elaborazioni ed analisi.

Per creare una nuova regola, si acceda alla piattaforma AWS IoT Core come spiegato nella sottosezione 3.4.1 e nel riquadro di navigazione cliccare su *Agisci* e quindi nella pagina *Regole* cliccare su *Crea una Regola*. A questo punto, nella pagina di creazione della nuova regola dovremmo inserire le caratteristiche della regola come Nome e Descrizione. Il campo *Istruzione query della Regola* scegliere la versione più recente del campo Uso di SQL. Infine, nel campo *Istruzioni query della Regola* va immessa la query in linguaggio SQL da eseguire quando viene attivata la regola.

In questo caso, l'istruzione SQL che sarà eseguita consiste in una interrogazione di tutti

The screenshot shows the AWS IoT Core Rules creation interface. At the top, it displays the rule name 'IoT_Storing_Data'. Below the name, there are tabs for 'Panoramica' (Overview), 'Tags', and 'Operazioni'. The 'Panoramica' tab is selected, showing a description: 'Storing into a DynamoDB table all the data coming from the android device'. Under the 'Istruzione query regola' section, there is a note: 'La sorgente di messaggi che desideri elaborare con questa regola.' followed by a code block containing the SQL query: 'SELECT * FROM `EU_WEST_2`'. At the bottom of the interface, it says 'Utilizzo della versione SQL 2016-03-23'.

Figura 3.35: Visualizzazione della Regola sul database DynamoDB

i dati contenuti in una tabella di DynamoDB.

Creazione di una nuova Tabella

Una volta creata la regola in IoT Core, va ora creata e collegata ad una tabella di DynamoDB. Pertanto, nella pagina *Seleziona un'operazione*, scegliere *Inserisci un messaggio in una tabella DynamoDB* e quindi su *Configura Operazione*. Nella pagina *Configura Operazione* scegliere *Crea una nuova Risorsa* e quindi nella pagina di DynamoDB scegliere *Crea Tabella*. Nella pagina *Crea tabella DynamoDB*, compilare i campi relativi alle informazioni della nuova tabella come Nome, Chiave di Partizione e Chiave di Ordinamento. Questi nomi, per testare un primo esempio di comunicazione, sono stati impostati a Row e PoistionInRow. Questi nomi assegnati alle colonne della tabella di DynamoDB dovranno rispettare quelli che saranno ricevuti all'interno del messaggio JSON ricevuto da DynamoDB.

Nel campo *Partizione* e *Chiave di ordinamento* scegliere String e quindi creare la nuova tabella con un click sul pulsante *Crea*.

A questo punto, la tabella di DynamoDB è creata e si può chiudere la scheda del browser

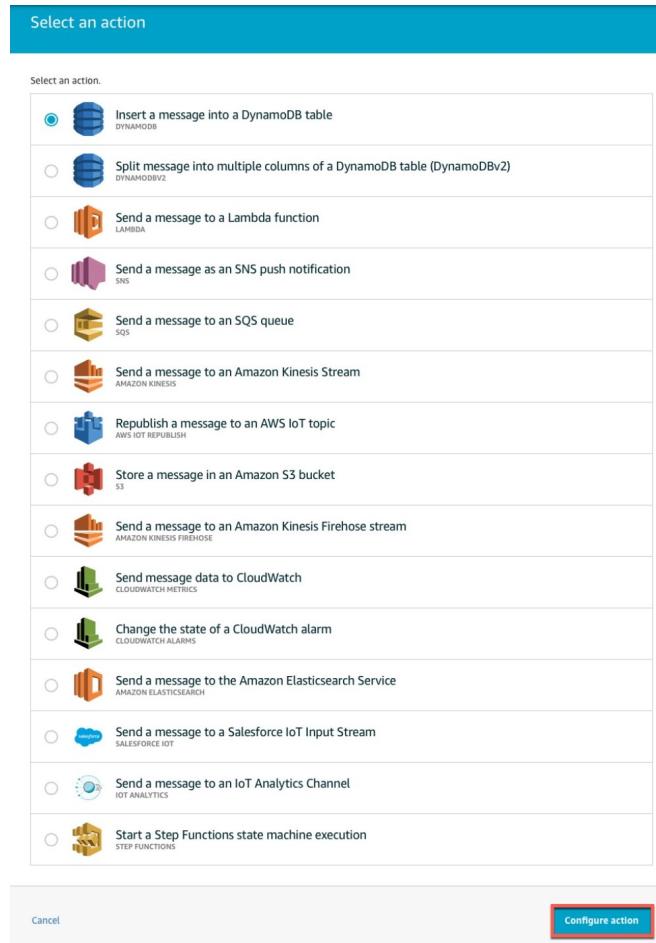


Figura 3.36: Creazione di una nuova tabella

in cui è aperta la console Amazon DynamoDB. A questo punto, nella console di AWS IoT Core precedentemente in fase di configurazione sarà visibile, nell'elenco delle tabelle, la tabella appena creata.

Creazione di un nuovo Ruolo

Una volta creata la nuova tabella in DynamoDB, è ora necessario fornire l'accesso alla lettura e scrittura di questa tabella ad IoT Core. Pertanto, nella console IoT Core, nella sezione *Configura Operazione*, selezionare la tabella appena creata dall'elenco.

Quindi bisogna configurare i dettagli della comunicazione tra IoT Core e DynamoDB e, nel campo *Partition Key* immettere \$row. In questo modo, si indica alla regola di recuperare il valore dell'attributo row dal messaggio MQTT e di scriverlo nella colonna row della tabella DynamoDB. In *Sort Key*, immettere \$pos. In questo modo, il valore dell'attributo pos viene scritto nella colonna PositionInRow. Per impostazione predefinita, l'intero messaggio viene

scritto in una colonna della tabella denominata Payload.

Scegliere quindi *Crea un nuovo ruolo* per procedere alla creazione di un nuovo ruolo da associare ad IoT Core per l'accesso alla tabella di DynamoDB.

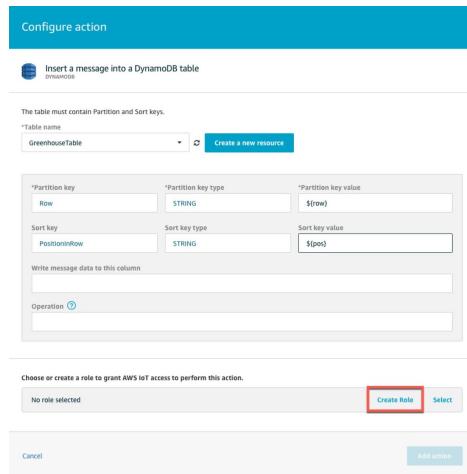


Figura 3.37: Creazione di un nuovo ruolo

Test della Comunicazione

L'infrastruttura di DynamoDB è stata in questo modo creata e collegata ad IoT Core ed è quindi possibile testare la comunicazione tra AWS IoT Core e DynamoDB per verificare che nel momento in cui IoT Core riceva una nuova entry, questa venga automaticamente inviata alla tabella di DynamoDB che è stata in precedenza collegata.

Per fare questo bisogna recarsi nella sezione *Test* e selezionare *Pubblica in un Topic* e nella sezione topic immettere **EU_WEST_2** (lo stesso topic sul quale è in ascolto la regola precedentemente creata Figura 3.35) e nell'area messaggio inserire un nuovo messaggio in formato JSON:

```

1   {
2     "row" : "0",
3     "pos" : "0",
4     "moisture" : "75"
5   }

```

Listing 3.2: Esempio di un possibile messaggio in formato JSON da salvare nel database

Quindi cliccare sul pulsante *Pubblica nell'argomento* per inviare il messaggio a tutti i subscriber che sono in ascolto sul topic **EU_WEST_2**, tra i quali la regola di DynamoDB. Infine, per verificare che la regola stia funzionando, accedere alla console di DynamoDB e nella sezione *Tabelle*, cercare la tabella che è stata creata in precedenza e collegata alla regola. Si dovrà quindi vedere la nuova entry all'interno della tabella contenente proprio il messaggio appena pubblicato nel topic.

Si noti come, anche in questo caso Figura 3.38 si è utilizzato il topic **EU_WEST_2**,

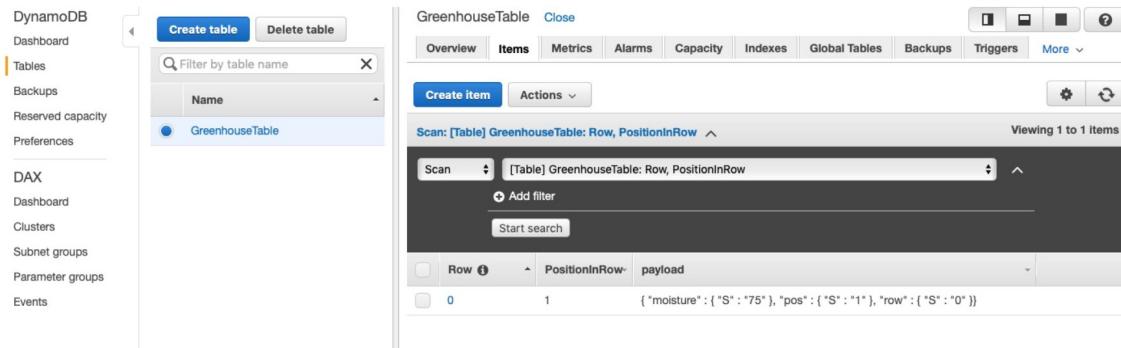


Figura 3.38: Visualizzazione della nuova entry nella tabella

lo stesso utilizzato anche nella sezione 3.4.2. Sebbene il motivo per cui il nome del topic scelto è proprio **EU_WEST_2** sarà esplicitato in dettaglio nella sezione 4.2, per ora basti sapere che, questo stesso set-up della configurazione resta valido e funzionante con qualsiasi altro nome assegnato al topic, purchè i nomi lato Client, Server e lato Database corrispondano.

Capitolo 4

Implementazione del Sistema

In questo modo è stata conclusa la fase di progettazione di tutte le componenti del sistema per lo sviluppo di una infrastruttura software che consenta a dispositivi IoT collegati alla rete di poter:

- Effettuare delle letture con i sensori ed elaborare questi dati
- Stabilire una comunicazione con una piattaforma Cloud opportunamente progettata
- Scambiare i dati con la piattaforma Cloud seguendo un protocollo di comunicazione leggero (MQTT) ed uno standard di comunicazione noto (DATEX II)
- Conservare automaticamente i dati all'interno di un Database Non Relazione
- Rendere i dati disponibili a query ed analisi dei dati

Questi step implementativi che si sono progettati, non sono esclusivi del sistema sviluppato in questo lavoro di tesi, ma sono comuni a moltissime applicazioni del mondo IoT che condividono queste necessità. Pertanto, nel Capitolo 3, si è cercato di mantenere un basso livello di dettaglio implementativo, proprio per consentire di riutilizzare la fase di progettazione mostrata anche in molte altre applicazioni legate al mondo dell'IoT.

In questo capitolo, sulla base della progettazione effettuata, si procederà all'implementazione di tutte le componenti e quindi si raggiungerà un maggiore livello di dettaglio attorno ad alcune tecnologie e linguaggi di programmazione. Nella fattispecie, saranno utilizzati:

- Linguaggi di Programmazione
 - Java
 - SQL
 - XML
 - JSON
- Software
 - Android Studio

- Visual Studio Code
- Tecnologie
 - Smartphone Android
 - Amazon Web Services

Seguendo il processo logico utilizzato nel Capitolo 3, al fine di rendere più chiara l'implementazione del software, si continuerà ad utilizzare il Processo Unificato per la documentazione del Software in fase di sviluppo. Dopo aver visto la tabella dei requisiti nel Capitolo 3 ed il diagramma delle classi relativo all'implementazione dello standard di comunicazione DATEX II nel Appendice A, viene ora realizzato anche il Diagramma dei Casi d'uso il cui scopo è quello di mostrare gli **Attori** ed i **Casi d'Uso** e le relazioni tra di essi. Per **Attore** si intende un qualsiasi utente o servizio esterno al software e per **Casi d'Uso** si indica una funzionalità del software.

Si procederà quindi per step all'implementazione delle singole componenti del sistema e

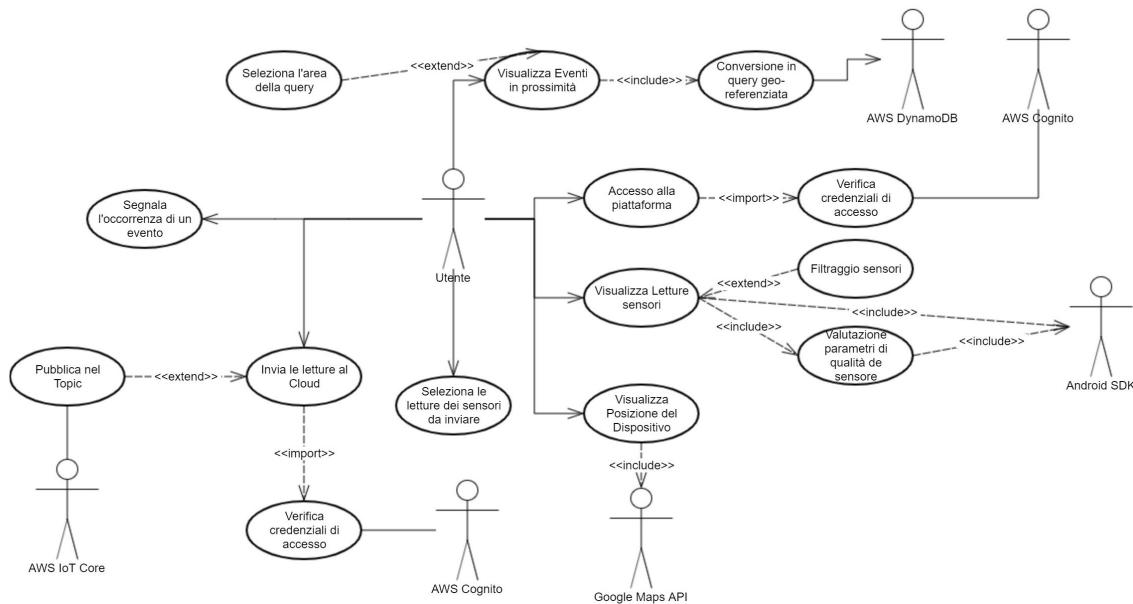


Figura 4.1: Diagramma dei casi d'uso elaborato sulla base dei requisiti funzionali del sistema Tabella 3.2

nella fattispecie, l'ordine implementativo che sarà seguito è:

1. Sviluppo della applicazione Android
2. Sviluppo del Middleware IoT utilizzando la piattaforma AWS IoT Core
3. Sviluppo delle tabelle nel Database AWS DynamoDB

4.1 Applicazione Android

Per lo sviluppo della applicazione Android, si è scelto di utilizzare l'IDE Android Studio ed il linguaggio di programmazione JAVA. Di seguito, pertanto, utilizzando queste tecnologie saranno implementate le singole componenti della applicazione. Nella fattispecie, la struttura che si vuole dare alla Applicazione Andorid è mostrata in Figura 4.2. In Android, ciascuna pagina di una applicazione Android contenente un riferimento al layout da mostrare all'utente ed un riferimento al codice JAVA associato alla pagina è chiamata Activity. Invece, un servizio utilizzato all'interno di una Activity viene chiamato Modulo. Ciascun modulo assolverà ad uno specifico compito. Pertanto, prima di passare alla implementazione di ogni Activity e Modulo della applicazione, si progetti lo schema generale che i vuole implementare.

Dal momento che si è deciso di utilizzare l' IDE Android Studio, si proceda con la creazione

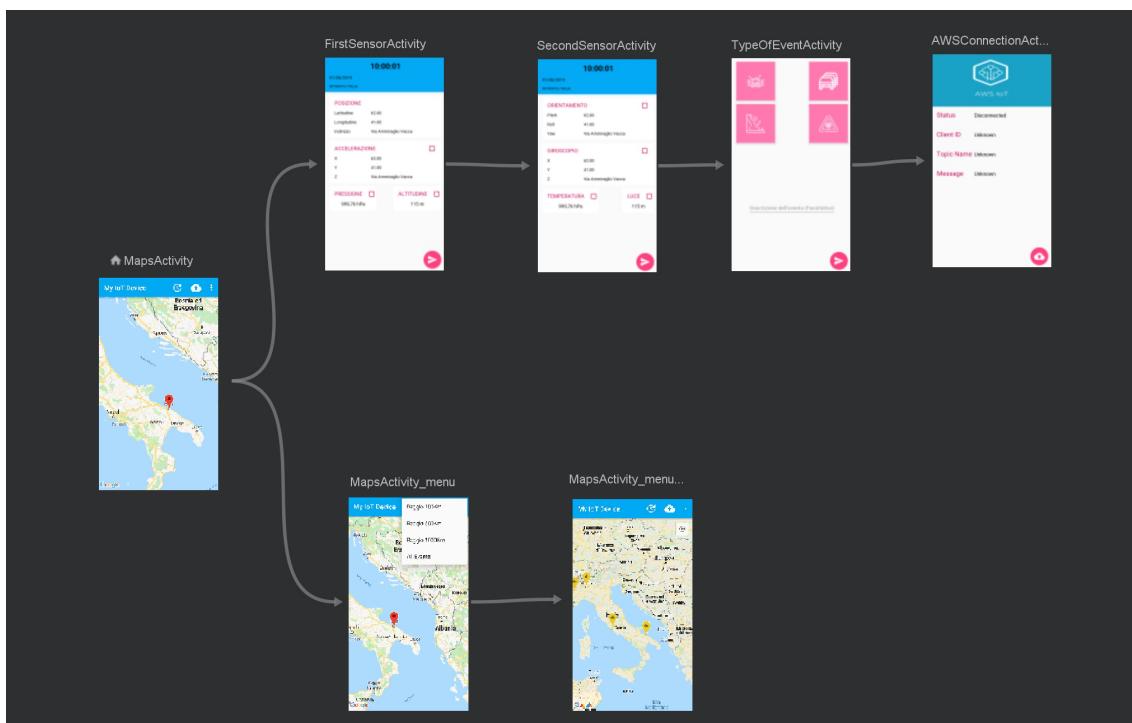


Figura 4.2: Schema delle Activity che compongono la applicazione Android

di un nuovo progetto che quindi produrrà automaticamente lo schema della applicazione. Questo sarà lo scheletro della applicazione dal quale partire con l'implementazione di tutte le Activity ed i Moduli Figura 4.4. Sono quindi mostrate nelle sezioni successive tutte le Activity che compongono la applicazione ed i rispettivi Moduli utilizzati.

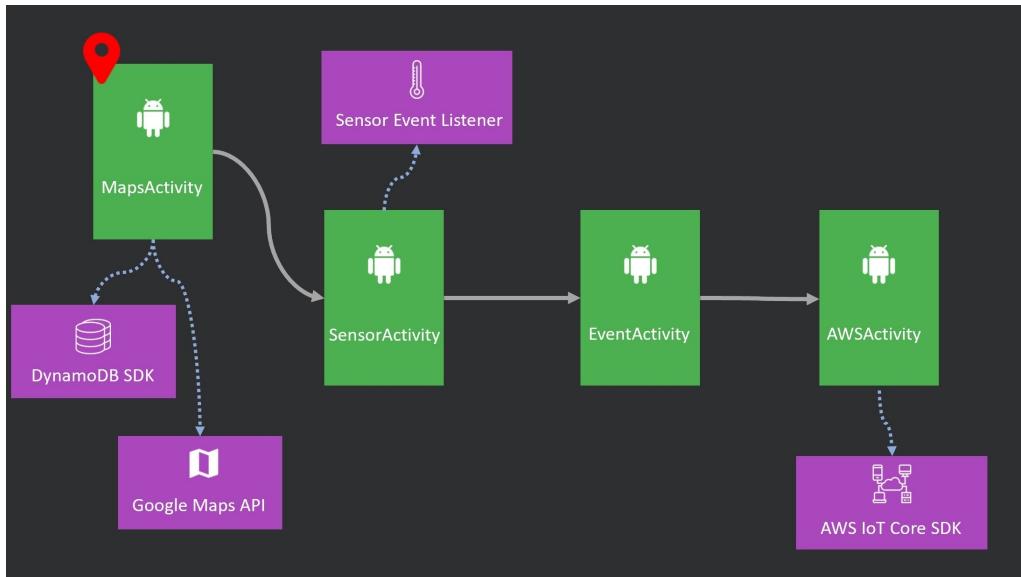


Figura 4.3: Moduli utilizzati da ciascuna Activity

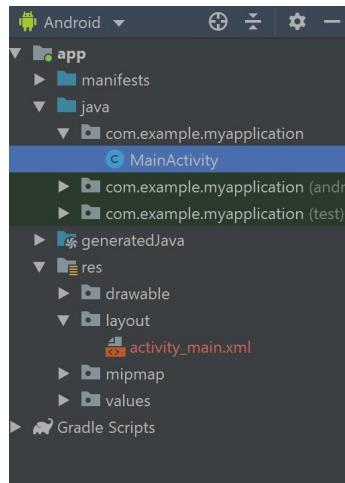


Figura 4.4: Overview della nuova directory creata da Android Studio

4.1.1 MapsActivity

La MainActivity è quella che Android definisce come punto di partenza per il bootstrap o avvio di tutta la Applicazione. In questo caso, si vuole fare in modo che la prima Activity da mostrare all'utente sia la MapsActivity.

Lo scopo di questa Activity è quello di mostrare a schermo una mappa e, sfruttando la localizzazione GPS dello smartphone, di mostrare la posizione in tempo reale dell'utente

sulla stessa mappa.

Ulteriormente, in questa Activity è necessario dare la possibilità all’utente di recuperare gli eventi conservati in una tabella di un Database DynamoDB (Figura 4.5) attraverso una query geo-referenziata 1 e di mostrarli nella mappa attraverso dei marker 2. Inoltre, accedendo al menù 3, l’utente è in grado di modificare l’area degli eventi da recuperare dal Database. In dettaglio, ad un click sul pulsante 3, viene mostrato il menù a tendina

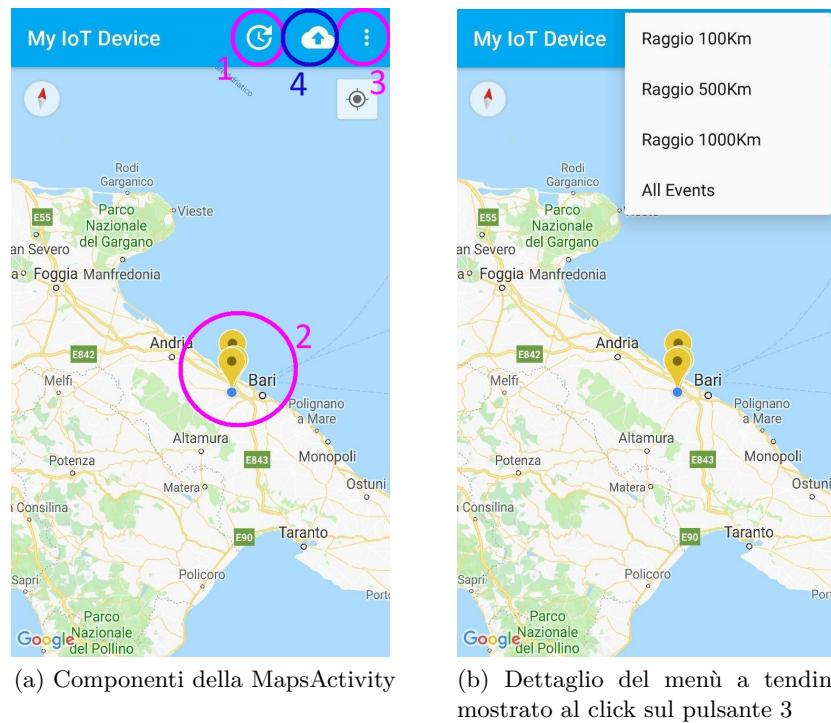


Figura 4.5: Layout della Activity iniziale nell’applicazione Android

che consente all’utente di selezionare la dimensione della query da inoltrare al Database. Invece, un click sul pulsante 4 consentirà all’utente di navigare verso le altre Activities che compongono la applicazione Android e nella fattispecie verso la SensorActivity che mostrerà a schermo le letture dei sensori dei quali lo smartphone è provvisto in real-time. Di seguito, sono sposti gli step implementativi necessari alla realizzazione di questa Activity. Il codice relativo al layout di ciascuna applicazione è stato omesso in quanto non influenza direttamente sulle funzionalità implementate dalla applicazione stessa ed è ritenuto pertanto superfluo.

Option Menu

Per la creazione di un menù a tendina e per la conseguente gestione dei click sullo stesso è necessario:

- *Creare un nuovo layout del menu:* nella cartella **res** si crei una nuova cartella **menu** ed un nuovo file XML al suo interno. Questo file XML appena creato dovrà contenere il layout del menù a tendina da mostrare.
- *Creare la MapsActivity.java ed associare i layout:* Nella cartella **java/com.example.myapplication** si crei una nuova Activity che mostri a schermo il layout della mappa e quello del menu.

```
1 public class MapsActivity extends AppCompatActivity {  
2  
3  
4     @Override  
5     protected void onCreate(Bundle savedInstanceState) {  
6         super.onCreate(savedInstanceState);  
7         setContentView(R.layout.activity_maps);  
8     }  
9 }
```

Listing 4.1: Mostro a schermo il layout che ospiterà la mappa

- *Gestire le interazioni dell’utente con il Menu:* Si rende necessario predisporre un listener in ascolto di eventuali iterazioni dell’utente con gli elementi del menù a tendina. In tal modo, al click su uno dei pulsanti del menù, si andrà ad aggiornare il valore dell’area della query e conseguentemente la query al database verrà reinviata per soddisfare le nuove richieste.

```
1 @Override  
2     public boolean onCreateOptionsMenu(Menu menu) {  
3         MenuInflater menuInflater = getMenuInflater();  
4         menuInflater.inflate(R.menu.main_menu,menu);  
5         return true;  
6     }  
7  
8     @Override  
9     public boolean onOptionsItemSelected(MenuItem item) {  
10         switch (item.getItemId()){  
11             case R.id.item_update:  
12                 refreshGPS();  
13                 return true;  
14             case R.id.item_upload:  
15                 // Go to the new activity  
16                 Intent i = new  
17                     Intent(MapsActivity.this,FirstSensorActivity.class);  
18                 i.putExtra("latitude",String.valueOf(latitude));  
19                 i.putExtra("longitude",String.valueOf(longitude));  
20                 i.putExtra("address",address);  
21                 startActivity(i);  
22                 return true;  
23             case R.id.item_small_radius:
```

```
23         // set min e max latitude/logitude
24         radius=100;
25         mMap.clear();
26         refreshGPS();
27         return true;
28     case R.id.item_medium_radius:
29         // set min e max latitude/logitude
30         radius=500;
31         mMap.clear();
32         refreshGPS();
33         return true;
34     case R.id.item_large_radius:
35         // set min e max latitude/logitude
36         radius=1000;
37         mMap.clear();
38         refreshGPS();
39         return true;
40     case R.id.item_all:
41         // set min e max latitude/logitude
42         radius=8000;
43         mMap.clear();
44         refreshGPS();
45         return true;
46     default:
47         return super.onOptionsItemSelected(item);
48     }
49 }
```

Listing 4.2: Listener associato agli elementi del Menu

GPS e Google Maps API

Una volta creato il layout della applicazione, si implementi ora la componente che consente alla Activity di mostrare a schermo la posizione dell’utente nella mappa. Questa componente utilizzerà la posizione GPS dello smartphon per aggiornare in tempo reale una mappa. Inoltre, al fine di mostrare correttamente la mappa, è necessario sfruttare le API messe a disposizione da Google Maps.

- *Richiesta delle permission all’utente:* Si richiede all’utente il permesso di accedere alla posizione GPS ed alla connessione Internet del suo dispositivo. Il GPS sarà utilizzato per la posizione in real-time e la connessione ad internet sarà utilizzata per mostrare la Mappa.

```
1 // ASK FOR PERMISSIONS
2 // GPS access
```

```
3 if (ActivityCompat.checkSelfPermission(MapsActivity.this,
        Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
        ActivityCompat.checkSelfPermission(MapsActivity.this,
        Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
4     ActivityCompat.requestPermissions(MapsActivity.this, new
            String[]{Manifest.permission.ACCESS_FINE_LOCATION}, 1);
5     return;
6 }
7 // INTERNET access
8 if (ActivityCompat.checkSelfPermission(MapsActivity.this,
        Manifest.permission.INTERNET) != PackageManager.PERMISSION_GRANTED) {
9     ActivityCompat.requestPermissions(MapsActivity.this, new
            String[]{Manifest.permission.INTERNET}, 1);
10    return;
11 }
```

Listing 4.3: Accesso al GPS e connessione Internet

- *Utilizzo delle Google Maps API:* Al fine di mostrare a schermo una mappa è necessario ottenere una chiave API da Google Maps, previa registrazione al Web Service. Va pertanto creato un nuovo documento XML nella directory `res/google_maps_api.xml` e seguire le istruzioni in <https://developers.google.com/maps/documentation/android/start#get-key> per ottenere una nuova chiave. Questa chiave andrà poi inserita all'interno del file XML appena creato.

```
1 <resources>
2     <string name="google_maps_key"
            templateMergeStrategy="preserve"
            translatable="false">API_KEY_HERE</string>
3 </resources>
```

Listing 4.4: Utilizzo della chiave fornita da Google Maps per l'utilizzo delle Mappe

- *Aggiornamento in real time della view:* Sfruttando la posizione GPS e le Google Maps API, viene aggiornata in real-time la posizione dell'utente sulla mappa.

Query del Database

Infine, è necessario gestire le query da inviare al Database DynamoDB. Infatti, in funzione della scelta dell'utente sulla dimensione del raggio della query, verranno formulate delle query geo-referenziate al Database DynamoDB in modo da raccogliere i soli eventi segnalati nell'area prescelta.

- *Utilizzo l'SDK di DynamoDB:* Viene sfruttato l'SDK messo a disposizione da AWS DynamoDB per mappare i dati contenuti nelle tabelle del database, sottoforma di classe all'interno dell'applicazione Android.

```
1 @DynamoDBTable(tableName = "IoT_Data_Storage")
2 public class PublicationDO {
3     private String _creator;
4     private String _publication_time;
5     private Payload _payload;
6
7     @DynamoDBHashKey(attributeName = "Creator")
8     @DynamoDBAttribute(attributeName = "Creator")
9     public String get_creator() {
10         return _creator;
11     }
12
13     public void set_creator(final String _creator) {
14         this._creator = _creator;
15     }
16
17     @DynamoDBRangeKey (attributeName = "PublicationTime")
18     @DynamoDBAttribute(attributeName = "PublicationTime")
19     public String get_publication_time() {
20         return _publication_time;
21     }
22
23     public void set_publication_time(final String
24         _publication_time) {
25         this._publication_time= _publication_time;
26     }
27     @DynamoDBAttribute(attributeName = "payload")
28     public Payload getPayload(){return _payload;}
29     public void setPayload(Payload
30         payload){this._payload=payload;}
31     @DynamoDBDocument
32     public static class Payload{
33         private String _publication_latitude;
34         private String _publication_longitude;
35         private String _publication_location;
36
37         @DynamoDBAttribute(attributeName =
38             "publication_latitude")
39         public String getPublication_latitude(){return
40             _publication_latitude;}
41         public void setPublication_latitude(String
42             publication_latitude)
43             {this._publication_latitude=publication_latitude;}
44         @DynamoDBAttribute(attributeName =
45             "publication_location")
```

```
43     public String getPublication_location(){return
44         _publication_location;}
45     public void setPublication_location(String
46         publication_location)
47     {this._publication_location=publication_location;}
48     @DynamoDBAttribute(attributeName =
49         "publication_longitude")
50     public String getPublication_longitude(){return
51         _publication_longitude;}
52     public void setPublication_longitude(String
53         publication_longitude)
54     {this._publication_longitude=publication_longitude;}
55 }
```

Listing 4.5: Mapping delle tabelle ed attributi del Database DynamoDB che si vuole mappare

- *Creazione di un task asincrono:* Viene utilizzato un task asincrono per l’interrogazione del database in modo da consentire all’utente di continuare ad usare l’interfaccia grafica anche durante le interrogazioni del Database.

```
1 private class QueryTask extends AsyncTask<String,Void,String>{
2
3     @Override
4     protected String doInBackground(String... params) {
5         try {
6             ...
7         }
8         catch (Exception e){}
9         return "Executed";
10    }
11
12    @Override
13    protected void onPostExecute(String result) {
14        ...
15    }
16
17    @Override
18    protected void onPreExecute() {}
19
20    @Override
21    protected void onProgressUpdate(Void... values) {}
22 }
```

Listing 4.6: <struttura del Task Asincrono che dovrà interrogare il Database

- *Composizione della Query:* Dovendo lanciare una query geo-referenziata e, dal momento che i dati nel database sono anch'essi geo-referenziati attraverso i campi latitudine e longitudine, è necessario ora unire le informazioni relative alla latitudine e longitudine corrente dell'utente con quelle relative al raggio della query. L'obiettivo di questo calcolo quello di ottenere un intervallo di posizioni accettabili. Queste saranno caratterizzate dall'avere una latitudine e longitudine comprese all'interno di un certo intervallo.

```
1 private double getDistanceFromPoint(double latB, double lonB){  
2     double latA = latitude;  
3     double lonA = longitude;  
4     double distance=0;  
5  
6     // Conversion in Radiani  
7     latB = convertDegreesInRadians(latB);  
8     lonB = convertDegreesInRadians(lonB);  
9     latA = convertDegreesInRadians(latA);  
10    lonA = convertDegreesInRadians(lonA);  
11  
12    distance = EARTH_RADIUS *  
13        Math.acos((Math.sin(latA)*Math.sin(latB)+  
14            Math.cos(latA)*Math.cos(latB)*Math.cos(lonA-lonB)));  
15  
16    return distance;  
17}  
17 private double convertDegreesInRadians(double deg){  
18     double rad = deg *Math.PI / 180;  
19     return rad;  
20 }
```

Listing 4.7: Calcolo dei parametri con i quali inviare una query geo-referenziata

- *Invio della query geo-referenziata:* Sulla base del mapping effettuato tra gli attributi della tabella di DynamoDB e l'applicazione android, si può ora lanciare la query ed applicare il filtro sull'intervallo di posizioni che si vogliono raccogliere.

```
1 @Override  
2 protected String doInBackground(String... params) {  
3     try {  
4         PublicationDO publication = new PublicationDO();  
5         publication.where(latitude>=latA);  
6         publication.where(latitude<=latB);  
7         publication.where(longitude>=lonA);  
8         publication.where(longitude<=lonB);  
9  
10        Condition rangeKeyCondition = new Condition()  
11            .withComparisonOperator(ComparisonOperator.BEGINS_WITH)  
12            .withAttributeValueList(new  
13                AttributeValue().withS("it"));  
14    } catch (Exception e) {  
15        Log.e("Error", "Error in doInBackground");  
16    }  
17    return null;  
18}
```

```
13     DynamoDBQueryExpression queryExpression = new
14         DynamoDBQueryExpression()
15             .withHashKeyValues(publication)
16             //..withRangeKeyCondition("creator",
17                 rangeKeyCondition)
18             .withConsistentRead(false);
19
20     result = dynamoDBMapper.query(PublicationDO.class,
21         queryExpression);
22 }
```

Listing 4.8: Composizione ed invio della query

Creazione di Marker sulla Mappa

Una volta effettuata la query al Database DynamoDB, è necessario predisporre la ricezione della lista di tutti gli elementi che rispettano i parametri della query, decodificarli dal formato JSON nel quale sono trasmessi ad un formato più user-friendly e conseguentemente mostrarli a schermo sottoforma di marker nella mappa.

- *Ricevo la Lista di messaggi:* Inviata la query, mi aspetto di ricevere una lista di messaggi da parte del Database DynamoDB corrispondenti agli elementi all'interno del Database che soddisfano la query.
-

```
1 @Override
2 protected String doInBackground(String... params) {
3     try {
4         ...
5
6         result = dynamoDBMapper.query(PublicationDO.class,
7             queryExpression);
8
9         gson = new Gson();
10        responses = new ArrayList<JSON_Response>();
11        stringBuilder = new StringBuilder();
12        // Loop through query results
13        for (int i = 0; i < result.size(); i++) {
14            jsonFormOfItem = gson.toJson(result.get(i));
15            response = new JSON_Response();
16            response = new Gson().fromJson(jsonFormOfItem,
17                JSON_Response.class);
18        }
19    } catch (Exception e){}
20    return "Executed";
21 }
```

Listing 4.9: Ricezione degli item che soddisfano la query

- *Creo un marker per ogni evento:* Ogni elemento ricevuto corrisponderà ad un evento geo-referenziato contenuto nel Database. Uso gli attributi latitudine e longitudine dell'evento per mostrarlo nella Mappa precedentemente creata.

```
1  @Override
2  protected void onPostExecute(String result) {
3      // Print markers on the map
4      if(responses!=null) {
5          for (int i = 0; i < responses.size(); i++) {
6              item_latitude = Double.parseDouble(responses.get(i)
7                  .get_payload().get_publication_latitude());
8              item_longitude =
9                  Double.parseDouble(responses.get(i)
10                 .get_payload().get_publication_longitude());
11              item_title = responses.get(i).get_payload()
12                 .get_publication_location();
13
14              Log.d("Latitude: ", 
15                  responses.get(i).get_payload().get_publication_latitude());
16
17              // Reading valid data
18              if ((item_latitude != 0.0) && (item_longitude !=
19                  0.0)) {
20
21                  markerOption = new MarkerOptions();
22                  markerOption.position(new
23                      LatLng(item_latitude, item_longitude));
24                  markerOption.title(item_title);
25                  markerOption.icon(getMarkerIcon("#fdd835"));
26                  mMap.addMarker(markerOption);
27              }
28          }
29      }
30  }
```

Listing 4.10: Visualizzazione di ciascun evento come marker sulla mappa

4.1.2 SensorActivity

La SensorActivity viene attivata ad un click dell'utente sul pulsante 4 della Figura 4.5. Lo scopo di questa Activity è quello di mostrare a schermo una panoramica delle letture restituite da tutti i sensori di cui è dotato lo smartphone per sottoporle ad una revisione dell'utente al fine di decidere quali di queste letture inviare al Server per le successive analisi. Dal momento che il panorama degli smartphone è notoriamente frammentato in termini di dotazione software ed hardware, si è deciso di predisporre una view per ogni

sensore che potrebbe risultare utile alle successive analisi dei dati. Nella fattispecie, i sensori per cui sono stati predisposti i metodi necessari alla raccolta e visualizzazione dei dati sono:

- Accelerazione
- Pressione Atmosferica
- Altimetro
- Orientamento
- Giroscopio
- Temperatura
- Intensità Luminosa

A prescindere dalla dotazione del dispositivo, si predisporranno i metodi utili ad ottenere le letture di questi sensori ove presenti. In caso uno dei sensori sopra citati dovesse non essere in dotazione dello smartphone, non ne sarà data alcuna lettura. Un click da parte

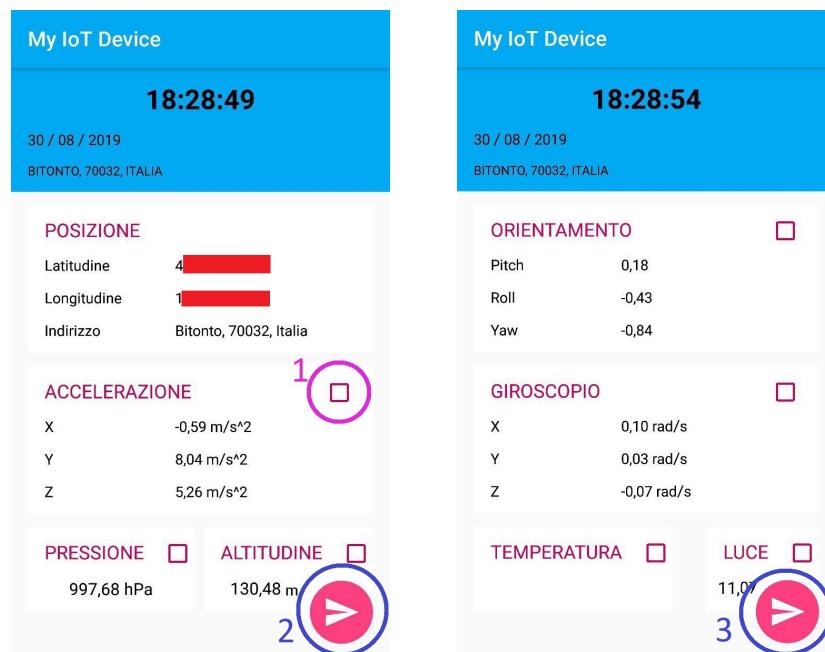


Figura 4.6: Lista Sensori con le relative misure

dell’utente sulla CheckBox 1 consentrà di selezionare la lettura del relativo sensore da inviare al Middleware IoT. Cliccando invece sul FloatingButton 2 si navigherà ad una SensorActivity gemella che mostrerà le letture di altri sensori. Infine, un click sul pulsante 3, farà navigare l’utente verso la Activity successiva (EventActivity).

Di seguito sono mostrati gli step per la visualizzazione dei dati di ciascun sensore. A titolo di esempio, questi step saranno mostrati per l'accelerometro ma potranno essere ripetuti per tutti gli altri sensori.

- *Creazione di un riferimento al SensorManager:* Al fine di ottenere l'accesso alle letture dei sensori, è necessario sfruttare un Modulo messo a disposizione dal framework Android chiamato SensorManager. Il SensorManager, si occuperà di disabilitare i sensori non necessari, specialmente quando la Activity è in pausa. Questo perchè il sistema non disattiva automaticamente i sensori quando lo schermo viene spento né quando l'Activity viene messa in pausa e ciò si tradurrebbe in un eccessivo consumo della batteria.

```
1 public class SensorActivity extends AppCompatActivity {
2
3     @Override
4     protected void onCreate(Bundle savedInstanceState) {
5         super.onCreate(savedInstanceState);
6         setContentView(R.layout.firts_sensor_activity);
7
8         ...
9
10        // Parameters shared by all the sensors
11        senSensorManager = (SensorManager)
12            getSystemService(Context.SENSOR_SERVICE);
13    }
14 }
```

Listing 4.11: Creazione del riferimento al SensorManager che sarà condiviso tra tutti i sensori mostrati nella Activity

- *Accesso alle risorse dello Smartphone:* Per ogni sensore che si intende utilizzare è necessario recuperare un riferimento al particolare sensore attraverso il SensorManager precedentemente istanziaato. I diversi Sensori presenti all'interno di uno smartphone sono individuati dal SensorManager attraverso una lista di costanti accessibili da : https://developer.android.com/guide/topics/sensors/sensors_overview. In questo caso, si crei il riferimento all'accelerometro.

```
1 public class SensorActivity extends AppCompatActivity {
2
3     @Override
4     protected void onCreate(Bundle savedInstanceState) {
5         super.onCreate(savedInstanceState);
6         setContentView(R.layout.firts_sensor_activity);
7
8         ...
9
10        senAccelerometer =
11            senSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
12    }
13 }
```

```
11      }
12 }
```

- *Creazione di un Thread separato:* Al fine di gestire le letture provenienti dal sensore e poterle inoltrare all’utente nella SensorActivity, è necessario creare un Thread separato rispetto a quello sul quale viene eseguita la SensorActivity. In questo modo, la SensorActivity sarà ancora disponibile all’utilizzo mentre un thread parallelo si starà procurando le letture dei sensori.

```
1 public class SensorActivity extends AppCompatActivity {
2
3     @Override
4     protected void onCreate(Bundle savedInstanceState) {
5         super.onCreate(savedInstanceState);
6         setContentView(R.layout.firts_sensor_activity);
7
8         ...
9
10        mAccelerometerThread = new
11            HandlerThread("Accelerometer"
12                "Thread", Thread.NORM_PRIORITY);
13        mAccelerometerThread.start();
14        mAccelerometerHandler = new
15            Handler(mAccelerometerThread.getLooper());
16    }
17}
```

Listing 4.12: Creazione e lancio del thread separato che leggerà i valori dall’accelerometro.

- *Registrazione di un Listener sul Thread separato:* Al fine di recuperare e trattare i dati provenienti dal thread appena creato per l’accelerometro, è necessario creare e registrare un listener il cui compito è quello di eseguire delle operazioni sulle letture del sensore, nonappena sono ricevuti dei nuovi dati dal sensore stesso.

```
1 public class SensorActivity extends AppCompatActivity {
2     public MySensorListener msl;
3
4     @Override
5     protected void onCreate(Bundle savedInstanceState) {
6         super.onCreate(savedInstanceState);
7         setContentView(R.layout.firts_sensor_activity);
8
9         ...
10
11        msl = new MySensorListener(this);
12        senSensorManager.registerListener(msl, senAccelerometer,
13            SensorManager.SENSOR_DELAY_NORMAL, mAccelerometerHandler);
```

```
14      }
15 }
```

Listing 4.13: Registrazione del Listener sull'accelerometro.

```
1 public class MySensorListener implements SensorEventListener {
2     public FirstSensorActivity mainActivity;
3
4     public MySensorListener(FirstSensorActivity activity){
5         this.mainActivity = activity;
6     }
7     public void onAccuracyChanged(Sensor sensor, int
8         accuracy){}
9     public void onSensorChanged(final SensorEvent event){
10        this.mainActivity.runOnUiThread(new Runnable() {
11            @Override
12            public void run() {
13                Sensor mySensor = event.sensor;
14                if
15                    (mySensor.getType() == Sensor.TYPE_ACCELEROMETER){
16                        ...
17                    }
18            }
19        });
}
```

Listing 4.14: Modulo Sensor Event Listener.

- *Filtraggio dei dati in uscita dall'accelerometro:* Come evidenziato nella sottosezione 3.2.3, è necessario effettuare il filtraggio delle letture provenienti da alcuni sensori al fine di ottenere delle letture più accurate. L'accelerometro è tra quei sensori Hardware che richiedono un filtraggio dei dati.

```
1 public float[] lowPassFilter(float[] input, float[] output){
2     if (output == null) return input;
3     for (int i=0; i<input.length; i++) {
4         output[i] = output[i] + ALPHA * (input[i] - output[i]);
5     }
6     return output;
7 }
```

Listing 4.15: Funzione applicata all'output dell'accelerometro per il filtraggio dei dati con un filtro passa-basso.

- *Stampa a schermo delle letture del sensore:* Una volta filtrati, i dati sono disponibili ad essere visualizzati nella SensorActivity.

```
1 public class MySensorListener implements SensorEventListener {
```

```
2     public FirstSensorActivity mainActivity;
3
4     public MySensorListener(FirstSensorActivity activity){
5         this.mainActivity = activity;
6     }
7     public void onAccuracyChanged(Sensor sensor, int
8         accuracy){}
9     public void onSensorChanged(final SensorEvent event){
10        this.mainActivity.runOnUiThread(new Runnable() {
11            @Override
12            public void run() {
13                Sensor mySensor = event.sensor;
14                if
15                    (mySensor.getType() == Sensor.TYPE_ACCELEROMETER){
16                        output =
17                            lowPassFilter(event.values.clone(), output);
18                        // FILTERED
19                        mainActivity.acceleration_coordinateX
20                            .setText(String.format("%.2f",
21                                output[0]) + " m/s^2");
22                        mainActivity.acceleration_coordinateY
23                            .setText(String.format("%.2f",
24                                output[1]) + " m/s^2");
25                        mainActivity.acceleration_coordinateZ
26                            .setText(String.format("%.2f",
                                output[2]) + " m/s^2");
27                    }
28            }
29        });
30    }
31 }
```

Listing 4.16: Accesso al contenuto degli elementi della view nella SensorActivity.

- *Recupero dei parametri di qualità del sensore:* Per concludere con il processo di lettura dei dati di un sensore, è necessario corredare l'output del sensore con i parametri di qualità del sensore, la cui utilità è stata discussa nella sottosezione 3.2.2.

```
1 if (mySensor.getType() == Sensor.TYPE_ACCELEROMETER){
2     ...
3     // Sensor information and name
4     mainActivity.accelerometer_name = mySensor.getName();
5     mainActivity.accelerometer_resolution =
6         String.valueOf(mySensor.getResolution());
7     mainActivity.accelerometer_vendor = mySensor.getVendor();
8     mainActivity.accelerometer_type =
9         String.valueOf(mySensor.getType());
10 }
```

Listing 4.17: Lettura delle caratteristiche salienti del sensore.

4.1.3 EventActivity

La EventActivity viene attivata ad un click dell’utente sul pulsante 3 della Figura 4.6. Lo scopo di questa Activity è quello di mostrare a schermo una lista di possibili eventi stradali tra i quali l’utente può scegliere quelli eventualmente occorsi. Qualora infatti uno di questi eventi dovesse verificarsi lungo la rete stradale, l’utente può informare il sistema e di conseguenza tutti gli altri automobilisti dell’eventuale pericolo. Infine si mette a disposizione anche una label per l’eventuale inserimento di una descrizione dell’evento 2. Un click sul pulsante 3 consentirà invece la navigazione verso la Activity successiva (AWSActivity). Considerata la semplicità del contenuto da mostrare a schermo in questa

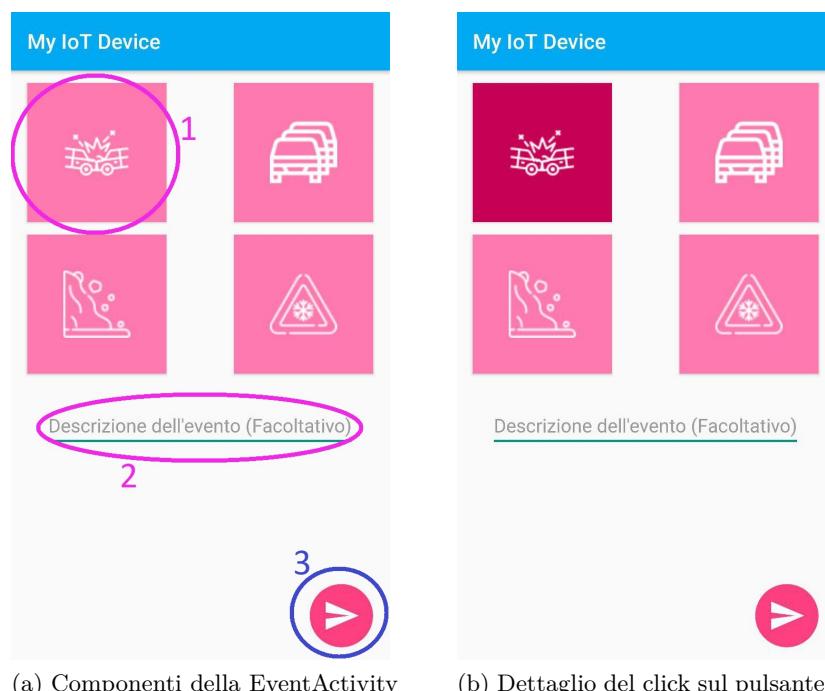


Figura 4.7: Layout della EventActivity

Activity e considerato che questa sarà l’ultima Activity a raccogliere i dati proposti dall’utente, conseguentemente al click dell’utente sul pulsante 3 si creerà direttamente il payload del pacchetto da inviare nella Activity successiva (AWSActivity) in formato XML ed in formato JSON.

Per l’implementazione di queste componenti, sono stati seguiti gli step seguenti:

- *Gestire i click sui Pulsanti:* Per ciascun pulsante corrispondente ad un possibile evento stradale, viene predisposto un Listener che quindi resti in ascolto di interazioni con l’utente e si comporti di conseguenza.

1 `@Override`

2 `protected void onCreate(Bundle savedInstanceState) {`

```
3     super.onCreate(savedInstanceState);
4     setContentView(R.layout.type_of_event_activity);
5
6     car_accident= (Button) findViewById(R.id.car_accident);
7     ...
8
9     car_accident.setOnClickListener(new
10        View.OnClickListener() {
11            @Override
12            public void onClick(View v) {
13                car_accident_click++;
14                if ((car_accident_click%2)-1==0) {
15                    car_accident.setBackgroundColor(getResources()
16                        .getColor(R.color.colorSecondaryDark));
17                }
18                else
19                {
20                    car_accident.setBackgroundColor(getResources()
21                        .getColor(R.color.colorSecondaryLight));
22                }
23            }
24        });
25 }
```

Listing 4.18: Gestione del click sul primo pulsante.

- *Creazione della Publication in DATEX II:* In accordo con quanto definito dallo standard DATEX II che si è deciso di utilizzare come formato per lo scambio dati relativi al traffico, questi dati raccolti dall’utente saranno impacchettati in una Publication e conseguentemente convertiti in XML e JSON. Questa procedura è eseguita in questa Activity sulla base dello schema delle classi dell’Appendice A e sulla base dell’implementazione delle stesse evidenziata nell’Appendice B.
- *Conversione della Publication in XML e JSON:* Android non prevede dei moduli integrati per la conversione di un oggetto in formato XML, ma solo in formato JSON. Pertanto, la conversione dell’oggetto Publication in formato XML sarà fatta utilizzando una libreria esterna.

```
1 @Override
2     protected void onCreate(Bundle savedInstanceState) {
3         super.onCreate(savedInstanceState);
4         setContentView(R.layout.type_of_event_activity);
5         ...
6         XStream xml = new XStream();
7         xml.registerConverter(new
8             SingleObjectConverter([p].class));
9     }
```

Listing 4.19: Utilizzo della libreria XStream per la conversione della publication in formato XML.

Per quanto riguarda invece la convesrione in formato JSON, sarà utilizzata la libreria Gson messa a disposizione da Android.

```

1  @Override
2      protected void onCreate(Bundle savedInstanceState) {
3          super.onCreate(savedInstanceState);
4          setContentView(R.layout.type_of_event_activity);
5          ...
6          Gson gson= new Gson();
7          messageJSON = gson.toJson(p);
8      }

```

Listing 4.20: Utilizzo della libreria Gson per la conversione della Publication in formato JSON.

4.1.4 AWSActivity

La AWS Activity viene attivata ad un click dell’utente sul pulsante 3 della Figura 4.7. Lo

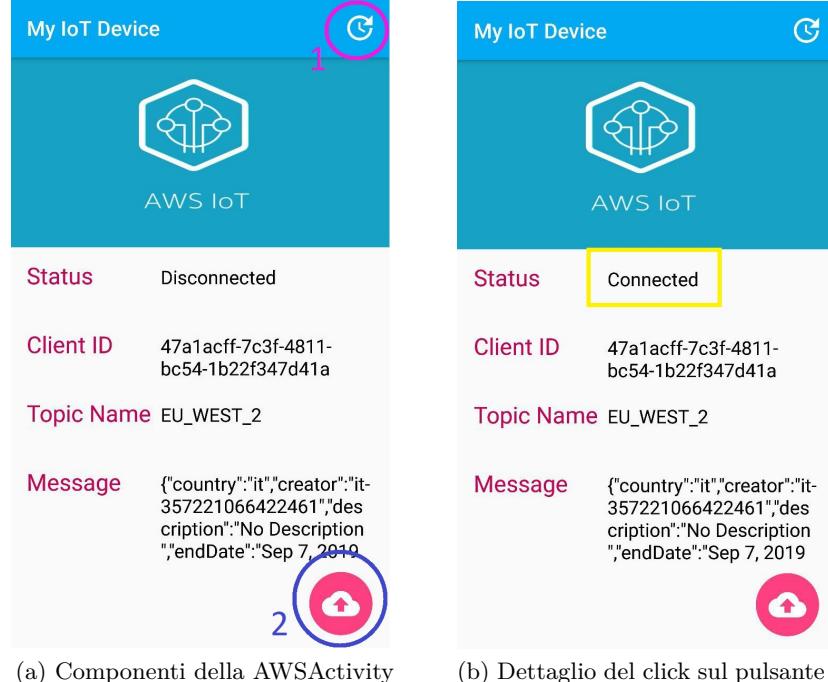


Figura 4.8: Layout della AWSActivity

scopo di questa Activity è quello di stabilire una connessione con la piattaforma Cloud AWS IoT Core attraverso un click sul pulsante **1** e, una volta stabilita la connessione, procedere all’invio del messaggio al Middleware IoT con un click sul pulsante **2**.

Le funzionalità associate a questa Activity potrebbero essere tranquillamente incorporate in un click sul pulsante **3** della Figura 4.7. Tuttavia, ai fini della prototipazione, si è scelto di tenere queste funzionalità in una Activity separata in modo da mostrarne in dettaglio le singole operazioni. La sequenza di operazioni eseguite in questa Activity saranno quindi:

- *Set-Up dei parametri necessari alla Connessione:* Al fine di stabilire la connessione con il Server AWS IoT Core, è necessario che ciascun Client sia registrato al servizio Amazon Web Services e che quindi abbia un ID ed una Password per il proprio accesso. In questa fase prototipale, il processo di registrazione è stato fatto manualmente seguendo la procedura esposta nella sottosezione 3.4.1. Le credenziali di accesso ottenute in questo processo di registrazione saranno utilizzate per l’autorizzazione della applicazione in fase di sviluppo alla comunicazione con il server.

```
1 public class AWSActivity extends AppCompatActivity {  
2  
3     private final String TOPIC_NAME = "EU_WEST_2";  
4  
5     // AWS Connection  
6     static final String LOG_TAG =  
7         AWSConnectionActivity.class.getCanonicalName();  
8     // --- Constants to modify per your configuration ---  
9     // IoT endpoint  
10    // AWS IoT CLI describe-endpoint call returns:  
11        XXXXXXXXXX.iot.<region>.amazonaws.com  
12    private static final String CUSTOMER_SPECIFIC_ENDPOINT =  
13        "ENDPOINT-HERE";  
14    // Cognito pool ID. For this app, pool needs to be  
15        unauthenticated pool with  
16    // AWS IoT permissions.  
17    private static final String COGNITO_POOL_ID =  
18        "POOLID-HERE";  
19    // Name of the AWS IoT policy to attach to a newly created  
20        certificate  
21    private static final String AWS_IOT_POLICY_NAME =  
22        "myAndroidIoTPolicy";  
23    // Region of AWS IoT  
24    private static final Regions MY_REGION = Regions.EU_WEST_2;  
25    // Filename of KeyStore file on the filesystem  
26    private static final String KEYSTORE_NAME = "iot_keystore";  
27    // Password for the private key in the KeyStore  
28    private static final String KEYSTORE_PASSWORD =  
29        "PASSWORD-HERE";  
30    // Certificate and key aliases in the KeyStore  
31    private static final String CERTIFICATE_ID = "default";  
32    AWSIoTClient mIoTAndroidClient;  
33    AWSIoTMqttManager mqttManager;
```

```
26     String clientId;
27     String keystorePath;
28     String keystoreName;
29     String keystorePassword;
30     KeyStore clientKeyStore = null;
31     String certificateId;
32     CognitoCachingCredentialsProvider credentialsProvider;
33     private boolean isConnected;
34
35
36     @Override
37     protected void onCreate(Bundle savedInstanceState) {
38         super.onCreate(savedInstanceState);
39         setContentView(R.layout.aws_connection_activity);
40         ...
41     }
42 }
```

Listing 4.21: Variabili necessarie alla autenticazione del Client.

- *Creazione del ClientID*: La trasmissione dei dati tra la applicazione android Client ed il Server AWS IoT Core avverrà utilizzando il protocollo MQTT il quale richiede un Client ID univoco. Tale Client ID viene generato automaticamente da una funzione randomica.

```
1 public class AWSActivity extends AppCompatActivity {
2     @Override
3     protected void onCreate(Bundle savedInstanceState) {
4         super.onCreate(savedInstanceState);
5         setContentView(R.layout.aws_connection_activity);
6
7         // MQTT client IDs are required to be unique per AWS
8         // IoT account.
9         // This UUID is "practically unique" but does not
10        // guarantee_
11        // uniqueness.
12        clientId = UUID.randomUUID().toString();
13        clientIdView.setText(clientId);
14    }
}
```

Listing 4.22: Generazione di un Client ID.

- *Autenticazione attraverso AWS Cognito*: AWS Cognito è lo strumento che consente di aggiungere strumenti di registrazione, accesso e controllo sugli accessi in applicazioni Web e dispositivi mobili. In automatico quindi, si andrà a registrare il dispositivo ad AWS Cognito per l'accesso ad AWS IoT Core tramite le credenziali di accesso precedentemente registrate nella AWSActivity.

```
1 public class AWSConnectionActivity extends AppCompatActivity {
2     @Override
3     protected void onCreate(Bundle savedInstanceState) {
4         super.onCreate(savedInstanceState);
5         setContentView(R.layout.aws_connection_activity);
6         ...
7         // Initialize the AWS Cognito credentials provider
8         credentialsProvider = new
9             CognitoCachingCredentialsProvider(
10                 getApplicationContext(), // context
11                 COGNITO_POOL_ID, // Identity Pool ID
12                 MY_REGION // Region
13             );
14         Region region = Region.getRegion(MY_REGION);
15         mqttManager = new AWSIotMqttManager(clientId,
16             CUSTOMER_SPECIFIC_ENDPOINT);
17         // Set keepalive to 10 seconds. Will recognize
18         // disconnects more quickly but will also send
19         // MQTT pings every 10 seconds.
20         mqttManager.setKeepAlive(10);
21         // Set Last Will and Testament for MQTT. On an unclean
22         // disconnect (loss of connection)
23         // AWS IoT will publish this message to alert other
24         // clients.
25         AWSIotMqttLastWillAndTestament lwt = new
26             AWSIotMqttLastWillAndTestament("my/lwt/topic",
27                 "Android\u00d7client\u00d7lost\u00d7connection",
28                 AWSIotMqttQos.QOS0);
29         mqttManager.setMqttLastWillAndTestament(lwt);
30         // IoT Client (for creation of certificate if needed)
31         mIoTAndroidClient = new AWSIotClient(credentialsProvider);
32         mIoTAndroidClient.setRegion(region);
33         keystorePath = getFilesDir().getPath();
34         keystoreName = KEYSTORE_NAME;
35         keystorePassword = KEYSTORE_PASSWORD;
36         certificateId = CERTIFICATE_ID;
37         try {
38             if
39                 (AWSIotKeystoreHelper.isKeystorePresent(keystorePath,
40                     keystoreName)) {
41                 if
42                     (AWSIotKeystoreHelper.keystoreContainsAlias(certificateId,
43                         keystorePath,
44                             keystoreName, keystorePassword)) {
45                         Log.i(LOG_TAG, "Certificate\u00d7" + certificateId
46                             + "\u00d7found\u00d7in\u00d7keystore\u00d7\u00d7using\u00d7for\u00d7
47                             MQTT.\u00d7");
48                     }
49                 }
50             }
51         }
52     }
53 }
```

```
37         clientKeyStore =
38             AWSIoTKeystoreHelper.getIotKeystore(certificateId,
39                                         keystorePath, keystoreName,
40                                         keystorePassword);
41     } else {
42         Log.i(LOG_TAG, "Key/cert" + certificateId + " "
43               + "not found in keystore.");
44     }
45 } else {
46     Log.i(LOG_TAG, "Keystore" + keystorePath + "/" +
47           keystoreName + "not found.");
48 }
49 } catch (Exception e) {
50     Log.e(LOG_TAG, "An error occurred retrieving cert/key "
51           + "from keystore.", e);
52 }
53 if (clientKeyStore == null) {
54     Log.i(LOG_TAG, "Cert/key was not found in keystore - "
55           + "creating new key and certificate.");
56     new Thread(new Runnable() {
57         @Override
58         public void run() {
59             try {
60                 CreateKeysAndCertificateRequest
61                 createKeysAndCertificateRequest =
62                     new
63                         CreateKeysAndCertificateRequest();
64                 createKeysAndCertificateRequest.setSetActive(true);
65                 final CreateKeysAndCertificateResult
66                     createKeysAndCertificateResult;
67                 createKeysAndCertificateResult =
68                     mIoTAndroidClient
69                         .createKeysAndCertificate(
70                             createKeysAndCertificateRequest);
71                 Log.i(LOG_TAG,
72                     "Cert ID: " +
73                         createKeysAndCertificateResult
74                             .getCertificateId() +
75                         " created.");
76                 AWSIoTKeystoreHelper
77                     .saveCertificateAndPrivateKey(certificateId,
78                         createKeysAndCertificateResult
79                             .getCertificatePem(),
80                         createKeysAndCertificateResult
81                             .getKeyPair().getPrivateKey(),
82                         keystorePath, keystoreName,
83                         keystorePassword);
84             clientKeyStore =
85                 AWSIoTKeystoreHelper.getIotKeystore(certificateId,
```

```
75                     keystorePath, keystoreName,
76                     keystorePassword);
77             AttachPrincipalPolicyRequest
78             policyAttachRequest =
79                 new AttachPrincipalPolicyRequest();
80             policyAttachRequest.setPolicyName(AWS_IOT_POLICY_NAME);
81             policyAttachRequest
82                 .setPrincipal(createKeysAndCertificateResult
83                               .getCertificateArn());
84             mIoTAndroidClient
85                 .attachPrincipalPolicy(policyAttachRequest);
86         } catch (Exception e) {
87             Log.e(LOG_TAG,
88                   "Exception occurred when
89                   generating new private key and
90                   certificate.",
91                   e);
92         }
93     }
94 }
```

Listing 4.23: Utilizzo di AWS Cognito per l'accesso ad AWS IoT Core.

- *Stabilire la Connessione con il Server:* Una volta implementati i metodi messi a disposizione dall'AWS SDK per garantire l'accesso alla applicazione android, si gestisce ora il click sul pulsante 1 per la creazione di una connessione con il server.

```
1 private void refreshConnection(){
2     Log.d(LOG_TAG, "clientId=" + clientId);
3     try {
4         mqttManager.connect(clientKeyStore, new
5             AWSIoTMqttClientStatusCallback() {
6                 @Override
7                 public void onStatusChanged(final
8                     AWSIoTMqttClientStatus status,
9                     final Throwable
10                        throwable) {
11                     Log.d(LOG_TAG, "Status=" +
12                         String.valueOf(status));
13                     runOnUiThread(new Runnable() {
14                         @Override
15                         public void run() {
16                             if (status ==
17                                 AWSIoTMqttClientStatus.Connecting) {
18                                 statusView.setText("Connecting...");
19                             } else if (status ==
20                                 AWSIoTMqttClientStatus.Connected) {
21                                 statusView.setText("Connected");
22                             }
23                         }
24                     });
25                 }
26             });
27     } catch (Exception e) {
28         Log.e(LOG_TAG, "Error connecting to IoT Core: " + e.getMessage());
29     }
30 }
```

```
16                     isConnected = true;
17             } else if (status ==
18                 AWSIoTMqttClientStatus.Reconnecting)
19             {
20                 if (throwable != null) {
21                     Log.e(LOG_TAG, "Connection\u2014
22                         error.", throwable);
23                 }
24                 statusView.setText("Reconnecting");
25             } else if (status ==
26                 AWSIoTMqttClientStatus.ConnectionLost)
27             {
28                 if (throwable != null) {
29                     Log.e(LOG_TAG, "Connection\u2014
30                         error.", throwable);
31                 }
32                 statusView.setText("Disconnected");
33                 isConnected = false;
34             } else {
35                 statusView.setText("Disconnected");
36                 isConnected = false;
37             }
38         }
39     }
40 }
```

Listing 4.24: Connessione con l’AWS IoT Core.

- *Invio del messaggio:* In conclusione, una volta stabilita la connessione, l’utente può cliccare sul pulsante **2** per l’invio del pacchetto al Server.

```
1 send_btn.setOnClickListener(new View.OnClickListener() {
2     @Override
3     public void onClick(View view) {
4         if(isConnected==true) {
5             publish(messageJSON);
6             //disconnect();
7             Toast.makeText(getApplicationContext(),
8                 "Publication\u2014Complete",
9                 Toast.LENGTH_SHORT).show();
10        }
11    }
12 }
```

```
10         Toast.makeText(getApplicationContext(),"Make
11             sure you are connected to the u
12             server",Toast.LENGTH_SHORT);
13     }
14 }
15 private void publish(String msg){
16     try {
17         mqttManager.publishString(msg, TOPIC_NAME,
18             AWSIoT MQTTQos.QOS0);
19     } catch (Exception e) {
20         Log.e(LOG_TAG, "Publisherror.", e);
21     }
}
```

Listing 4.25: Invio della Publication al Server.

4.2 AWS IoT Core

Nella sottosezione 3.4.2 si è mostrato come procedere alla registrazione ed utilizzo della piattaforma AWS IoT Core, tuttavia se ne sono volutamente omessi alcuni dettagli implementativi per essere trattati in questa sezione, in modo da avere una migliore panoramica sulla struttura della Applicazione e sulla tipologia dei messaggi inviati. Si è visto quindi come registrarsi al servizio e come fare in modo che IoT Core sia in ascolto su un predefinito topic e che tutti i client che vogliono comunicare con il server, debbano pubblicare messaggi sullo stesso topic nel quale il server è in ascolto.

Nella sottosezione 3.4.2 e nella sottosezione 4.1.4 si è fatto sempre riferimento a pubblicazioni sul topic denominato **EU_WEST_2** senza tuttavia spiegarne il motivo.

Si rende necessario introdurre una feature di AWS IoT Core per la quale viene creata una console indipendente (o Server) per ogni area geografica. Questo significa che, un dispositivo in una data area geografica, avrà un topic ad esso corrispondente (che in questo caso è appunto **EU_WEST_2**) e quindi potrà inviare i dati solo al topic corrispondente. In questo modo IoT Core riesce a garantire una buona scalabilità e delle ottime prestazioni anche in corrispondenza di un numero molto elevato di dispositivi.

Questa suddivisione granulare dei topic e dei server in ascolto sui diversi topic facilita anche l’interfacciamento con il Database e facilita analogamente l’invio di query geo-referenziate al Database. Infatti, attraverso questa granularità sarà possibile implementare un diverso Database DynamoDB associato a ciascun server in ascolto su ciascun topic in modo da estendere la scalabilità di cui gode AWS IoT Core anche al database DynamoDB. In questo modo, infatti, una query da parte di un utente appartenente ad una data area geografica (e quindi con uno specifico Topic) sarà riferita ad un singolo e specifico Database DynamoDB che conterrà i dati relativi a quella sola area geografica.

Tuttavia, qualora necessario, nulla vieta ad un utente di interrogare tutti gli altri Database DynamoDB relativi ad altre aree geografiche. Eventualmente, queste interrogazioni

su diversa scala, possono anche essere regolamentate da diversi processi di autenticazione e che quindi solo alcuni utenti possano interrogare Database diversi rispetto a quello relativo alla propria area geografica. Un’altro vantaggio di questa separazione tra i server che

The screenshot shows the AWS IoT Core interface. On the left, there's a sidebar with navigation links like Monitoraggio, Integrazione, Gestione Oggetti, Tipi, Gruppi di oggetti, Gruppi di fatturazione, Processi, Greengrass, Sicurezza, Difendi, Esecuzione azioni, and Test. The main area is titled 'Oggetti' and contains a search bar with the placeholder 'Cerca oggetti' and a button with a magnifying glass icon. Below the search bar, there's a list with one item: 'Android_Device' and 'NESSUN TIPO'. To the right of the main area is a sidebar titled 'Londra' which lists various geographical regions: Stati Uniti orientali (Virginia settentrionale), Stati Uniti orientali (Ohio), Stati Uniti occidentali (California settentrionale), Stati Uniti occidentali (Oregon), Asia Pacifico (Hong Kong), Asia Pacifico (Mumbai), Asia Pacifico (Seoul), Asia Pacifico (Singapore), Asia Pacifico (Sydney), Asia Pacifico (Tokyo), Canada (Centrale), UE (Francoforte), UE (Irlanda), UE (Londra) (which is highlighted in orange), UE (Parigi), UE (Stoccolma), Medio Oriente (Bahrein), and Sud America (San Paolo).

Figura 4.9: Lista di tutti i topic supportati da AWS IoT Core suddivisi per area geografica.

sono in ascolto sui diversi topic e quindi sulle diverse aree geografiche, consiste nel fatto che gli oggetti ‘shadow’ creati su un server (come spiegato nella sottosezione 3.4.2), sono relativi solo a quel server in ascolto su quel topic in quell’area geografica. In questo modo, è possibile creare oggetti diversi in aree geografiche diverse in funzione eventualmente delle diverse disponibilità, necessità e standard utilizzati in ciascuna area geografica.

Ulteriormente, così come gli oggetti sono relativi ad una singola area geografica o topic, anche le regole di AWS IoT Core lo saranno. Nella sottosezione 3.5.3 si era infatti implementata una regola che consentisse al Middleware IoT Core di inviare ogni nuovo dato ricevuto avente come topic **EU_WEST_2** ad una tabella del database di DynamoDB. Questa regola sarà ovviamente valida per il solo server IoT Core in ascolto sul topic

The screenshot shows the configuration of a Lambda rule named 'IoT_Storing_Data'. At the top, it says 'REGOLA' and 'IoT_Storing_Data'. Below that is a section titled 'ABILITATI' with a 'Operazioni ▾' button. The main area has two tabs: 'Panoramica' and 'Tags'. Under 'Panoramica', there's a 'Descrizione' field containing 'Storing into a DynamoDB table all the data coming from the android device'. There's also an 'Istruzione query regola' field with the text 'La sorgente di messaggi che desideri elaborare con questa regola.' followed by a code block containing 'SELECT * FROM `EU_WEST_2`'. At the bottom, it says 'Utilizzo della versione SQL 2016-03-23'.

Figura 4.10: Regola per la scrittura sul Database DynamoDB

EU_WEST_2 mentre, per gli altri server in ascolto su altri topic potrà essere seguita la

stessa procedura mostrata nella sottosezione 3.5.3 con il riferimento ad un diverso topic. Le nuove regole sviluppate potranno, ove ritenuto opportuno, inserire i loro dati all'interno della stessa tabella di DynamoDB, ovvero all'interno della stessa tabella nella quale inserisce i dati anche la regola mostrata in Figura 4.10. Alternativamente, i dati potranno essere inseriti all'interno di diverse tabelle nello stesso database o ancora potranno essere inseriti in una diversa tabella di un diverso database. Certamente questo ultimo caso si rende ideale per garantire delle performance migliori e si rende adatto a realizzare l'architettura descritta in [32]. Tuttavia, nel prototipo realizzato in questo lavoro di tesi si è scelto per semplicità di implementare la prima soluzione. In questo modo, una singola tabella di un singolo database in DynamoDB conterrà tutti i dati relativi a tutti i topic.

4.3 DynamoDB

Come ultima componente, resta da implementare la struttura del database in DynamoDB. La procedura che è stata seguita è quella descritta nella sottosezione 3.5.3. Questa stessa procedura può essere seguita per la creazione di diverse tabelle in diversi Database relativi a diversi topic.

L'ultima considerazione da effettuare riguarda il formato nel quale i dati sono memorizzati all'interno del Database ed inviati allo stesso. Nella sottosezione 4.1.3 si è ritenuto opportuno convertire (o serializzare) il payload del messaggio (o publication) da inviare al middleware IoT Core in formato XML ed in formato JSON. Questa duplia serializzazione dello stesso payload è effettuata in modo da poter mostrare in output, sottoforma di file di testo, il payload del messaggio in formato XML, come da Standard DATEX II. Un esempio di output è mostrato nell' Appendice B per evidenziare la corrispondenza tra il payload generato dalla applicazione ed il payload richiesto dallo Standard DATEX II.

Tuttavia, sebbene il middleware IoT AWS Core supporti messaggi in formato XML, il database DynamoDB, essendo un database NoSQL memorizza i dati sottoforma di coppie chiave-valore e quindi in formato JSON. Questo problema potrebbe essere risolto inviando il payload in formato XML fino al middleware IoT Core, effettuare la conversione dall'XML al JSON ed infine memorizzare il payload in formato JSON all'interno del Database. Tuttavia, questo procedimento di riconversione risulterebbe più lento e dispendioso rispetto alla generazione e trasmissione dei dati direttamente in formato JSON. Per questo motivo, già all'interno della applicazione android viene generato il payload in formato JSON.

```
1  {
2      "country": "it",
3      "creator": "it-357221066422461",
4      "description": "No\u00d7Description",
5      "endDate": "Sep\u20227,\u20222019\u202217:44:17",
6      "isCarAccident": true,
7      "isLandSlide": false,
8      "isSnow": false,
9      "isTrafficJam": false,
10     "language": "it",
11     "measurementOrCalculationPeriod": "7169",
```

```
12 "measurementOrCalculationTimePrecision": "1us",
13 "nationalIdentifier": "null",
14 "publicationTime": "07/09/2019\u00a005:44:15",
15 "publication_latitude": "4----",
16 "publication_location": "Bitonto ,\u202270032,\u2022Italia",
17 "publication_longitude": "1----",
18 "reading_duration": -7169,
19 "records": [
20 {
21     "ID": "ACC",
22     "computationalMethod": "low\u00a9pass\u00a9filter",
23     "exception_period": "null",
24     "isOverrunning": false,
25     "overallEndTime": "07/09/2019\u00a005:44:10",
26     "overallStartTime": "07/09/2019\u00a005:44:10",
27     "recordId": 0,
28     "safetyRelatedMessage": false,
29     "sensor_address": "Bitonto ,\u202270032,\u2022Italia",
30     "sensor_latitude": "4----",
31     "sensor_longitude": "1----",
32     "sensor_name": "ICM20610\u00a9Acceleration\u00a9Sensor",
33     "sensor_reading_1": "-0,03\u00b3m/s^2",
34     "sensor_reading_2": "2,36\u00b3m/s^2",
35     "sensor_reading_3": "9,60\u00b3m/s^2",
36     "sensor_resolution": "0.0011971008",
37     "sensor_type": "1",
38     "sensor_vendor": "Invensense",
39     "type": "null",
40     "validity_period": "100",
41     "validity_status": "suspended"
42 },
43 {
44     "ID": "PRESS",
45     "computationalMethod": "low\u00a9pass\u00a9filter",
46     "exception_period": "null",
47     "isOverrunning": false,
48     "overallEndTime": "07/09/2019\u00a005:44:10",
49     "overallStartTime": "07/09/2019\u00a005:44:10",
50     "recordId": 0,
51     "safetyRelatedMessage": false,
52     "sensor_address": "Bitonto ,\u202270032,\u2022Italia",
53     "sensor_latitude": "4----",
54     "sensor_longitude": "1----",
55     "sensor_name": "LPS25H\u00a9Barometer\u00a9Sensor",
56     "sensor_reading_1": "996,79\u00a9hPa",
57     "sensor_reading_2": "null",
58     "sensor_reading_3": "null",
59     "sensor_resolution": "1.0",
60     "sensor_type": "6",
```

```
61     "sensor_vendor": "STMicroelectronics",
62     "type": "null",
63     "validity_period": "100",
64     "validity_status": "suspended"
65   },
66   {
67     "ID": "ALT",
68     "computationalMethod": "lowpassfilter",
69     "exception_period": "null",
70     "isOverrunning": false,
71     "overallEndTime": "07/09/2019\u05:44:10",
72     "overallStartTime": "07/09/2019\u05:44:10",
73     "recordId": 0,
74     "safetyRelatedMessage": false,
75     "sensor_address": "Bitonto,\u202270032,\u2022Italia",
76     "sensor_latitude": "4----",
77     "sensor_longitude": "1----",
78     "sensor_name": "LPS25H\u2022Barometer\u2022Sensor",
79     "sensor_reading_1": "137,92\u00b3m",
80     "sensor_reading_2": "null",
81     "sensor_reading_3": "null",
82     "sensor_resolution": "1.0",
83     "sensor_type": "6",
84     "sensor_vendor": "STMicroelectronics",
85     "type": "null",
86     "validity_period": "100",
87     "validity_status": "suspended"
88   },
89   {
90     "ID": "ORIEN",
91     "computationalMethod": "lowpassfilter",
92     "exception_period": "null",
93     "isOverrunning": false,
94     "overallEndTime": "07/09/2019\u05:44:15",
95     "overallStartTime": "07/09/2019\u05:44:15",
96     "recordId": 0,
97     "safetyRelatedMessage": false,
98     "sensor_address": "Bitonto,\u202270032,\u2022Italia",
99     "sensor_latitude": "4----",
100    "sensor_longitude": "1----",
101    "sensor_name": "Rotation\u2022Vector",
102    "sensor_reading_1": "0,01",
103    "sensor_reading_2": "-0,74",
104    "sensor_reading_3": "-0,04",
105    "sensor_resolution": "5.9604645E-8",
106    "sensor_type": "11",
107    "sensor_vendor": "Invensense",
108    "type": "null",
109    "validity_period": "100",
```

```
110      "validity_status": "suspended"
111  },
112  {
113      "ID": "GYRO",
114      "computationalMethod": "low_pass_filter",
115      "exception_period": "null",
116      "isOverrunning": false,
117      "overallEndTime": "07/09/2019\u05:44:15",
118      "overallStartTime": "07/09/2019\u05:44:15",
119      "recordId": 0,
120      "safetyRelatedMessage": false,
121      "sensor_address": "Bitonto,\u202270032,\u2022Italia",
122      "sensor_latitude": "4----",
123      "sensor_longitude": "1----",
124      "sensor_name": "ICM20610\u2022Gyroscope\u2022Sensor",
125      "sensor_reading_1": "-0,05\u00b5rad/s",
126      "sensor_reading_2": "-0,14\u00b5rad/s",
127      "sensor_reading_3": "-0,01\u00b5rad/s",
128      "sensor_resolution": "0.0010652645",
129      "sensor_type": "4",
130      "sensor_vendor": "Invensense",
131      "type": "null",
132      "validity_period": "100",
133      "validity_status": "suspended"
134  },
135  {
136      "ID": "LIGHT",
137      "computationalMethod": "low_pass_filter",
138      "exception_period": "null",
139      "isOverrunning": false,
140      "overallEndTime": "07/09/2019\u05:44:15",
141      "overallStartTime": "07/09/2019\u05:44:15",
142      "recordId": 0,
143      "safetyRelatedMessage": false,
144      "sensor_address": "Bitonto,\u202270032,\u2022Italia",
145      "sensor_latitude": "4----",
146      "sensor_longitude": "1----",
147      "sensor_name": "TMG399X\u2022RGB\u2022Sensor",
148      "sensor_reading_1": "4247,00\u2022lx",
149      "sensor_reading_2": "null",
150      "sensor_reading_3": "null",
151      "sensor_resolution": "1.0",
152      "sensor_type": "5",
153      "sensor_vendor": "AMS,\u2022Inc.\u2022",
154      "type": "null",
155      "validity_period": "100",
156      "validity_status": "suspended"
157  },
158  {
```

```
159     "ID": "TEMP",
160     "computationalMethod": "lowpassfilter",
161     "exception_period": "null",
162     "isOverrunning": false,
163     "overallEndTime": "07/09/2019\u00a005:44:15",
164     "overallStartTime": "07/09/2019\u00a005:44:15",
165     "recordId": 0,
166     "safetyRelatedMessage": false,
167     "sensor_address": "Bitonto,\u202a70032,\u202aItalia",
168     "sensor_latitude": "4----",
169     "sensor_longitude": "1----",
170     "sensor_reading_1": "null",
171     "sensor_reading_2": "null",
172     "sensor_reading_3": "null",
173     "type": "null",
174     "validity_period": "100",
175     "validity_status": "suspended"
176   },
177 ],
178   "startDate": "Sep\u202a7,\u202a2019\u00a017:44:10",
179   "time_precision": 0,
180   "type": "nullCar\u00a0Accident"
181 }
```

Listing 4.26: Payload del messaggio in formato JSON.

Capitolo 5

Conclusioni

Attraverso il percorso di progettazione ed implementazione affrontato nel Capitolo 3 e nel Capitolo 4, si è giunti all'implementazione di un prototipo che mostri le principali componenti di una cloud application nel mondo dell'Internet of Things.

Il contributo che si è cercato di dare con questo lavoro di tesi è stato quello di mostare il processo logico e le motivazioni che abbiano portato ad alcune scelte progettuali che fossero poi riproducibili non solo in una particolare istanza, ma in tutte quelle applicazioni legate al mondo dell'IoT che condividano le stesse necessità e che, a prescindere dal loro scopo, abbiano una struttura comune.

La struttura, appunto, che è stata progettata ed implementata è sostanzialmente riconducibile a tre componenti fondamentali:

- **Client Application:** Ovvero la applicazione e l'interfaccia che raccoglie e mostra i dati. In questo lavoro di tesi, l'interfaccia è un applicativo per smartphone Android. Tuttavia, le operazioni effettuate da questo applicativo sono eseguibili anche da qualsiasi altro dispositivo connesso in rete. Infatti, il suo compito sarà quello di raccogliere dati, inviare e ricevere dati attraverso Internet ed infine mostrare i dati. Si può pensare ad infinite altre soluzioni che siano in grado di riprodurre questi compiti e che quindi possano sostituire la client application implementata in questo lavoro di tesi per addattarsi a nuove necessità.
- **Server Application:** Ovvero la architettura del Server che è in ascolto e riceve i dati inviati dai client. In questo lavoro di tesi, per l'implementazione della Server application ci si è affidati ad un servizio esterno fornito da Amazon Web Services chiamato IoT Core. Anche in questo caso, i compiti che la server application deve svolgere sono, nella maggior parte, riproducibili da qualsiasi altro servizio cloud offerto da altre compagnie e perfino da servizi cloud implementati privatamente. I compiti ai quali dovrà assolvere una server application saranno infatti: autenticazione della comunicazione, implementazione di uno standard di comunicazione comune con i client, implementazione di una comunicazione publisher/subscriber con il client ed infine automatizzazione del processo di salvataggio dei dati ricevuti all'interno di un database.

- **Database Architecture and Data Analysis:** Strettamente collegata alla server application vi è anche la architettura del database nella quale memorizzare i dati raccolti ed eventualmente già filtrati dalla server application. Anche in questo caso, per l'implementazione di questo servizio, ci si è affidati ad un servizio esterno fornito da Amazon Web Services chiamato DynamoDB. Ma ancora, i compiti assolti da questo servizio sono riproducibili da altri servizi offerti da altre compagnie e perfino da databases gestiti localmente e privatamente. Infatti, il Database dovrà : consentire l'inserimento di nuovi elementi, consentire l' interrogazione di elementi già presenti, fornire delle interfacce per l'analisi dei dati contenuti al suo interno.

Proprio focalizzandosi su queste poche componenti dell'architettura di una cloud application, si è dato un esempio implementativo di ognuna per lo sviluppo del sistema di raccolta dati relativi ad eventi stradali.

Sebbene si renda necessaria una successiva fase di 'ingegnerizzazione' di questo prototipo, i suoi utilizzi possono avere un impatto molto forte sul settore dell'automotive, già soggetto ad una vera e propria rivoluzione. Con una applicazione simile a quella prototipata in questo lavoro di tesi, si potrebbe infatti avere una sorta di scatola nera in ogni veicolo che attraversa la rete stradale e che quindi raccoglie miliardi di dati in frazioni di secondo. Ulteriormente, l'unione di questi dati e la loro analisi attraverso algoritmi di Machine Learning, potrebbero supportare lo sviluppo e l'efficacia delle auto a guida autonoma. Tutto questo, al costo di uno smartphone.

Appendice A

Utilizzo del Protocollo DATEX II

In questo appendice sono mostrati i diagrammi delle classi relativi all'utilizzo del Protocollo DATEX II all'interno del sistema definito nel Capitolo 3.

I diagrammi delle classi di seguito mostrati sono stati ricavati da quelli progettati per lo standard DATEX. (http://d2docs.ndwcloud.nu/_static/umlmodel/v3.0/index.htm)
Si presta inoltre attenzione al fatto che i diagrammi delle classi mostrati in questo appendice non coprono il progetto nella sua interezza ma si intendono riferite alla sola implementazione dello standard DATEX II all'interno del progetto. Infine, i diagrammi delle classi di seguito esposti non comprendono lo standard DATEX II nella sua interezza, ma riguardano le sole componenti che si è ritenuto opportuno integrare nel progetto.

General Structure

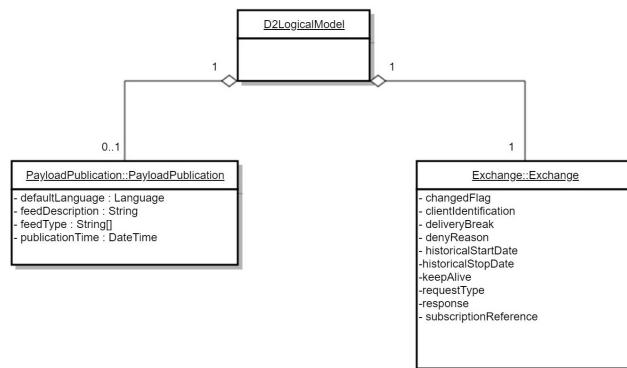


Figura A.1: Distinzione effettuata dallo Standard DATEX II tra metodo di comunicazione (*Exchange*) e formato del Payload (*PayloadPublication*)

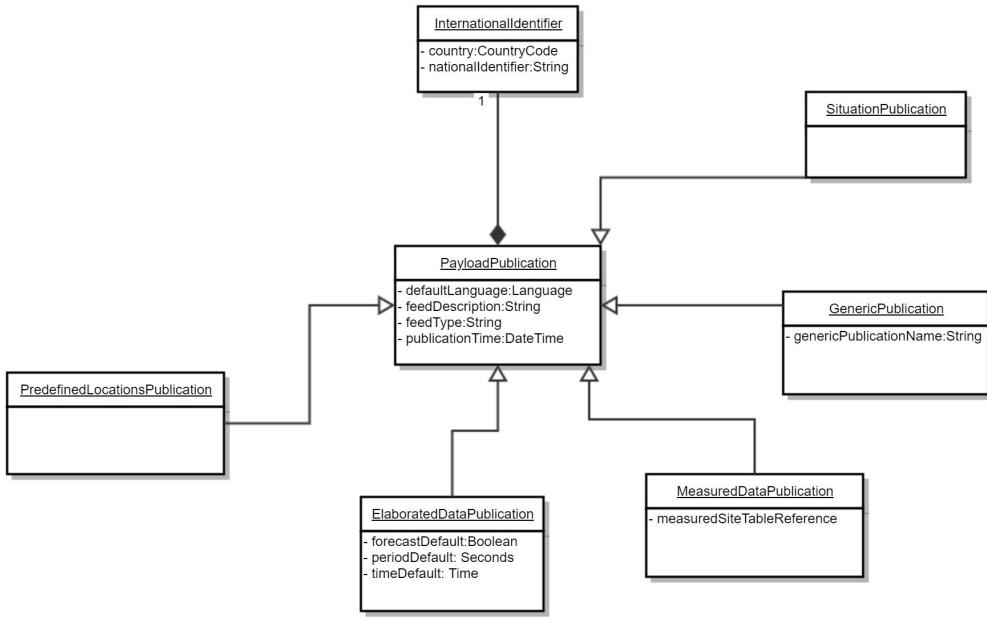


Figura A.2: Implementazione della Publication

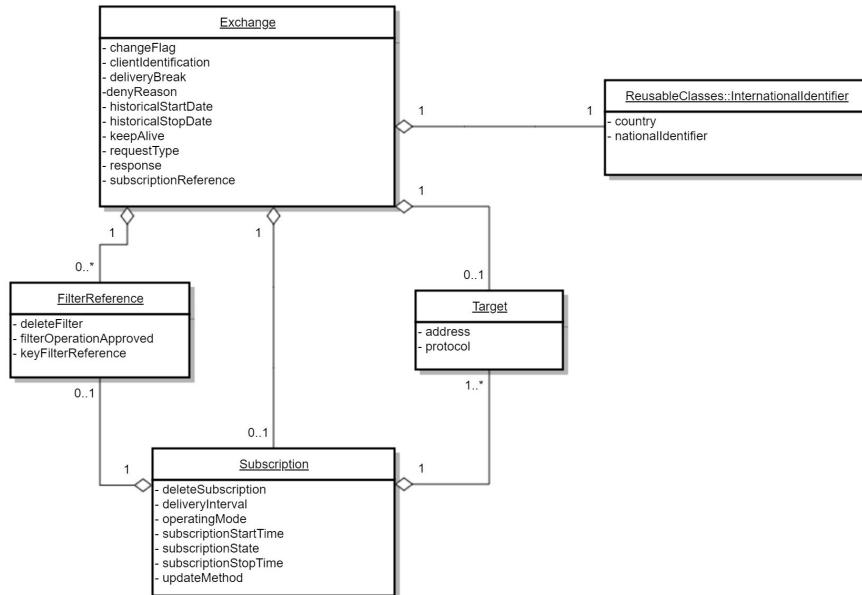


Figura A.3: Implementazione del' Exchange

Location

Elaborated Data Publication

Measured Data Publication

108

Situation Publication

Standard Extension

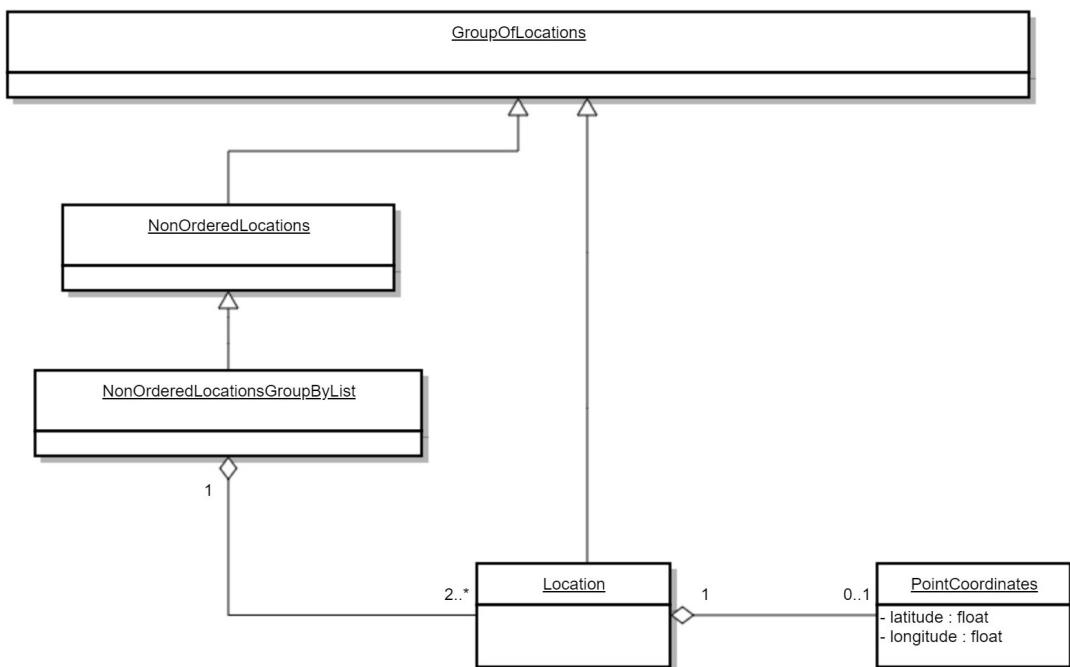


Figura A.4: Location reference

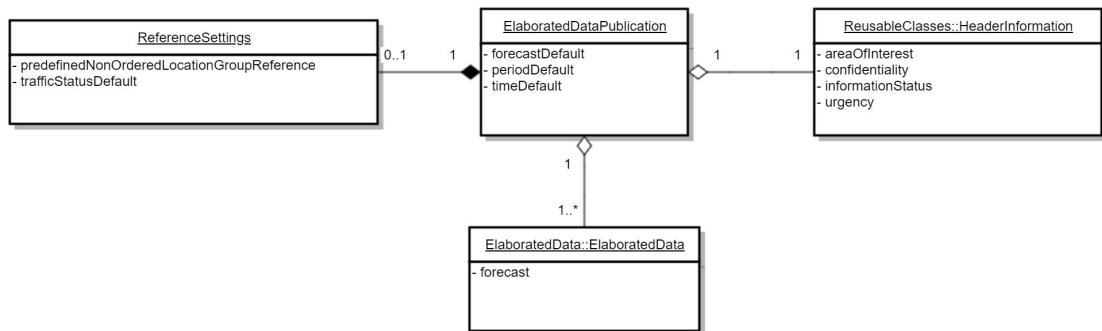


Figura A.5: Elaborated Data Publication structure

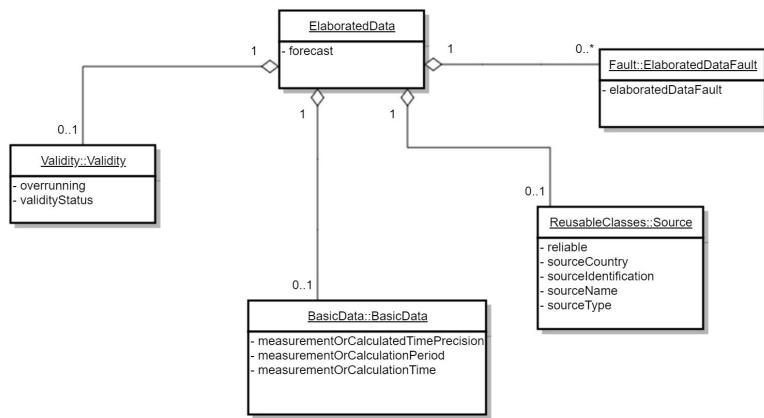


Figura A.6: Elaborated Data Implementation

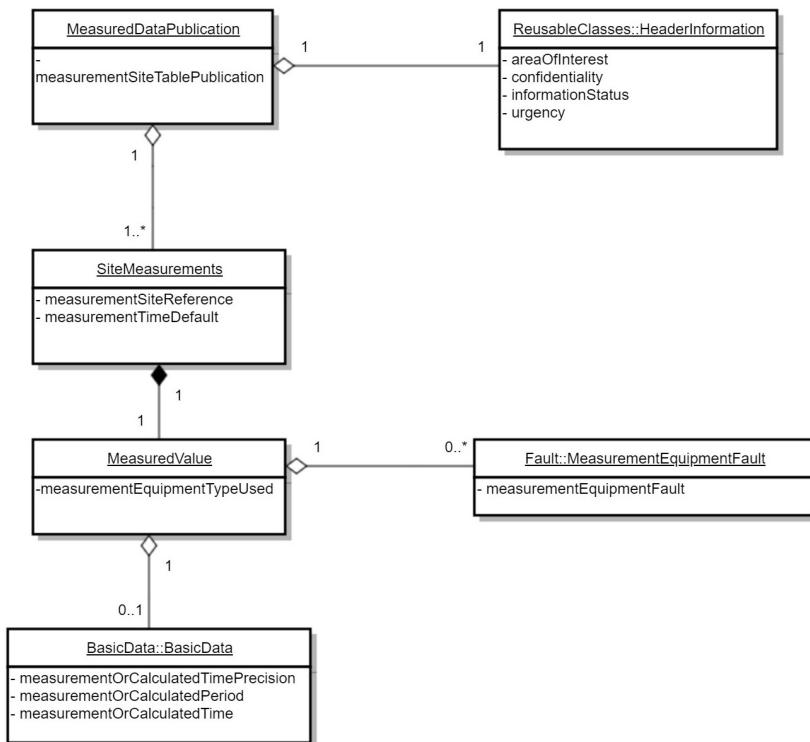


Figura A.7: Measured Data Structure

A – Utilizzo del Protocollo DATEX II

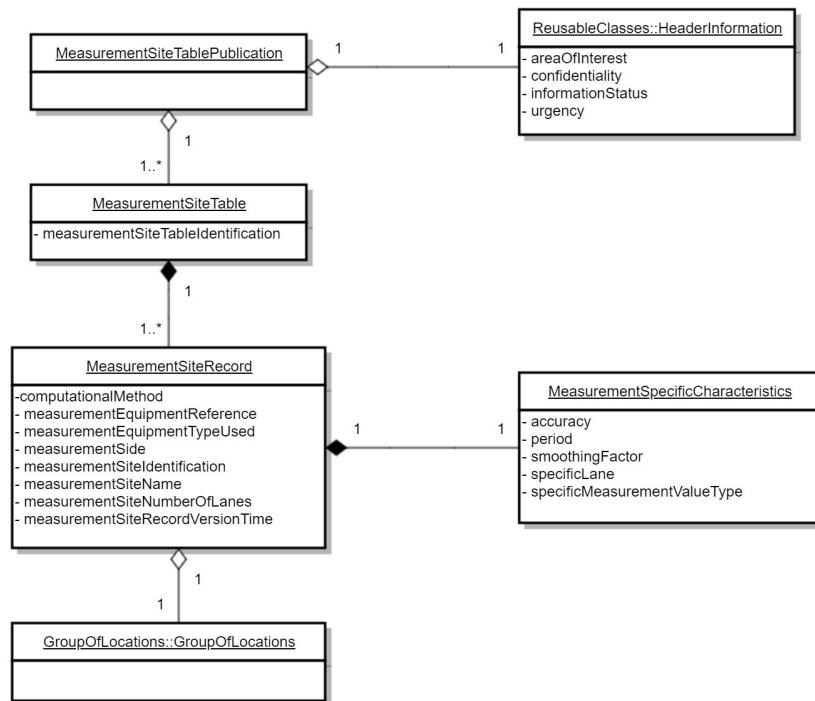


Figura A.8: Measurement Site Record Structure

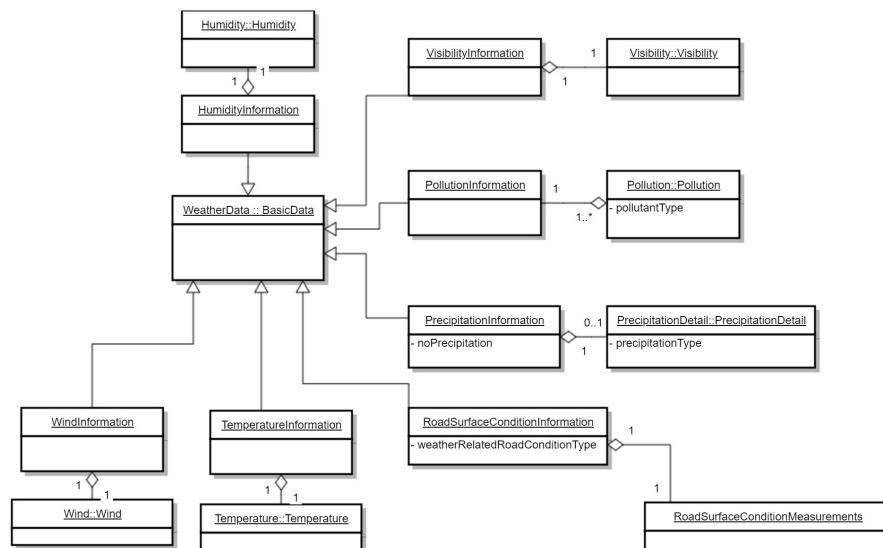


Figura A.9: Basic Data Default Implementation

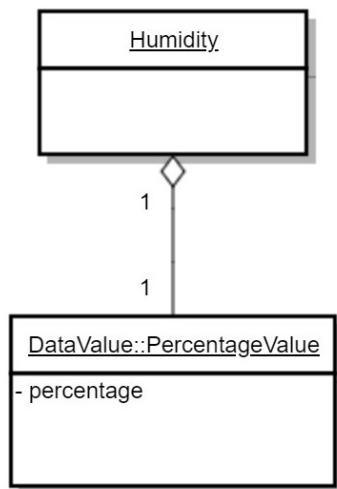


Figura A.10: Humidity Implementation

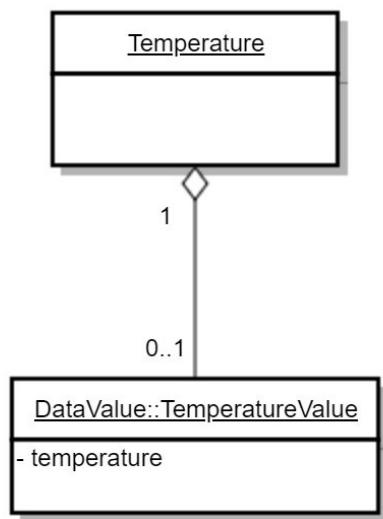


Figura A.11: Temperature Implementation

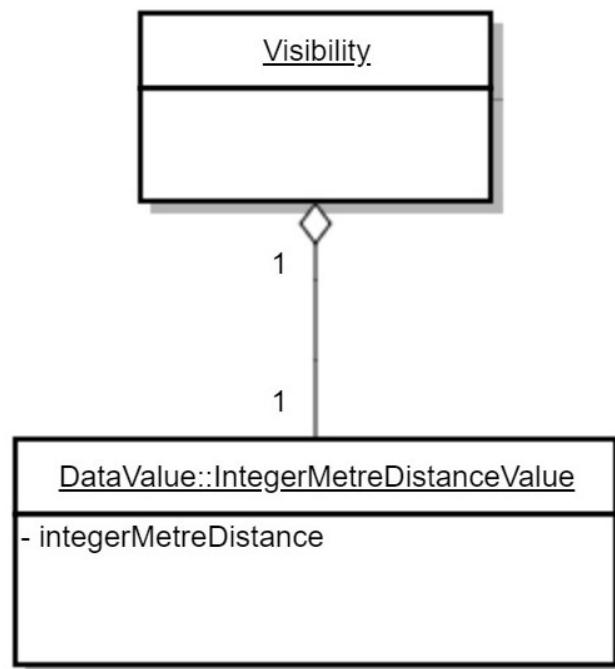


Figura A.12: Visibility Implementation

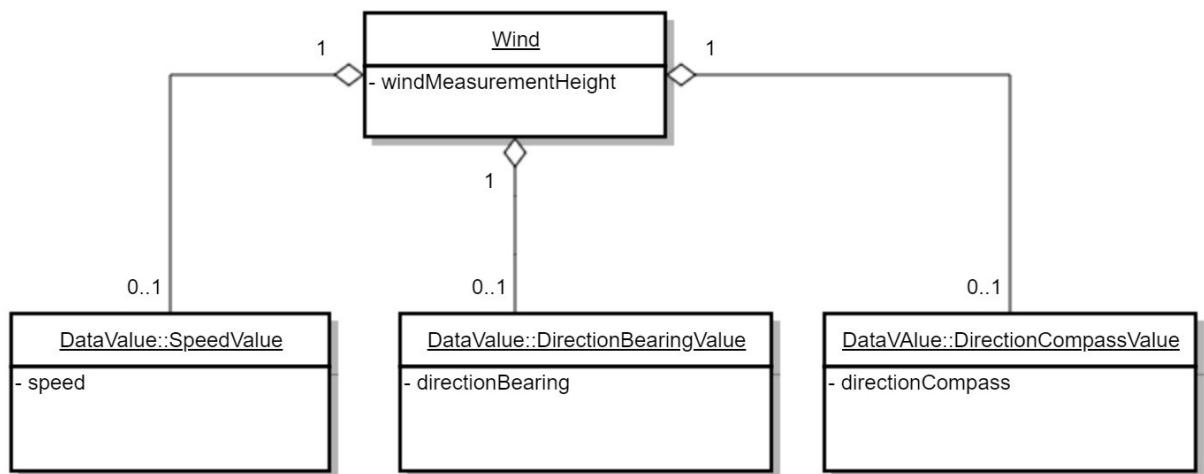


Figura A.13: Wind Implementation

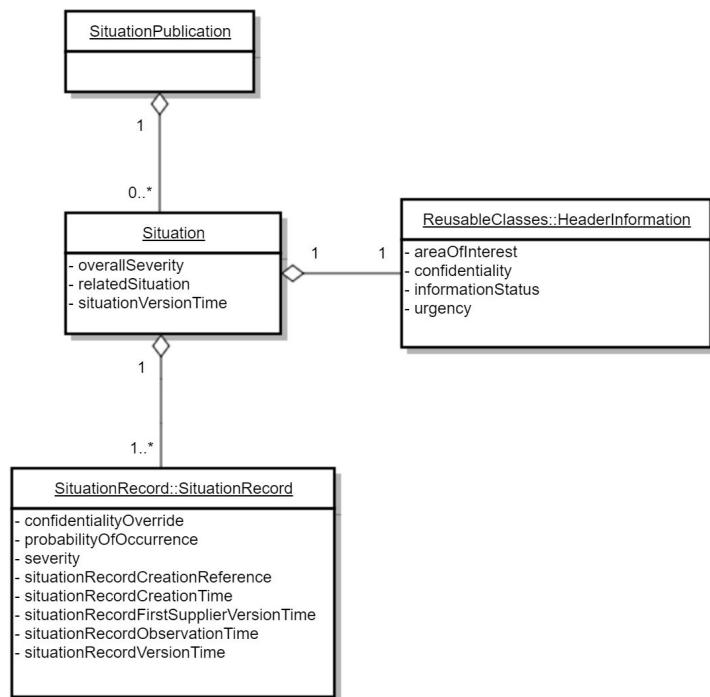


Figura A.14: Situation Publication Structure

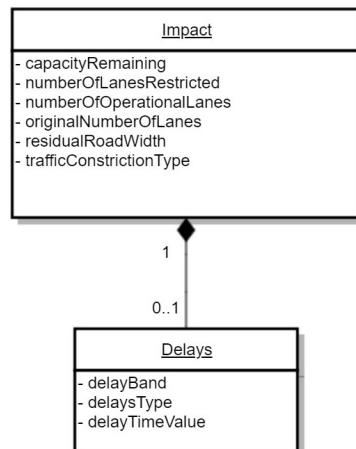


Figura A.15: Impact Implementation

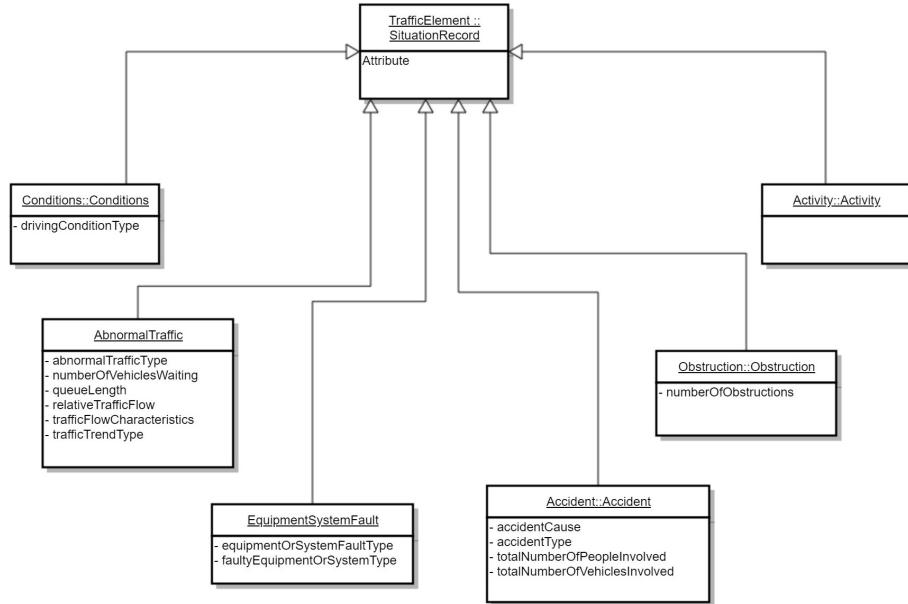


Figura A.16: Traffic Element Implementation

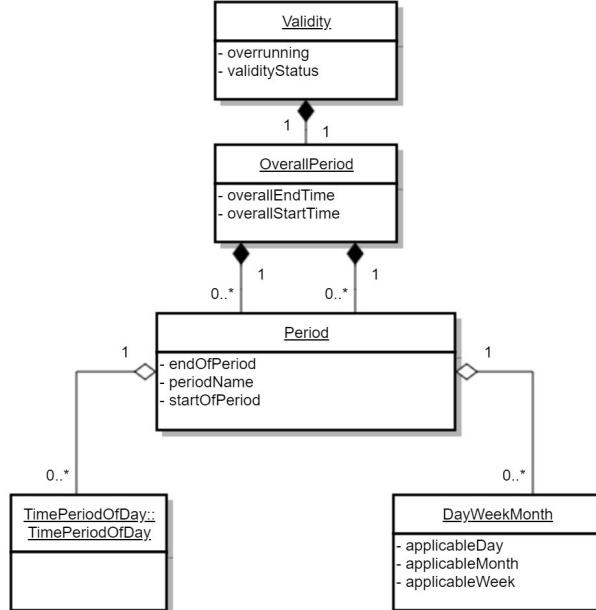


Figura A.17: Validity Implementation

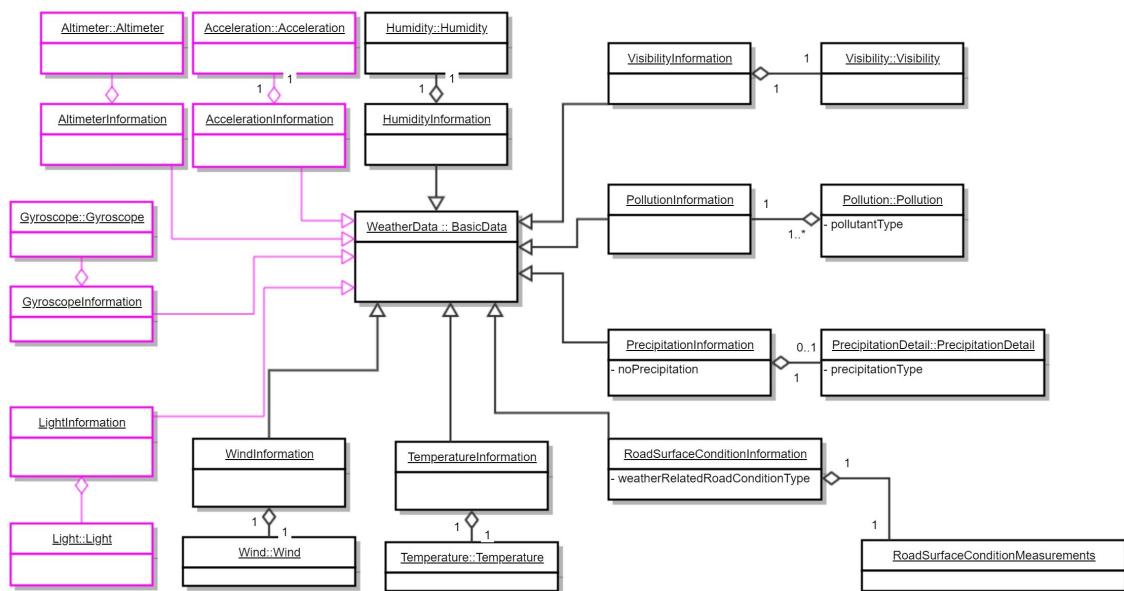


Figura A.18: Level B extension to the DATEX II Standard

Appendice B

Implementazione del Protocollo DATEX II

In questo appendice sono mostrati i dettagli implementativi del protocollo DATEX II all'interno della Applicazione Android sviluppata. Nella fattispecie, in riferimento ai diagrammi delle classi mostrati nell' Appendice A, si mostri ora l'implementazione delle classi ed il loro utilizzo all'interno della applicazione al fine di ottenere una formattazione dei dati in linea con il protocollo DATEX II. L'implementazione dei diagrammi delle classi mostrate nel Appendice A seguirà un processo analogo per l'implementazione di ogni classe. Nella fattispecie, sarà solo necessario tradurre ogni diagramma in una classe ed ogni collegamento tra i diagrammi in un collegamento tra le classi.

```
1 public class Publication{
2     private Exchange _exchange;
3     private Payload _payload;
4
5     public Payload getPayload(){
6         return this._payload;
7     }
8     public setPayload(Payload payload){
9         this._payload=payload;
10    }
11
12    public Exchange getExchange(){
13        return this._exchange;
14    }
15    public setPayload(Exchange exchange){
16        this._exchange=exchange;
17    }
18 }
```

Listing B.1: Implementazione della classe Publication che sarà il contenitore di tutto il pacchetto trasmesso.

```
1 public class Exchange{
2     ChangedFlagEnum changedFlag;
3     String clientIdentification;
4     boolean deliveryBreak;
5     DenyReasonEnum denyReason;
6     Date historicalStartDate;
7     Date historicalStopDate;
8     boolean keepAlive;
9     RequestTypeEnum requestType;
10    ResponseEnum response;
11    String subscriptionReference;
12
13    public ChangedFlagEnum getChangedFlag() {
14        return changedFlag;
15    }
16    public void setChangedFlag(ChangedFlagEnum changedFlag) {
17        this.changedFlag = changedFlag;
18    }
19    public String getClientIdentification() {
20        return clientIdentification;
21    }
22    public void setClientIdentification(String
23        clientIdentification) {
24        this.clientIdentification = clientIdentification;
25    }
26    public boolean isDeliveryBreak() {
27        return deliveryBreak;
28    }
29    public void setDeliveryBreak(boolean deliveryBreak) {
30        this.deliveryBreak = deliveryBreak;
31    }
32    public DenyReasonEnum getDenyReason() {
33        return denyReason;
34    }
35    public void setDenyReason(DenyReasonEnum denyReason) {
36        this.denyReason = denyReason;
37    }
38    public Date getHistoricalStartDate() {
39        return historicalStartDate;
40    }
41    public void setHistoricalStartDate(Date historicalStartDate) {
42        this.historicalStartDate = historicalStartDate;
43    }
44    public Date getHistoricalStopDate() {
45        return historicalStopDate;
46    }
47    public void setHistoricalStopDate(Date historicalStopDate) {
48        this.historicalStopDate = historicalStopDate;
}
```

```
49     public boolean isKeepAlive() {
50         return keepAlive;
51     }
52     public void setKeepAlive(boolean keepAlive) {
53         this.keepAlive = keepAlive;
54     }
55     public RequestTypeEnum getRequestType() {
56         return responseType;
57     }
58     public void setRequestType(RequestTypeEnum responseType) {
59         this.responseType = responseType;
60     }
61     public ResponseEnum getResponse() {
62         return response;
63     }
64     public void setResponse(ResponseEnum response) {
65         this.response = response;
66     }
67     public String getSubscriptionReference() {
68         return subscriptionReference;
69     }
70     public void setSubscriptionReference(String
71         subscriptionReference) {
72         this.subscriptionReference = subscriptionReference;
73     }
```

Listing B.2: Implementazione della classe Exchange che conterrà informazioni utili alla comunicazione

```
1  public class PayloadPublication{
2      Language defaultLanguage;
3      MultilingualString feedDescription;
4      String[] feedType;
5      Date publicationTime;
6      ElaboratedDataPublication elaboratedDataPublication;
7      MeasuredDataPublication measuredDataPublication;
8      SituationRecord situationRecord;
9      ElaboratedData elaboratedData;
10
11     public Language getDefaultLanguage() {
12         return defaultLanguage;
13     }
14     public void setDefaultLanguage(Language defaultLanguage) {
15         this.defaultLanguage = defaultLanguage;
16     }
17     public MultilingualString getFeedDescription() {
18         return feedDescription;
19     }
```

```
20     public void setFeedDescription(MultilingualString
21         feedDescription) {
22         this.feedDescription = feedDescription;
23     }
24     public String[] getFeedType() {
25         return feedType;
26     }
27     public void setFeedType(String[] feedType) {
28         this.feedType = feedType;
29     }
30     public Date getPublicationTime() {
31         return publicationTime;
32     }
33     public void setPublicationTime(Date publicationTime) {
34         this.publicationTime = publicationTime;
35     }
36     public ElaboratedDataPublication
37         getElaboratedDataPublication() {
38         return elaboratedDataPublication;
39     }
40     public void
41         setElaboratedDataPublication(ElaboratedDataPublication
42             elaboratedDataPublication) {
43         this.elaboratedDataPublication = elaboratedDataPublication;
44     }
45     public MeasuredDataPublication getMeasuredDataPublication() {
46         return measuredDataPublication;
47     }
48     public void setMeasuredDataPublication(MeasuredDataPublication
49         measuredDataPublication) {
50         this.measuredDataPublication = measuredDataPublication;
51     }
52     public SituationRecord getSituationRecord() {
53         return situationRecord;
54     }
55     public void setSituationRecord(SituationRecord
56         situationRecord) {
57         this.situationRecord = situationRecord;
58     }
59 }
```

Listing B.3: Implementazione della classe PayloadPublication che conterrà tutti i dati prodotti dalla applicazione

In questo modo si sono visti gli esempi implementativi della Figura A.1 dell' Appendice A. Allo stesso modo, sono state implementate tutte le altre classi appartenenti al diagramma delle classi dello Standard DATEX II e che quindi sono omesse.

Nella creazione del messaggio seguendo lo standard DATEX II, si sono quindi compilati tutti i campi della classe Payload, con i dati raccolti dalla applicazione e sarà quindi proprio la classe Payload ad essere serializzata e quindi convertita in formato JSON ed XML. Nella fattispecie, viene di seguito mostrato l'output della serializzazione della classe Publication in formato XML.

```
1 <?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
2 <d2LogicalModel modelBaseVersion="2"
  xmlns="http://datex2.eu/schema/2/2_0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://datex2.eu/schema/2/2_0
    http://www.datex2.eu/schema/2/2_3/DATEXIISchema_2_2_3.xsd">
3 <exchange>
4   <supplierIdentification>
5     <country>it</country>
6     <nationalIdentifier>IT</nationalIdentifier>
7   </supplierIdentification>
8 </exchange>
9 <payloadPublication xsi:type="MeasuredDataPublication" lang="it">
10  <publicationTime>22/06/2019 11:02:35</publicationTime>
11  <feedDescription></feedDescription>
12  <feedType>, Obstruction</feedType>
13  <defaultLanguage>it</defaultLanguage>
14  <publicationCreator>
15    <country>it</country>
16    <nationalID>it</nationalID>
17  </publicationCreator>
18 </payloadPublication>
19 <measuredValue>
20   <measurementEquipmentTypeUsed>Goldfish 3-axis
      Accelerometer</measurementEquipmentTypeUsed>
21   <measurementEquipmentTypeUsed>Goldfish Pressure
      sensor</measurementEquipmentTypeUsed>
22   <siteMeasurement>
23     <measurementSiteReference>it-35824005111110</measurementSiteReference>
24     <measurementTimeDefault>22/06/2019
      11:02:35</measurementTimeDefault>
25   </siteMeasurement>
26   <basicData>
27     <measurementOrCalculationTimePrecision>1us
28     </measurementOrCalculationTimePrecision>
29     <measurementOrCalculationPeriod>6467</measurementOrCalculationPeriod>
30   </basicData>
31 </measuredValue>
32 <measurementSiteRecord>
33   <computationalMethod>low pass filter</computationalMethod>
```

```
34      <measurementEquipmentReference>The Android Open Source
35          Project</measurementEquipmentReference>
36      <measurementSpecificCharacteristics>
37          <accuracy>2.480159E-4</accuracy>
38          <period>100</period>
39      </measurementSpecificCharacteristics>
40      <groupOfLocations>
41          <tpegPointLocation xsi:type="ns1:TPEGSimplePoint">
42              <point xsi:type="ns1:TPEGNonJunctionPoint">
43                  <pointCoordinates>
44                      <latitude>37-----</latitude>
45                      <longitude>-122-----</longitude>
46                  </pointCoordinates>
47                  <name>
48                      <descriptor>
49                          <value>Mountain View, 94043, United States</value>
50                      </descriptor>
51                  </name>
52              </point>
53          </tpegPointLocation>
54      </groupOfLocations>
55      <weatherData></weatherData>
56      <trafficElement>
57          <abnormalTraffic>
58              <abnormalTrafficType>, Obstruction,
59                  Obstruction</abnormalTrafficType>
60          </abnormalTraffic>
61      </trafficElement>
62  </measurementSiteRecord>
63  <measurementSiteRecord>
64      <computationalMethod>low pass filter</computationalMethod>
65      <measurementEquipmentReference>The Android Open Source
66          Project</measurementEquipmentReference>
67      <measurementSpecificCharacteristics>
68          <accuracy>1.0</accuracy>
69          <period>100</period>
70      </measurementSpecificCharacteristics>
71      <groupOfLocations>
72          <tpegPointLocation xsi:type="ns1:TPEGSimplePoint">
73              <point xsi:type="ns1:TPEGNonJunctionPoint">
74                  <pointCoordinates>
75                      <latitude>37-----</latitude>
76                      <longitude>-122-----</longitude>
77                  </pointCoordinates>
78                  <name>
79                      <descriptor>

```

```
80      </point>
81      </tpegPointLocation>
82  </groupOfLocations>
83  <weatherData></weatherData>
84  <trafficElement>
85      <abnormalTraffic>
86          <abnormalTrafficType>, Obstruction , Obstruction ,
87          Obstruction</abnormalTrafficType>
88      </abnormalTraffic>
89  </trafficElement>
90  </measurementSiteRecord>
91 </d2LogicalModel>
```

Listing B.4: Esempio di output della applicazione in formato XML in rispetto dello standard DATEX II

Bibliografia

- [1] D.Uckelmann et al. *Architecting the Internet of Things*. first edition. Springer, 2011.
- [2] O.Hersent et al. *The Internet of Things:Key Applications and Protocols*. second edition. Wiley, 2012.
- [3] Apple. «Apple Developer». In: (2019). URL: <https://developer.apple.com/documentation/>.
- [4] ARCEP. «Préparer la révolution de l'Internet des objets». In: *Le Livre Blanc* (2016).
- [5] F. Attivissimo. *Corso di Sensori e Trasduttori, Slide del Docente*. Politecnico di Bari, 2017.
- [6] M. Losciale; P. Boccadoro; G. Piro; G. Ribezzo; L.A. Grieco; N. Blefari-Melazzi. «A Novel ICN-Based Communication Bus for Intelligent Transportation Systems». In: *IEEE* (2018).
- [7] <https://commons.wikimedia.org/w/index.php?curid=16949432> Chunte7 Opera propria.
- [8] European Commission. «Directive 2010/40/EU: Framework for the Deployment of Intelligent Transportation Systems». In: *Official Journal of the European Union* (2010).
- [9] J. Toldinas; B. Lozinskis; E. Baranauskas; A. Dobrovolskis. «MQTT Quality of Service versus Energy Consumption». In: *IEEE* (2019).
- [10] M.R. Palattella; M. Dohler; A. Grieco; G. Rizzo; J. Torsner; T. Engel e L. Ladid. «Internet of Things in the 5G Era: Enablers, Architecture, and Business Models». In: *IEEE Journal on Selected Areas in Communications* 34 (2016), pp. 510–527.
- [11] Dave Evans. «The Internet of Things, how the next evolution of the Internet is changing everything». In: *Cisco Internet Business Solutions Group (IBSG)* (2011).
- [12] H. Li; X. Zhang; J. Fan. «The safety analysis of IPSec based on IPv6 protocol». In: *IEEE* (2010).
- [13] A. Grieco. *Corso di Internet of Things, Slide del Docente*. Politecnico di Bari, 2017.
- [14] McKinsey Global Institute. «THE INTERNET OF THINGS: MAPPING THE VALUE BEYOND THE HYPE». In: (2015).

- [15] L. Wei-Feng; C. Wei; H. Jian. «Research on a DATEX II based Dynamic Traffic Information Publish Platform». In: *IEEE* 3 (2008), pp. 412–416.
- [16] C. Gomez; J.Oller e J. Paradells. «Overview and Evaluation of bluetooth low energy: An emerging low-power wireless technology». In: *Sensors* 12 (2012), pp. 11734–11753.
- [17] N. Tantitharanukul; K. Osathanunkul; K. Hantrakul; P. Pramokchon; P. Khoenkaw. «MQTT-Topics Management System for Sharing of Open Data». In: *IEEE* (2017).
- [18] Google LLC. «Android Developer». In: (2019). URL: <https://developer.android.com/>.
- [19] MathWorks. «Developing an IoT Analytics System with MATLAB, Machine Learning, and ThingSpeak». In: (). URL: <https://it.mathworks.com/company/newsletters/articles/developing-an-iot-analytics-system-with-matlab-machine-learning-and-thingspeak.html>.
- [20] Andrea Detti ; Michele Orru ; Riccardo Paolillo ; Giulio Rossi ; Pierpaolo Loreti ; Lorenzo Bracciale ; Nicola Belfari Melazzi. «Application of Information Centric Networking to NoSQL databases: The spatio-temporal use case». In: *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)* (2017).
- [21] Michele Orru ; Riccardo Paolillo ; Andrea Detti ; Giulio Rossi ; Nicola Belfari Melazzi. «Demonstration of OpenGeoBase: The ICN NoSQL spatio-temporal database». In: *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)* (2017).
- [22] M.H. Asghar; N. Mohammadzadeh. «Design and Simulation of energy efficiency in node based on MQTT protocol in Internet of Things». In: *IEEE* (2016).
- [23] B. Kantarci; H.T. Mouftah. «Mobility-aware trustworthy crowdsourcing in cloud-centric Internet of Things». In: *IEEE* (2014).
- [24] M. Mongiello ; F. Nocera. *Corso di Ingegneria del Software, Slide del Docente*. Politecnico di Bari, 2017.
- [25] M. Siekkinen; M. Hiienkarri; J.K. Nurminen e J. Nieminen. «How low energy is bluetooth low energy? Comparative measurement with zigbee/802.15.4». In: *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)* (2012), pp. 232–237.
- [26] OECD. «IoT measurement and applications». In: *OECD Digital Economy Papers* 271 (2018). URL: <https://doi.org/10.1787/35209dbf-en>.
- [27] S. Gisdakis; V. Manolopoulos; S. Tao; A. Rusu; P. Papadimitratos. «Secure and Privacy-Preserving Smartphone-Based Traffic Information Systems». In: *IEEE Transactions on Intelligent Transportation Systems* 16 (2014), pp. 1428–1438.
- [28] Bruce W. DeKock; Kevin L. Russell; Richard J. Qian. «System for providing traffic information». In: *Traffic Information LLC* (2002). URL: <https://patents.google.com/patent/US6466862B1/en>.

- [29] Tinati; Ramine; Madaan; Aastha e Hall; Wendy. «The Role of Crowdsourcing in the Emerging Internet-Of-Things». In: *IEEE* (2017).
- [30] M. Research. «The need for low cost, high reach, wide area connectivity for the internet of things. A mobile network operator's perspective». In: *White Paper* (2014).
- [31] S. Ziegler; C. Crettaz; M. Hazan; P. Alexandrou; G. Filios; S. Nikoletseas; T.P. Raptis; X. Ziouvelou; F. McGroarty; A. Rankov; S. Krco; C. M. Angelopoulos; O. Evanelatos; M. Karagiannis; J. Rolim e N. Loumis. *Combining Internet of Things and Crowdsourcing for Pervasive Research and End-user Centric Experimental Infrastructures*. IoT Lab.
- [32] Andrea Detti ; Nicola Blefari Melazzi ; Michele Orru ; Riccardo Paolillo ; Giulio Rossi. «OpenGeoBase: Information Centric Networking Meets Spatial Database Applications». In: *2016 IEEE Globecom Workshops (GC Wkshps)* (2017).
- [33] G. Piro; G. Boggia; C. Campolo; L.A. Grieco; A. Molinaro; G. Ruggeri. «Wazing into the Crystal Ball: When the Future Internet Meets the Mobile Clouds». In: *IEEE Transactions on Cloud Computing* 7 (2016), pp. 210–223.
- [34] M. Savino. *Fondamenti di scienza delle misure*. La Nuova Italia Scientifica, 1992.
- [35] V.L. Kalyani; D. Sharma. «IoT: Machine to Machine (M2M), Device to Device (D2D) Internet of Everything (IoE) and Human to Human (H2H): Future of Communication». In: *Journal of Management Engineering and Information Technology (JMEIT)* 2 (2015).
- [36] I. Sommerville. *Ingegneria del Software*. Pearson, 2007.
- [37] Stuart Taylor. «The Next Generation of the Internet, Revolutionating the Way we Work, Live, Play and Learn». In: *Cisco Internet Business Solutions Group (IBSG)* (2013).
- [38] A. Rauniyar; P. Engelstad; B. Feng; D. Van Thanh. «Crowdsourcing-Based Disaster Management using Fog Computing in Internet of Things Paradigm». In: *IEEE* (2017).
- [39] Datex II Website. «Datex II Standard». In: (2019). URL: <https://datex2.eu/datex2/about>.
- [40] l'enciclopedia libera. Wikipedia.
- [41] Wikistats. «Wikimedia Statistics». In: (2019). URL: <https://stats.wikimedia.org/>.
- [42] A. Jian; G. Xiaolin; Y. Jianwei; S. Yu; H. Xin. «Mobile Crowd Sensing for Internet of Things: A Credible Crowdsourcing Model in Mobile-Sense Service». In: *IEEE* (2015).
- [43] P. Yadav; A. Mittal; H. Yadav. «IoT: Challenges and Issues in Indian Perspective». In: *IEEE* (2018).