Capitolo 1

Contesto ed obiettivi

In questo capitolo sarà introdotto il contesto in cui ci si trova ad operare e gli obiettivi del progetto.

1.1 DATABENC

Il progetto DATABENC (http://www.databenc.it) (Distretto Ad Alta Tecnologia per i BENi Culturali) è nato in Campania, grazie all'Università degli Studi di Napoli Federico II e all'Università di Salerno, con l'intento di stabilire una programmazione strategica per valorizzare i beni culturali, il patrimonio ambientale e il turismo. DATABENC ha l'obiettivo di costituire, tra università, centri di ricerca, imprese e amministrazioni comunali presenti sul territorio, una rete che focalizzi le proprie risorse su di un programma di alta tecnologia al fine di creare nuove realtà imprenditoriali (spin-off e start-up), nuove figure professionali, percorsi di alta formazione qualificati, valorizzazione delle conoscenze (brevetti, know-how). Il distretto è un contenitore in cui riunire ed integrare itinerari eterogenei di ricerca, formazione ed innovazione, con l'obiettivo comune della tutela e della valorizzazione del patrimonio culturale campano inteso in senso esteso: territori, siti, beni e attività.

1.1.1 Le linee di intervento

Gli ambiti di intervento del Distretto si sviluppano su tre linee portanti:

- Conoscenza integrata: la prima forma di tutela di un bene è nella conoscenza e, per tale motivo, è necessario realizzare un esauriente sistema di salvaguardia cognitiva del patrimonio culturale;
- Monitoraggio diagnostico: ai fini della tutela di un bene risulta indispensabile il monitoraggio diagnostico inteso in senso ampio, che non si limiti solo alla verifica dell'integrità materiale del bene stesso ma si estenda anche all'area in cui il bene è inserito o alle dinamiche turistiche che lo coinvolgono;
- Fruizione sostenibile: un aspetto fondamentale del bene culturale è quello del suo utilizzo. Perciò risulta indispensabile conseguire un utilizzo sostenibile del patrimonio culturale.

In sintesi, DATABENC ha come obiettivo l'introduzione di una nuova ottica con cui affrontare il grave problema della tutela e della valorizzazione del patrimonio culturale, e la creazione di un sistema che faccia della Campania una regione dell'innovazione e un centro di produzione e diffusione di cultura capace di attrarre capitali economici e, soprattutto, capitali umani.

1.2 Ancitel

Ancitel S.p.A. (http://www.ancitel.it) è la principale società dell'ANCI - Associazione Nazionale Comuni Italiani - e da 25 anni supporta gli enti locali nella gestione di tutti i processi di innovazione. Dalla sua fondazione, avvenuta nel 1987, Ancitel affianca le pubbliche amministrazioni locali con un'ampia rete di servizi e progetti ideati per rispondere alle loro esigenze operative quotidiane. In quanto partner dei Comuni, Ancitel agisce ed

opera ogni giorno come centro di competenza per fornire loro soluzioni e strumenti pensati per facilitarne e supportarne l'azione quotidiana ed affrontare le sfide dell'innovazione. Grazie ad una profonda conoscenza delle dinamiche interne alla Pubblica Amministrazione, Ancitel ha conseguito notevoli capacità di ascolto, dialogo ed intervento. Per questo uno dei ruoli fondamentali dell'azienda è quello di promuovere e favorire lo scambio delle informazioni tra gli enti pubblici, centrali e locali. Grazie alle forti competenze e professionalità e alla capacità di valorizzare le esperienze locali e coinvolgere trasversalmente l'intera rete dei Comuni Italiani, Ancitel è stata scelta come partner affidabile dai principali organismi istituzionali italiani, quali Camera dei Deputati, Presidenza del Consiglio dei Ministri, Ministero dell'Ambiente, Ministero dell'Interno, Ministero del Lavoro e delle Politiche Sociali, Ministero dello Sviluppo Economico, Autorità per l'energia elettrica e il gas. All'interno del distretto DATABENC, Ancitel è uno dei principali soci.

1.3 Fruizione informazioni culturali e turistiche in mobilità

Nell'ambito dei beni culturali, vi sono diversi progetti che coinvolgono enti locali, grandi, piccole e medie imprese ed istituti universitari. L'obiettivo comune è quello di sviluppare strumenti di valorizzazione e capitalizzazione dell'offerta culturale e delle risorse ambientali, al fine di promuovere e commercializzare l'offerta turistica da parte delle P.A. locali.

Si rende spesso necessaria la definizione e lo sviluppo di una piattaforma abilitante su cui basare servizi per l'offerta culturale. Una piattaforma che ponga al centro le informazioni da offrire agli utenti, che renda facile ed accessibile la fruizione di esse e che possieda validi strumenti per la conservazione e la salvaguardia di tali informazioni. Le stesse informazioni, inoltre, vanno validate e standardizzate, in modo da consentire facilmente l'estrazione e la catalogazione automatica, l'analisi e la correlazione di esse attraverso motori semantici. Tra i progetti già in corso, si può citare il campano *OR.C.HE.S.T.R.A.* (ORgani-

zation of Cultural HEritage and Smart Tourism and Real-time Accessibility), il cui partner universitario è l'Università Federico II di Napoli, che ha come obiettivo quello di realizzare un insieme di soluzioni tecnologiche orientate alla valorizzazione del patrimonio culturale, materiale e immateriale, del centro storico di Napoli.

1.4 Gli scavi di Paestum

Paestum, cittadina della provincia di Salerno, presenta un'area archeologica molto vasta, riconosciuta dall'UNESCO come patrimonio dell'umanità dal 1988. All'interno di quest'area sono presenti 3 importanti templi, oltre ad altre interessanti testimonianze storico-artistiche. Il *Tempio di Hera* (anche noto come *Basilica* a causa della quasi totale sparizione dei muri della cella, del frontone e della trabeazione) fu dedicato ad Hera, moglie di Zeus, risulta essere il più antico dei 3 templi, essendo stato eretto intorno al 550 a.C.. Il secondo tempio è il *Tempio di Poseidone* (fig. 1.4.1) e incarna il più riuscito esempio

Figura 1.4.1: Tempio di Poseidone

di architettura greca in Occidente. Costruito intorno al 450 a.C. seguendo lo stile architet-

tonico dorico classico (lo stesso del Partenone di Atene) risulta orientato da ovest ad est. Il terzo ed ultimo grande tempio è il *Tempio di Cerere*, costruito verso la fine del VI sec. a.C. nel punto più alto della città. Altri punti d'interesse minore sono il *Templio Italico* (dedicato a Giove, Giunone e Minerva, era posto su un alto basamento al cui centro si ergeva un imponente altare e vi si poteva accedere solo dal lato a Sud tramite un'ampia gradinata), il *Foro* (edificato dai romani sulla vecchia Agorà Greca), il *Boleuterion* (detto anche Teatro Greco, fu costruito originariamente per ospitare le riunioni del massimo Consiglio della città ed aveva forma circolare prima di essere ridotto nelle dimensioni dai romani prima sul lato occidentale e poi sul lato settentrionale), l'*Anfiteatro*(destinato agli spettacoli dei Gladiatori e risalente al primo secolo D.C.), il *Ginnasio* (dotato di piscina per gare di nuoto) e il *Sacello Ipogeico* (costruzione sotterranea rinvenuta nel 1954 di cui è ignota la finalità, si suppone che si tratti un tempio sotterraneo dedicato alla dea della fecondità e fertilità, oppure un cenotafio -una tomba simbolica- realizzata per onorare il fondatore della città).

1.5 Obiettivi

Lo scopo di questa tesi è quello di sviluppare, nel contesto di DATABENC, un'applicazione mobile che assista i turisti nella visita del sito archeologico degli scavi di Paestum. Il turista potrà, sul suo dispositivo mobile, avere a disposizione una mappa su cui poter visualizzare la propria posizione e quella dei punti di interesse presenti nel sito. Il turista potrà, inoltre, ottenere informazioni approfondite e materiale multimediale su tali punti di interesse e, all'occorrenza, consutarlo offline. Sono previsti, inoltre diversi itinerari e percorsi guidati all'interno del sito.

Capitolo 2

Architettura e tecnologie

In questo capitolo saranno introdotte le architetture e le tecnologie utilizzate per la realizzazione dell'applicazione. Dopo una presentazione dell'architettura di riferimento, saranno introdotti i principali sistemi operativi mobili, utilizzati per la piattaforma client, e, successivamente, le tecnologie di utilizzo per l'applicazione. Sarà poi descritto l'ambiente server, con particolare riferimento ai Web Services ed al paradigma REST. Infine, saranno introdotte la tecnologia .NET, ovvero il framework di sviluppo dell'applicazione, e la metodologia Agile, metodologia di sviluppo software utilizzata.

2.1 Architettura Client/Server

L'*architettura client/server* indica un'architettura software che è costituita da due moduli integrati ma distinti, residenti generalmente su calcolatori diversi.
In una rete informatica ci si riferisce ai computer ad essa collegati come host o terminali, perché ospitano programmi di livello applicativo. Gli host, o nodi, sono ulteriormente suddivisi in due categorie, client e server. La condivisione delle risorse, che generalmente avviene attraverso la rete, tra i client e i server definisce l'architettura client/server (fig. 2.1.1).

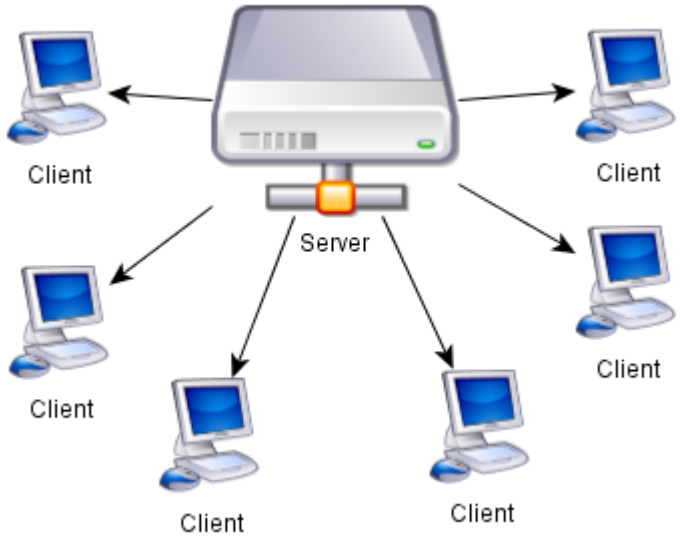

Figura 2.1.1: Architettura Client/Server.

Il server svolge le operazioni necessarie per realizzare un servizio, mentre il client può effettuare alcune operazioni e quindi richiede un terminale con capacità elaborative. Tipicamente il client gestisce la porzione d'interfaccia utente dell'applicazione, verifica i dati inseriti e provvede ad inviare al server le richieste formulate dall'utente. Inoltre gestisce le risorse locali, come la tastiera, il monitor, la CPU, e le periferiche. L'affermazione di questo modello è legata alla disponibilità di reti locali a basso costo ed alla diffusione della rete Internet, in cui i servizi seguono tale struttura. In un ambiente client/server, sul computer client è in esecuzione un software applicativo detto programma client, il quale richiede e riceve un servizio da un programma server che viene eseguito sul computer server. Il programma client e il programma server interagiscono tra loro attraverso comunicazioni, ovvero scambiandosi messaggi attraverso la rete informatica. Quando un client si connette ad un server, gli invia una richiesta; il server, essendo in ascolto, la intercetta e l'interpreta effettuando le elaborazioni necessarie al suo soddisfacimento e restituisce un risultato sotto forma di risposta al client il quale lo visualizza all'utente. Le applicazioni operanti in rete hanno protocolli di livello di applicazione che definiscono il formato e l'ordine dei messaggi scambiati tra i processi, così come definiscono le azioni

da intraprendere alla trasmissione o alla ricezione dei messaggi. A livello di applicazione, un protocollo definisce come i processi delle applicazioni, che funzionano su differenti terminali, si scambiano i messaggi, in particolare il protocollo specifica:

- i tipi di messaggi scambiati, per esempio, messaggi di richiesta e messaggi di risposta;
- la sintassi dei vari tipi di messaggio, per esempio i campi del messaggio e come questi ultimi vengono caratterizzati;
- la semantica dei campi, cioè il significato dell'informazione nei campi;
- le regole per determinare quando e come un processo invia messaggi o risponde a messaggi.

Prima di parlare dell'evoluzione del client/server è necessario fare una breve introduzione sulle componenti fondamentali di un'architettura di questo tipo.

Si distinguono tre parti principali di tale architettura: i dati sono la parte legata alla gestione delle informazioni persistenti, cioè quelle informazioni che si vuole mantenere nel tempo; la logica applicativa è rappresentata dai programmi veri e propri, ovvero quella parte del software che viene utilizzato per effettuare le operazioni per cui sono stati scritti; la presentazione, è quella parte del software che permette la comunicazione con l'utente. L'architettura client/server ha avuto una evoluzione nel tempo (fig. 2.1.2) che l'ha portata attraverso vari stadi analizzati di seguito:

- *Time-sharing*: con questa soluzione, sul server sono presenti tutte e tre le parti fondamentali dell'architettura: dati, logica e presentazione. Il client è rappresentato da un terminale cosiddetto stupido il quale permette l'interazione con il server, senza avere una sua capacità elaborativa;
- *Client presentation*: dai tre livelli iniziali, viene staccata la presentazione che viene portata sul client, mentre la parte logica e quella dati, continuano a rimanere sul server;

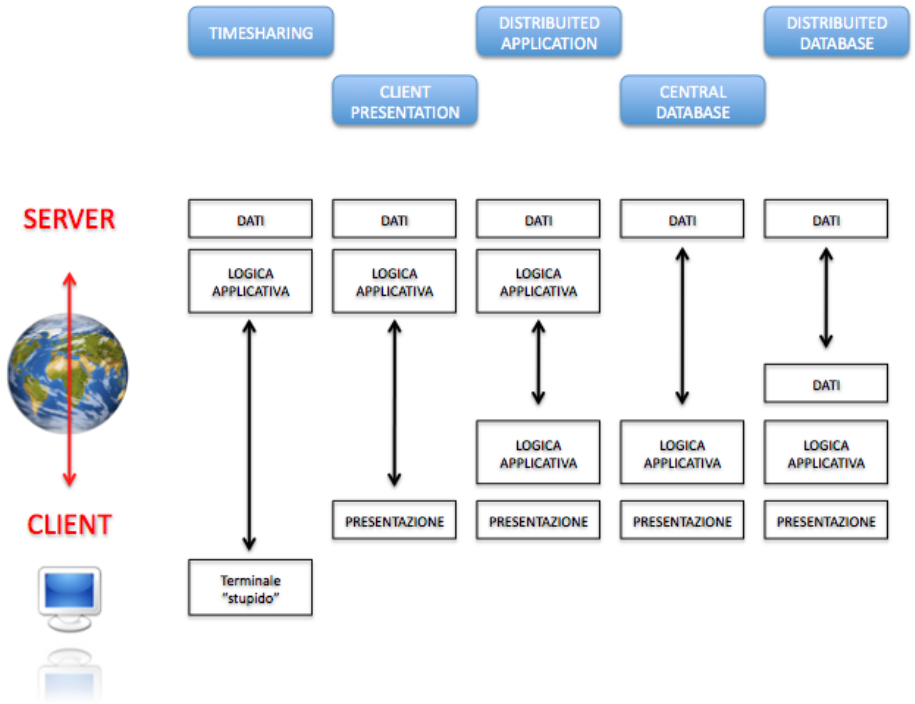

Figura 2.1.2: Organizzazioni Client/Server.

- *Application distribution*: aumentando la potenza elaborativa dei client, si è pensato di spostare oltre che alla presentazione anche una parte della logica su quest'ultimo; questo tipo di operazione ha, da un lato, reso meno gravose le elaborazioni sui server, ma dall'altro ha aumentato i costi a causa dell'introduzione di nuovo software, in quanto necessita l'aggiornamento su ogni client su cui è installato. La caratteristica principale di questo stadio è che la parte di logica è distribuita in parte sul client e in parte sul server;

- *Central database*: il passo successivo è stato quello di spostare tutta la logica sul client insieme alla presentazione, lasciando sul server solo la parte dati. Questo tipo di soluzione ha il vantaggio di avere una banca dati centralizzata, ma lo svantaggio, per aziende molto grandi, è quello di avere un grande numero di accessi localizzati in un singolo punto;

- *Distributed database*: per ovviare agli svantaggi della soluzione precedente si è

pensato di portare sui client anche la parte dati. In questo modo si pone una soluzione al sovraccarico generato dal gran numero di utenti che si connettono ai dati, ma si crea un altro problema legato alla sincronizzazione delle banche dati.

L'architettura del progetto sviluppato in questa tesi è quindi di tipo Client/Server, poichè ci sono vantaggi realizzativi sia in termini economici che in termini di estendibilità futura dei servizi offerti.

2.1.1 Piattaforma Client

I sistemi operativi per dispositivi mobili sono componenti software che garantiscono il funzionamento di dispositivi quali telefoni cellulari, smartphone, tablet, palmari e lettori MP3, coordinando e gestendo le risorse (hardware e software), e creando un'interfaccia con l'utente. Diversamente dai sistemi operativi per desktop e laptop, essi devono affrontare problematiche critiche tra cui: limitatezza delle risorse (sia in termini di CPU che di memoria RAM), dimensioni ridotte dello schermo, sistemi touch-screen più o meno avanzati, tecnologie differenti per l'accesso ad Internet, consumi energetici. Nell'accezione moderna, il sistema operativo mobile non è solo un prodotto software, ma una vera e propria piattaforma, che mette a disposizione degli sviluppatori delle API su cui sviluppare applicazioni.

Un *ecosistema mobile* è costituito dal sistema operativo, inteso come piattaforma, dagli sviluppatori, che incrementano il numero e migliorano la qualità delle applicazioni disponibili, e dagli utilizzatori, che acquistano sia la piattaforma che le applicazioni nello store. In questo sezione verranno trattati i principali sistemi operativi mobile e i relativi framework per lo sviluppo delle applicazioni.

Android

Android ((1)), nato nel 2003, è il più diffuso sistema operativo per dispositivi mobili; open source, è distribuito sotto licenza Apache, ovvero vi è la possibilità di modificare e

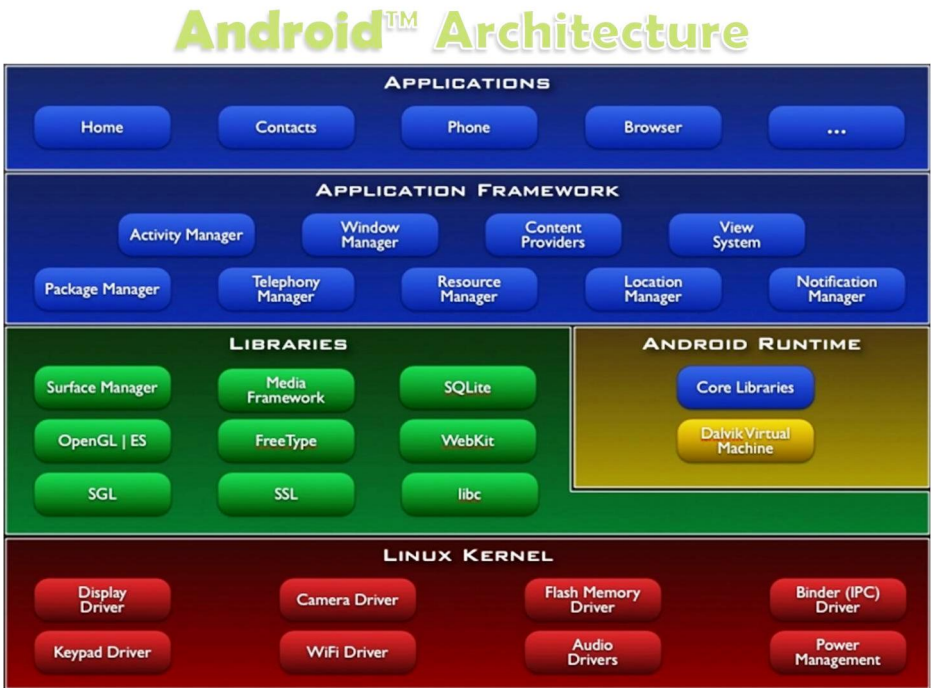

Figura 2.1.3: Architettura del sistema operativo Android.

distribuire liberamente il codice sorgente.

L'architettura Android, riportata in figura 2.1.3, è così definita:

- *kernel*, basato sul kernel Linux(versioni 2.6 e 3.x), contiene i driver per il funzionamento del dispositivo;
- *strato middleware contenente Librerie ed API*: scritte in C o C++, sono le librerie che forniscono le funzionalità standard al dispositivo, ovvero gestione delle funzioni del display (Surface Manager), gestione dei media (Media Framework), fonts di sistema, DBMS (SQLite), web browser engine (WebKit), e numerose altre;
- *Android Runtime*: Contiene la Dalvik virtual machine, una macchina virtuale ottimizzata per sfruttare la poca memoria presente nei dispositivi mobili, e che consente di far girare diverse istanze della macchina virtuale contemporaneamente, nascondendo al sistema operativo sottostante la gestione della memoria e dei thread. E'

presente, inoltre, un compilatore just-in-time che esegue il Dalvik dex-code, simile al bytecode Java;

- *Framework di applicazioni che include librerie java basate su Apache Harmony ((3))*, costituito da una serie di componenti e API che eseguono numerosi compiti, tra i quali la gestione dell'interazione con l'utente (tramite l'Activity Manager), la condivisione delle informazioni tra i vari processi (Content Provider), gestione delle finestre (Window Manager), gestione delle funzionalità telefoniche (Telephony Manager), ottimizzazione delle risorse (Resource Manager), gestione del ciclo di vita delle applicazioni (Package Manager), gestione della localizzazione (Location Manager), gestione delle notifiche con l'utente (Notification Manager).

Per quanto riguarda gli aggiornamenti, Android ha un rapido ciclo di rilascio (nuove versioni ogni sei-nove mesi). Gli aggiornamenti sono in genere di natura incrementale e apportano miglioramenti del software a intervalli regolari. Tra una major release e l'altra vengono messi a disposizione rilasci intermedi per risolvere problemi di sicurezza e altri bug del software.

Sviluppo Applicazioni Tutte le applicazioni Android, dovendo basarsi sul framework di librerie Java, sono scritte in Java.

L'Android software development kit SDK ((2)) include un insieme di tool di sviluppo, quali un debugger, librerie, un emulatore basato su QEMU ((4)), codici di esempio e tutorials. Le piattaforme di sviluppo supportate includono computer con sistema operativo Linux, MAC OS X (dalla 10.5.8) e Windows (da XP). L'IDE (Integrated Development Enviroment) ufficialmente supportato è ADT (Android Development Tools), basato su Eclipse ((5)) ma, in ogni caso, è possibile sviluppare applicazioni Android anche con altri IDE.

iOS

iOS ((6)) è un sistema operativo di Apple, derivato da Unix BSD ed ha un microkernel XNU Mach basato su Darwin OS. Nato nel 2007, è rilasciato sotto licenza APSL (Apple Public Source License), che consente agli utenti di migliorare il software ma non di rilasciarlo a terzi.

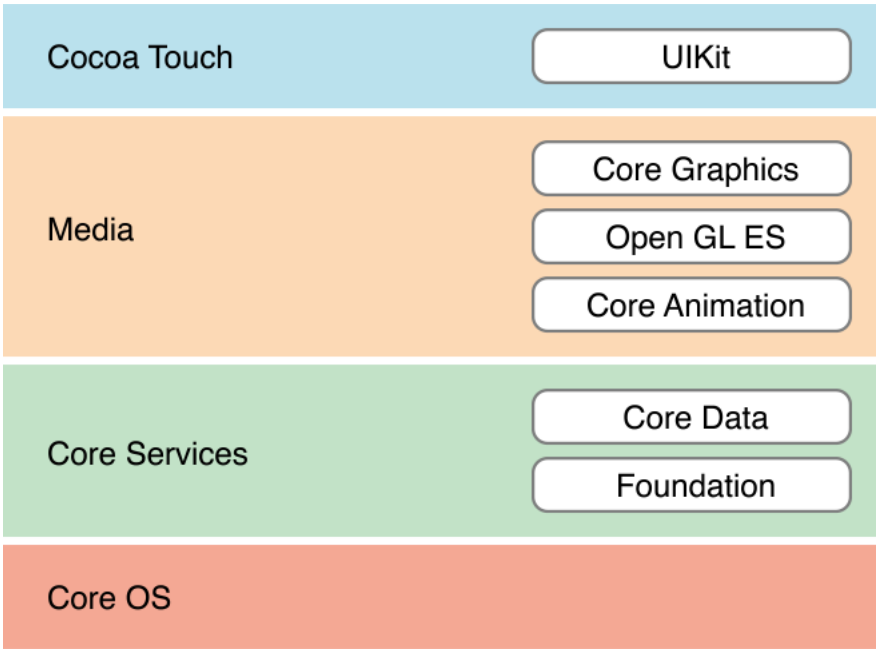

Figura 2.1.4: Architettura del sistema operativo iOS.

iOS ha quattro livelli di astrazione, riportati in figura 2.1.4:

- *Core OS Layer*: contiene le caratteristiche di basso livello su cui sono basate la maggior parte delle tecnologie. Contiene il kernel ed interfacce di basso livello UNIX. Gestisce la memoria virtuale, i threads, il filesystem e i driver delle periferiche. Sono presenti, inoltre, framework per l'esecuzione di DSP, calcoli di algebra lineare ed elaborazione delle immagini, per la gestione del Bluetooth, per la gestione degli accessori hardware eventualmente connessi al dispositivo e per la sicurezza;
- *Core Services Layer*: contiene i servizi di sistema fondamentali che tutte le applicazioni usano. Offre quindi servizi di alto livello basati sui Core Service Frameworks,

ad esempio lo storage di dati in remoto (iCloud); la protezione dei dati sensibili, per evitare che applicazioni malevole facciano uso di essi; la gestione dei documenti XML; la possibilità di condividere dati tra le varie applicazioni;

- *Media Layer*: contiene le tecnologie grafiche, audio e video;
- *Cocoa Touch Layer*: contiene i framework indispensabili per costruire applicazioni iOS (Cocoa Service Framework) e definisce l'infrastruttura di base ed il supporto per tecnologie fondamentali come il multitasking, l'input basato sul touch, le notifiche in push e locali e molti altri servizi di alto livello.

Apple rilascia aggiornamenti di iOS circa una volta l'anno. Tali aggiornamenti costituiscono revisioni complete del sistema operativo.

Sviluppo Applicazioni Le applicazioni per iOS possono essere sviluppate tramite l'iOS SDK ((7)), disponibile solo su sistema operativo MAC OS X. Esse sono scritte in Objective-C, anche se esiste il supporto per C e C++.

L'IDE di riferimento per lo sviluppo di applicazioni per iOS è XCode, che include un editor di sorgenti, uno strumento per disegnare le interfacce utente, il compilatore LLVM e un emulatore, oltre ad altri numerosi tool per il debug, il testing e la gestione degli errori.

Windows Phone OS

Windows Phone OS ((8)) è un sistema operativo Microsoft, nato nel 2010, orientato ai dispositivi mobile. Nonostante sia il successore di Windows Mobile, sistema operativo nato nel 1996 per i palmari, è completamente differente da quest'ultimo.

E' distribuito con licenza Microsoft EULA, per cui è soggetto a limitazioni d'uso, di garanzia e di responsabilità.

Windows Phone OS ha un'architettura a quattro livelli, come rappresentato in figura 2.1.5:

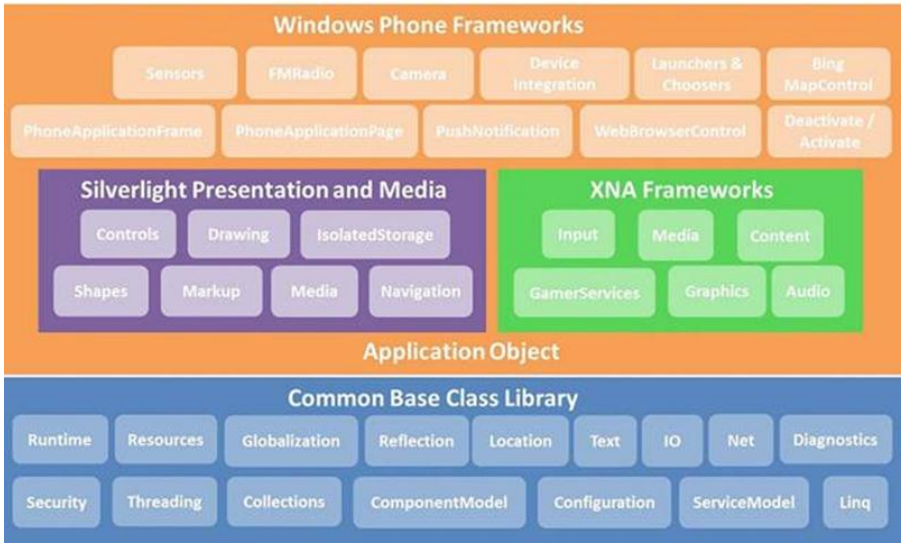

Figura 2.1.5: Architettura del sistema operativo iOS.

- *Common Base Class Library*: è una libreria standard che include un vasto numero di funzionalità comuni, come ad esempio la lettura e scrittura su un file, il rendering grafico, l'interazione con i database, la manipolazione di documenti XML e il supporto per il multithreading;

- *Silverlight Presentation and Media*: fornisce librerie per la navigazione, la grafica, la gestione dei media;

- *XNA Frameworks*: è un insieme di librerie, servizi e risorse per la gestione dell'accelerazione grafica e del rendering, dell'audio e dei servizi di autenticazione e connettività;

- *Windows Phone Frameworks*: definisce dei blocchi comuni per tutte le applicazioni Windows Phone. Esso fornisce l'interfaccia al sistema ed alle risorse hardware, ovvero i sensori, l'accelerometro, il compasso, il giroscopio e la fotocamera. In Windows Phone Framework è presente anche un sistema di notifiche.

Sviluppo Applicazioni Le applicazioni per Windows Phone possono essere scritte in C++, C#, C, Visual Basic e HTML5, grazie al vasto supporto del framework .NET. La più

recente versione della Windows Phone SDK ((9)) è disponibile solo per sistemi operativi Windows 8 a 64bit.

L'IDE di riferimento per lo sviluppo di applicazioni Windows Phone è Visual Studio per Windows Phone, che include un editor di sorgenti, di testing, di debug, di gestione delle risorse e del ciclo di vita dell'applicazione. Microsoft mette inoltre a disposizione altri significativi tools come Express Blend, che permette di creare interfacce utente; Silverlight ((10)) e XNA per la gestione della grafica 2D e 3D; Windows Phone Emulator, l'emulatore dei dispositivi Windows Phone.

Scelta adottata

Nella scelta dell'ecosistema su cui sviluppare e presentare un'applicazione per la visita dei siti culturali, **Windows Phone OS** è stato preferito agli altri, poiché si è tenuto conto di vari fattori:

- **originalità**: nello store di Windows Phone, non si trovano applicazioni simili, per cui essa costituirebbe il primo tentativo di visita assistita dei siti culturali;
- **possibilità di utilizzare un ambiente integrato per lo sviluppo, con ampio supporto alla metodologia agile, avendo a disposizione un'ampia expertise aziendale**.

2.1.2 Codifica e serializzazione dei dati

La serializzazione è un processo per salvare un oggetto in un supporto di memorizzazione lineare o per trasmetterlo su una connessione di rete. Essa può essere realizzata in modalità binaria o testuale. La modalità binaria, seppur molto efficiente, presenta scarsa interoperabilità e flessibilità. La serializzazione testuale, invece, essendo basata sul paradigma REST ((11)), determina un'ottimizzazione nello scambio di informazioni tra ambienti eterogenei, rendendo le strutture tradotte indipendenti dall'architettura e dalle differenti rappresentazioni dei dati. I formati di serializzazione testuale più noti sono XML, JSON e YAML, che sono trattati di seguito.

XML

La storia di XML (eXtra Markup Language, (12)) è strettamente legata a quella di SGML (Standard Generalized Markup Language), progetto ideato da IBM per migliorare l'interoperabilità aziendale, dal momento che la comunicazione tra computer era ostacolata da una ricca gamma di formati di file.

XML, infatti, è nato grazie ad una semplificazione di SGML per consentire di definire in maniera semplice nuovi linguaggi di markup da utilizzare in ambito web. In definitiva, XML è un linguaggio marcatore basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.

Sintassi XML è una tecnologia adoperata per creare linguaggi di markup che descrivono dati di qualsiasi tipo, quindi consente di descrivere dati in modo accurato creando nuovi tag. Un esempio di sintassi XML è l'esempio **Templi.xml** della figura 2.1.6.

Un documento XML si può quindi dividere in due sezioni: il *prologo* e l'*istanza*. Il prologo contiene informazioni di carattere generale sul documento, mentre l'istanza contiene i dati. Il documento XML è costituito da un insieme di dati e di marcatori che ne specificano la struttura, come:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE Rubrica SYSTEM Templi.dtd>
<templi>
        <tempio>
                <nome>Tempio di Atena</nome>
                <annodicostruzione>500 AC</annodicostruzione>
                <indirizzo>
                        <citta>Paestum</citta>
                        <provincia>Salerno</provincia>
                <indirizzo>
        </tempio>
</templi>
```

Figura 2.1.6: Esempio di documento XML Templi.xml.

- Tag,
- Processing instructions,
- Entità,
- Commenti,
- Sezione CDATA.

Dichiarazione Le prime due righe contenute nell'esempio della figura 2.1.6 costituiscono la dichiarazione del documento XML. La prima riga contiene la specifica della versione di XML utilizzata e del tipo di codifica del documento. Nella seconda riga è stabilita la tipologia di documento alla quale appartiene il file, specificando il particolare schema DTD scelto. La dichiarazione della tipologia di documento può essere di tre tipi, come riportato nella figura 2.1.7.

Tag L'elemento radice della figura 2.1.6 è il tag **<templi>**, che contiene all'interno tutti gli altri elementi del documento. Inserire più di un elemento radice è considerato errato. I tag interni sono definiti elementi *child*, e sono parte dell'organizzazione gerarchica di XML. I tag di apertura e di chiusura vanno sempre specificati, tranne nel caso non vi siano informazioni in essi. Per gli elementi vuoti XML prevede una sintassi abbreviata **<tag/>**.

- **Esterna**:

```
<!DOCTYPE ELEMENTO_ROOT SYSTEM "nomefile.dtd>
```

- **Interna**:

```
<!DOCTYPE ELEMENTO_ROOT [CONTENUTO DTD SCHEMA]>
```

- **Mista**:

```
<!DOCTYPE ELEMENTO_ROOT SYSTEM "nomefile.dtd
[CONTENUTO DTD SCHEMA]>
```

Figura 2.1.7: Esempio di dichiarazioni di schema DTD.

Attributi I tag XML possono contenere informazioni interne che vengono definite *attributi* del tag. Essi specificano proprietà intrinseche al tag e vanno indicati attraverso l'accoppiata nome-valore, come nell'esempio in figura 2.1.8.

```
<tempio stile="ionico">Tempio di Atena</tempio>
```

.

Figura 2.1.8: Esempio di attributo XML.

Commenti Oltre alle direttive di elaborazione, in un file XML è possibile individuare i commenti, racchiusi tra le sequenze di caratteri **<!–** e **–>**. Questi possono trovarsi in qualsiasi punto del documento, ed il testo contenuto non viene elaborato dal parser.

Correttezza sintattica e validità Un documento XML è valido se è conforme al DTD associato. Un DTD (Document Type Definition) è uno strumento che definisce le componenti ammesse nella costruzione di un XML. Un esempio di DTD è riportato in figura 2.1.9.

```
<! ELEMENT tempio (nome, annodicostruzione)>
<! ELEMENT nome (#PCDATA) >
<! ELEMENT annodicostruzione (#PCDATA) >
```

Figura 2.1.9: Esempio di file DTD.

Un documento XML, in ogni caso, non deve essere necessariamente valido, ma il requisito essenziale è la correttezza nella sintassi. Tali regole sintattiche possono essere riassunte in:

- Deve essere presente un solo elemento radice;
- I tag non possono iniziare con numeri o caratteri speciali e contenere spazi;
- I tag devono essere innestati correttamente;
- Tutti i tag e gli attributi sono espressi in minuscolo;
- E' obbligatorio inserire i tag di chiusura, sia in forma estesa che in forma abbreviata, ove consentito.

Se un documento è sintatticamente valido, può essere analizzato da un *parser*, un programma che, analizzando la struttura grammaticale del documento XML, ne ricostruisce l'albero a partire dall'elemento radice e proseguendo con gli elementi child.

JSON

JSON (JavaScript Object Notation, (**?**)) è un formato di serializzazione nato appositamente per lo scambio dei dati. Un documento JSON è di facile interpretazione anche senza il processo di parsing, data la leggibilità della sua struttura.

JSON è completamente indipendente dal linguaggio di programmazione utilizzato, in quanto utilizza convenzioni conosciute dai programmatori di linguaggi della famiglia del C. Grazie a queste caratteristiche, JSON è un linguaggio ideale per lo scambio dei dati e per l'interoperabilità. Con JSON è possibile rappresentare quattro tipi primitivi:

- numeri;
- stringhe;
- variabili booleane;

- NULL.

e due tipi strutturati:

- array;
- oggetti.

Il formato JSON è ampiamente diffuso, soprattutto tra gli sviluppatori web, per le basse dimensioni dello stream dati, dovute soprattutto ad un contenuto ridotto di tag.

Sintassi La sintassi JSON è molto semplice e si basa su due strutture:

- **coppia nome/valore**, realizzabile in diversi linguaggi di programmazione come un oggetto, un record, uno struct, una tabella hash, un elenco di chiavi o un array associativo;
- **elenco ordinato di valori**, realizzabile nella maggior parte dei linguaggi di programmazione con un array, un vettore, un elenco o una sequenza.

Queste due strutture sono utilizzabili in tutti i linguaggi di programmazione, e ciò rende ancora più evidente la natura di JSON, orientata all'interoperabilità.

Oggetto Un oggetto in JSON è contenuto tra due parentesi graffe. Tra il nome e il valore sono presenti due punti ed il nome dell'oggetto è scritto tra due doppi apici, mentre il valore è scritto tra due doppi apici nel caso non sia numerico. Un esempio è in figura2.1.10.

```
{"nome": "Tempio di Atena"}
```

Figura 2.1.10: Esempio di oggetto JSON.

```
{"templi": ["Tempio di Atena", "Tempio di Nettuno"]}
```

Figura 2.1.11: Esempio di oggetto JSON.

```
{
        "templi": {
                "tempio": {
                        "nome": "Tempio di Atena".
                        "annodicostruzione": "500 AC",
                        "indirizzo": {
                                "citta": "Paestum",
                                "provincia": "Salerno"
                        }
                }
        }
}
```

Figura 2.1.12: Esempio di oggetto JSON.

Array Un array è una raccolta ordinata di valori. In JSON un array è contenuto tra due parentesi quadre e i valori sono separati da virgola, come è evidente dalla figura 2.1.11.

Le stringhe sono rappresentate come nella maggior parte dei linguaggi di programmazione e vanno sempre messe tra doppi apici, mentre i numeri possono iniziare con identificatori di segno ed essere seguiti da una parte esponenziale, così come in molti linguaggi. Solo le rappresentazioni esadecimali o ottali non sono permesse.

Nella figura 2.1.12 è possibile visualizzare un esempio completo di file JSON, analogo a quello precedente di file XML in figura 2.1.6. L'indentazione è utilizzata solo a scopo di chiarezza sintattica, ma il testo si estende su di una sola riga, senza caratteri di ritorno a capo. Naturalmente anche un documento JSON può essere elaborato da un parser e ricostruito.

YAML

YAML (YAML Ain't Markup Language, (14)), è un linguaggio nato nel 2001 con l'obiettivo di essere *human-friendly*. E' progettato per persone che lavorano con dati, grazie alla sua comprensibilità ed interoperabilità. YAML è peraltro un linguaggio molto pulito, in quanto riduce al minimo la quantità di caratteri strutturali e permette di mostrare i dati in modo naturale e significativo. Gli obiettivi di YAML sono:

- semplicità di lettura;
- corrispondenza con i tipi per la maggioranza dei linguaggi di programmazione;
- esportabilità dei dati;
- estendibilità;
- facilità di implementazione e usabilità.

Sintassi YAML mette a disposizione due tipi di sintassi, una indentata e una in linea, simile al formato JSON. Inoltre in YAML ci sono degli indicatori che servono a descrivere la struttura ed il contenuto di un documento YAML.

Liste Le liste sono collezioni di elementi. La loro rappresentazione in YAML può essere indentata (fig. 2.1.13) o allineata (fig. 2.1.14).

```
--- # Templi di Paestum
 - Tempio di Nettuno
 - Tempio di Atena
```

Figura 2.1.13: Lista nel formato indentato YAML.

```
--- # Templi di Paestum
[Tempio di Nettuno, Tempio di Atena]
```

Figura 2.1.14: Lista nel formato convenzionale YAML.

Array associativi Gli array sono associazioni nome/valore, e possono essere rappresentati come in figura 2.1.15.

Stringhe Le stringhe non richiedono gli apici o i doppi apici per essere rappresentate. La rappresentazione delle stringhe può essere indentata, come nella figura 2.1.16, oppure allineata, come nella figura 2.1.17, dove il testo viene allineato in un singolo paragrafo.

```
--- # Blocco indentato
   nome: Tempio di Atena
   annodicostruzione: 500 AC
 --- # Blocco allineato
 {nome: Tempio di Atena, annodicostruzione: 500 AC}
```

Figura 2.1.15: Array in YAML.

```
--- |
       Questo è il
              Tempio di Atena,
              costruito nel 500 AC
       a Paestum.
```

Figura 2.1.16: Stringa nel formato indentato YAML.

```
--- >
Questo è il
Tempio di Atena,
costruito nel 500 AC
a Paestum.
```

Figura 2.1.17: Stringa nel formato convenzionale YAML.

Scelta

Nell'ambito del progetto di tesi, nonostante i formati JSON e YAML presentino ridondanze minori rispetto a XML, come già specificato, quest'ultimo risulta comunque il più adatto alla descrizione di strutture dati e presenta uno spettro di utilizzo più vasto. Inoltre, come si vedrà in seguito, l'utilizzo del formato GeoRSS, derivato da XML, per la descrizione delle caratteristiche georeferenziate dei punti di interesse, inducono comunque alla scelta del formato XML.

2.1.3 Tecniche di geolocalizzazione

La geolocalizzazione è l'identificazione della posizione geografica di un dato oggetto nel mondo reale. Gli oggetti possono essere telefoni cellulari, pc, tablet, palmari e ci sono diverse tecniche, indoor ed outdoor:

- **outdoor**: localizzazione satellitare;
- **indoor**: infrastrutture ad-hoc e WiFI;
- **indoor ed outdoor**: infrastruttura cellulare.

Vi sono inolte diversi metodi per valutare la qualità di un sistema di localizzazione:

- **Errore medio**: corrispondenza tra la posizione stimata e quella reale;
- **Probabilità di corretta locazione**: probabilità di localizzare l'obiettivo entro una determinata soglia;
- **Rendimento**: capacità del metodo a stimare la posizione in tutti i tipi di ambienti;
- **Consistenza**: misura della stabilità dell'errore medio in ambienti differenti;
- **Overhead**: quantità di informazione scambiata tra il terminale ed il sistema;
- **Consumo di potenza**: quantita delle risorse energetiche impiegate;
- **Latenza**: intervallo di tempo tra la richiesta di posizionamento e la risposta del sistema;
- **Costi roll-out**: costi necessari all'installazione dell'infrastruttura;
- **Costi operativi**: costi legati al mantenimento dell'infrastruttura.

Figura 2.1.18: Rappresentazione dei satelliti GPS in orbita intorno alla terra.

Localizzazione satellitare

La localizzazione satellitare si basa su infrastrutture indipendenti dedicate, ovvero un certo numero di satelliti utilizzati solo a questo scopo e di tipo terminal-based. I vantaggi del posizionamento satellitare sono la copertura globale e l'alto livello di accuratezza, ma ci sono anche importanti inconvenienti, come l'alto consumo dei dispositivi che utilizzano il segnale GPS e i disturbi naturali che possono ostacolarne la ricezione.

GPS Il sistema GPS (Global Positioning System, (15)), avviato dagli USA negli anni '70 e completato nel 1993, è stato realizzato per motivi essenzialmente militari, al fine di consentire il percorso dei mezzi militari, sia sulla terraferma che in mare in modo da localizzarne la posizione in ogni momento e consentire eventuali operazioni di supporto e salvataggio. Il GPS utilizza dai 24 ai 32 satelliti artificiali, divisi in gruppi da 4, che ruotano attorno alla terra a circa 20.200 km di altezza in orbite che formano tra loro angoli di 60 gradi.

Principi di funzionamento Il sistema GPS è nato come versione satellitare e perfezionamento del sistema LORAN, nato negli USA attorno al 1940, che consentiva la determi-

nazione della posizione lungo le rotte di grande traffico navale ed aereo, utilizzando un grande numero di stazioni terrestri.

Il sistema GPS, così come il LORAN, consente di determinare la propria posizione sulla superficie terrestre e la propria altitudine in qualunque punto ci si trovi tramite un ricevitore GPS, il quale intercetta a terra il segnale generato dai satelliti in orbita che passano sopra di noi, così come rappresentato in figura 2.1.18. Infatti, visto il numero, l'orbita ed il periodo di rotazione, in ogni istante e in ogni punto terreste è possibile intercettare il segnale generato da 6 a 12 satelliti.

Satelliti Le funzioni dei satelliti possono essere così sintetizzate:

- trasmettere informazioni agli utilizzatori mediante un segnale radio;
- mantenere un riferimento di tempo accurato, grazie agli orologi di bordo;
- ricevere e memorizzare informazioni dal segmento di controllo;
- eseguire manovre e correzioni di orbita.

Ricevitore I ricevitori GPS commerciali, dal costo molto contenuto, consentono di sintonizzarsi automaticamente sulle frequenze dei satelliti GPS e, dopo un tempo di ricerca relativamente breve, di determinare la propria posizione ed, eventualmente, la propria quota, elaborando le distanze di almeno quattro satelliti. Nei navigatori GPS per auto e negli smartphone, il risultato dell'elaborazione viene mostrato come punto all'interno di una cartina geografica completa, che può essere ingrandita fino a diventare una vera e propria cartina topografica.

Precisione La precisione della ricezione GPS è influenzata da una serie di fattori, tra cui la posizione dei satelliti, l'eventuale presenza di rumore del segnale radio, le condizioni atmosferiche e le barriere naturali. In generale, tali disturbi possono introdurre errori di precisione tra 1 e 10 metri. La determinazione più accurata della posizione si verifica

quando il satellite e il ricevitore GPS sono in vista tra loro senza nessun altro oggetto schermante. Sotto questa ipotesi, il margine di errore del posizionamento è di circa 90 cm.

Localizzazione cellulare

La localizzazione cellulare è utilizzata nelle reti cellulari, come GSM o UMTS, per ricavare la posizione di un utente. Questo sistema di localizzazione è utilizzato per ricavare la presenza di un utente all'interno di una cella, per cui è soggetto a margini di errore abbastanza ampi. Per questo motivo, i gestori hanno predisposto nuovi strumenti per equipaggiare le proprie reti con dispositivi e protocolli finalizzati alla realizzazione del posizionamento in modo più accurato ed efficiente. A tal proposito, diversi metodi sono stati specificati dai gruppi di standardizzazione, come 3GPP ((16)). La maggior parte di questi metodi è network-based, per cui anche i dispositivi più datati possono usufruirne. Il vantaggio di questi sistemi di localizzazione cellulare è che possono essere utilizzati anche in ambienti chiusi, diversamente dai sistemi satellitari. Il posizionamento cellulare può essere molto costoso per quanto riguarda l'overhead dovuto alla segnalazione, soprattutto se è richiesto un alto grado di accuratezza e, inoltre, la capacità impiegata per la localizzazione è indisponibile per i servizi voce e dati.

Localizzazione indoor

La localizzazione indoor è nata per essere utilizzata all'interno di grandi edifici, campus universitari, strutture museali, strutture ospedaliere e uffici di vario genere. E' basata su tecnologie radio, infrarossi o ad ultrasuoni, con un raggio di comunicazione evidentemente limitato. La realizzazione di un sistema di localizzazione indoor può avvenire mediante la realizzazione di infrastrutture dedicate o l'utilizzo di infrastrutture preesistenti, come le reti WLAN. La localizzazione indoor ha il vantaggio di presentare un basso consumo di potenza dei dispositivi utilizzati per la ricezione e l'elevata accuratezza dovuta al corto raggio delle tecnologie usate.

Esistono diverse soluzioni commerciali per la localizzazione indoor, come **RADAR** ((17)), sviluppata da Microsoft, che è un sistema a radio frequenza; **Real Time Location System - RTLS** ((18)), di Ekahau, che si basa sull'infrastruttura WLAN IEEE 802.11; **Visibility System** ((19)), di Aeroscout, basato sempre sull'infrastruttura WLAN IEEE 802.11; **Wireless Location Appliance** ((20)), di Cisco, basato su WLAN.

Scelta adottata

Nell'ambito di studio, la scelta operata è sulla tecnologia **GPS**, per i seguenti motivi:

- **disponibilità dispositivi GPS**: tutti gli smartphone con sistema operativo Windows Phone, anche quelli appartenenti a fasce di prezzo più basse, presentano un dispositivo di ricezione del segnale GPS;
- **impossibilità di utilizzo localizzazione indoor**: date le dimensioni considerevoli del sito archeologico, si è ritenuto improbabile l'installazione di una rete WiFi che coprisse tutta l'area, al fine di localizzare tutti i dispositivi presenti;
- **inutilità localizzazione cellulare**: la localizzazione cellulare è soggetta a margini di errore ampi per cui, nell'ambito del caso in questione, in cui vi è la necessità di individuare con margini d'errore ridotti la posizione corretta, risulta inutile;
- **elevata precisione**: i margini di errore del GPS sono piuttosto bassi, e quindi compatibili con la localizzazione e discriminazione della posizione dei monumenti.

2.1.4 Codifica e serializzazione dei dati georeferenziati

GeoRSS

GeoRSS è uno standard emergente per la codifica delle informazioni come parte di un feed Web, infatti il nome *GeoRSS* deriva proprio da *RSS*, il più conosciuto feed Web. Il GeoRSS, il contenuto riguardante la posizione consiste di punti geografici, linee e poligoni e delle relative descrizioni, come mostrato in figura 2.1.19. Tali feed sono progettati per essere utilizzati da software geografici come i generatori di mappe.

```
<?xml version="1.0"?>
<?xml-stylesheet href="/eqcenter/catalogs/rssxsl.php?feed=eqs7day-M5.xml" type="text/
    xsl"
                media="screen"?>
<rss version="2.0"
     xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#"
     xmlns:dc="http://purl.org/dc/elements/1.1/">
       <templi>
               <title>Elenco dei templi</title>
       <description>Questo è un elenco dei templi presenti negli scavi di Paestum</
           description>
               <tempio>
                       <nome>Tempio di Atena</nome>
                       <annodicostruzione>500 AC</annodicostruzione>
                       <indirizzo>
                               <citta>Paestum</citta>
                               <provincia>Salerno</provincia>
                       <indirizzo>
                       <geo:lat>5.5319</geo:lat>
                       <geo:long>95.8972</geo:long>
               </tempio>
       </templi>
</rss>
```

Figura 2.1.19: Esempio di documento GeoRSS.

Le due codifiche principali di GeoRSS sono GeoRSS-Simple e GeoRSS Geography Markup Language. Il primo è un formato leggero che supporta geometrie semplici (punti, linee, poligoni) e può essere utilizzato per gli scopi tipici di geolocalizzazione. GeoRSS GML è un formato dell'Open Geospatial Consortium (OGC) GML Application Profile, e supporta un vastissimo range di caratteristiche rispetto a GeoRSS Simple. E' presente anche un formato di serializzazione GeoRSS definito dalla W3C, ma è obsoleto e parzialmente deprecato, pur essendo ancora il più utilizzato. Per il lavoro oggetto di questa tesi,

il formato GeoRSS risulta particolarmente adatto per lo scambio di dati georeferenziati attraverso un Web Service.

2.1.5 Rappresentazione dei dati georeferenziati

Il Web Mapping è un processo di progettazione, implementazione, generazione e produzione di mappe sul web. Un caso particolare di mappe web sono le mappe mobili, visualizzate su periferiche mobili, come telefonini, smartphone, tablet e GPS.

I vantaggi derivanti dall'utilizzo di mappe web sono:

- facilità di generazione di mappe aggiornate;
- economicità dell'infrastruttura software ed hardware;
- facilità di distribuzione degli aggiornamenti;
- supporto e compatibilità con la maggior parte dei browser e dei sistemi operativi;
- facilità di personalizzazione;
- supporto per l'hyperlinking di punti di interesse;
- facilità di integrazione con altri oggetti multimediali.

I servizi di web mapping più noti sono **Google Maps** e **Bing Maps**.

Google Maps

Google Maps ((21)) è un servizio di web mapping nato nel 2005 e fornito da Google, che include numerosi altri servizi basati sulle mappe (mappe stradali, pianificazione di percorsi e localizzazione di imprese). Le mappe non sono aggiornate in tempo reale, ma spesso dopo mesi o anni.

Google Maps API Google ha messo a disposizione le proprie API per consentire agli sviluppatori di integrare Google Maps all'interno dei loro siti web. Il servizio offerto è gratuito, anche se Google si riserva, nei termini di licenza, di introdurre future forme di pagamento.

Bing Maps

Bing Maps ((22)) è un servizio di web mapping nato nel 2009 e distribuito da Microsoft nella *Bing suite*. Fornisce numerose tipologie di mappe: stradali, satellitari, aeree, tridimensionali, oltre che, naturalmente, servizi di traffico e di ricerca delle imprese.

Bing Maps API Microsoft, oltre a mettere a disposizione le Bing API agli sviluppatori, fornisce anche una SDK disponibile su più piattaforme, per realizzare applicazioni integrabili con la maggior parte dei prodotti commerciali Microsoft.

Scelte

La scelta di Windows Phone OS come piattaforma ha reso quasi obbligatoria l'adozione delle mappe di Bing Maps, che sono supportate nativamente dalla Windows Phone SDK.

2.1.6 Web Service e protocolli di scambio delle informazioni

Un *Web Service* è, secondo la definizione data dal consorzio *W3C* ((23)), un sistema software progettato per supportare l'interoperabilità tra diversi elaboratori su di una medesima rete ovvero in un contesto distribuito. L'interoperabilità è ottenuta associando all'applicazione un'interfaccia software (descritta in formato automaticamente elaborabile, ad esempio il Web Services Description Language), che espone all'esterno i servizi associati e utilizzando la quale altri sistemi possono interagire con l'applicazione stessa, attivando le operazioni descritte nell'interfaccia tramite appositi messaggi di richiesta. Tali messaggi sono formattati secondo gli standard XML ed incapsulati e trasportati tramite i protocolli web (di solito HTTP), da cui il nome web service. Un esempio di trasporto è in figura 2.1.20.

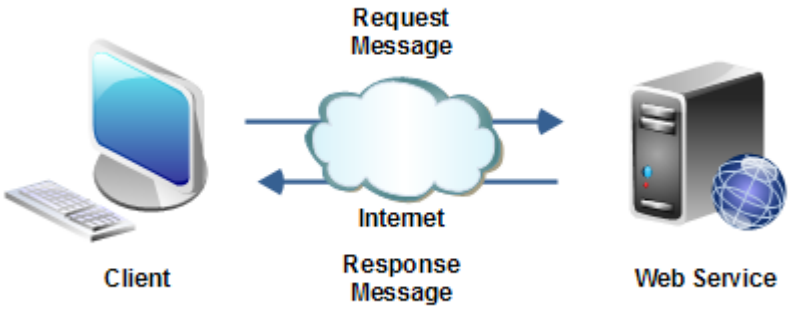

Figura 2.1.20: Scambio messaggi in un Web Service.

I *Web Service* hanno ridefinito nel tempo il modo in cui viene utilizzato il Web, che è visto non più come strumento di pubblicazione e consultazione di pagine statiche, ma anche e soprattutto come strumento per elaborare dati, interagire, integrare le applicazioni e creare business. Questa nuova generazione di applicazioni internet consente ai privati di accedere a servizi studiati appositamente per le loro esigenze, ed alle aziende di comunicare e gestire i rapporti con clienti, partner e fornitori in modo semplice e immediato. L'accesso ai Web Service è possibile da qualunque dispositivo in grado di interagire con la rete, ovvero PC, smartphone, palmari o notebook.

Un Web Service può essere scritto in qualsiasi linguaggio di programmazione e, come tutte le applicazioni web, ha la caratteristica fondamentale di trasmettere informazioni in formato testuale attraverso protocolli di tipo request-reply (HTTP). Solitamente, un Web Service comunica con i browser tramite pagine HTML, incapsulate all'interno di una risposta HTTP; diversamente, nella comunicazione tra Web Service o tra Web Service e applicazioni le informazioni sono scambiate in maniera strutturata (cioè capaci di autodescriversi in modo da essere comprensibili sia ad un agente software che ad un umano). I formati di descrizione sono naturalmente i formati di serializzazione dei dati trattati precedentemente, ovvero XML, JSON e YAML, la cui caratteristica principale è quella di essere testi semplici da cui estrarre informazioni tramite il parsing.
I protocolli per i web service utilizzano tre architetture diverse:

- **RPC oriented**;
- **Message oriented**;
- **REST**.

RPC oriented

L'architettura RPC oriented è basata sul concetto di Remote Procedure Call dei middleware tradizionali. Quindi, il client invia al server una richiesta, contenente il nome della procedura remota da invocare e i parametri, usando l'operazione POST del protocollo HTTP; Il server invia il risultato al client come risposta HTTP.

XML-RPC Il primo protocollo specificamente sviluppato per i Web Service è stato *XML-RPC*, in cui la richiesta e la risposta del messaggio sono formulate in linguaggio XML. XML-RPC sfrutta la disponibilità di parser XML su tutte le principali piattaforme e la facilità con cui si possono generare documenti XML. Le principali carenze di XML-RPC sono: l'assenza di meccanismi per la gestione di tipi di dato non supportati; il mancato supporto nativo di funzionalità avanzate come autenticazione e gestione delle

transazioni; mancanza di definizione formale delle interfacce delle procedure remote, per cui eventuali errori o incongruenze sono scoperti solo a tempo di esecuzione. Per ovviare a tali limiti è stato introdotto SOAP.

SOAP Acronimo di *Simple Object Access Protocol*, SOAP ((24)) è basato su XML ed è accattato come standard dal consorzio W3C. SOAP è supportato dai principali ambienti di sviluppo e ha la caratteristica di separare il formato dei messaggi dal modo in cui sono veicolati attraverso un protocollo già esistente. SOAP può utilizzare protocolli diversi da HTTP, come SMTP. I vantaggi dell'utilizzo di SOAP, rispetto ad XML-RPC sono prevalentemente nella versatilità, dal momento che SOAP consente di estendere l'insieme dei tipi da usare come parametri e valori di ritorno e consente di estendere il protocollo stesso al fine di aggiungere funzionalità come la sicurezza o la gestione delle transizioni senza perdere l'interoperabilità. Per la definizione formale dell'interfaccia, SOAP ricorre a WSDL.

WSDL WSDL, ovvero *Web Service Description Language* è un linguaggio per definire le interfacce dei Web Service basati su SOAP. Anch'esso è uno standard W3C ed è naturalmente basato su XML. Un documento WSDL contiene una specifica machine-readable di:

- punti di accesso ai web service;
- protocolli utilizzati;
- operazioni disponibili, con i rispettivi parametri di ritorno;
- tipi non predefiniti usati dalle operazioni.

Anche se in teoria un documento WSDL dovrebbe essere in un formato leggibile e modificabile da un essere umano, in pratica il formato è piuttosto complesso e si rende necessaria la generazione del documento tramite appositi tool e librerie, offerti dai principali ambienti di sviluppo. Tipicamente il WSDL è disponibile online sul sito che ospita il web service

ed è utilizzato a tempo di esecuzione sia per verificare che non ci siano modifiche dell'interfaccia server, sia per creare le classi che rappresentano la comunicazione col client. La combinazione SOAP+WSDL consente di realizzare web service con uno sforzo di programmazione contenuto, sia lato client che lato server e, grazie all'uso del servizio UDDI, garantisce ed aumenta l'interoperabilità.

UDDI UDDI (*Universal Description Discovery and Integration*) è un web service basato su SOAP+WSDL che realizza un repository di descrizioni WSDL con funzioni di pubblicazione e ricerca di web service in base ad una serie di metadati. Lo standard UDDI originariamente ipotizzava lo sviluppo di un *Universal Business Registry* pubblico che consentisse una ricerca globale dei web service, ma attualmente l'idea dell'UBR è stata abbandonata ed il servizio UDDI è utilizzato soprattutto in ambito intranet come repository centralizzato di documenti WSDL.

Message oriented

In un'architettura di servizi Message oriented le diverse applicazioni si scambiano messaggi unidirezionali che hanno le seguenti caratteristiche:

- il focus è sul messaggio più che sull'operazione;
- se una richiesta genera una risposta, questa è inviata con un messaggio indipendente dal messaggio di richiesta;
- il servizio non è tenuto a elaborare i messaggi che riceve in ordine di ricezione;
- è sfumata la distinzione tra client e server: l'architettura è peer-to-peer.

I vantaggi dell'utilizzo di un'architettura Message oriented sono: il disaccoppiamento anche temporale tra client e server, infatti, se il server non è momentaneamente disponibile, il messaggio può essere mantenuto in una coda; la maggiore flessibilità nella distribuzione (smistamento intelligente dei messaggi tra più server); la flessibilità nell'ordine delle operazioni (priorità diverse per i messaggi).

REST

REST (Representational state transfer, (11)) è un tipo di architettura software per i sistemi di ipertesto distribuiti come il World Wide Web. Esso si riferisce ad un insieme di principi di architetture di rete, i quali delineano come le risorse sono definite e indirizzate. Il termine è spesso usato nel senso di descrivere ogni semplice interfaccia che trasmette dati su HTTP senza un livello opzionale come SOAP o la gestione della sessione tramite i cookie.

Le applicazioni basate sui principi REST, spesso definite RESTful, usano richieste HTTP per inviare dati (creare o aggiornare), leggere dati (eseguire query) e cancellare dati.

Nel dettaglio, lo stile architetturale di REST, rappresentato in figura 2.1.21, consiste di un lato client e un lato server: i client inviano richieste ai server; i server elaborano tali richieste e restituiscono ai client i risultati delle elaborazioni.

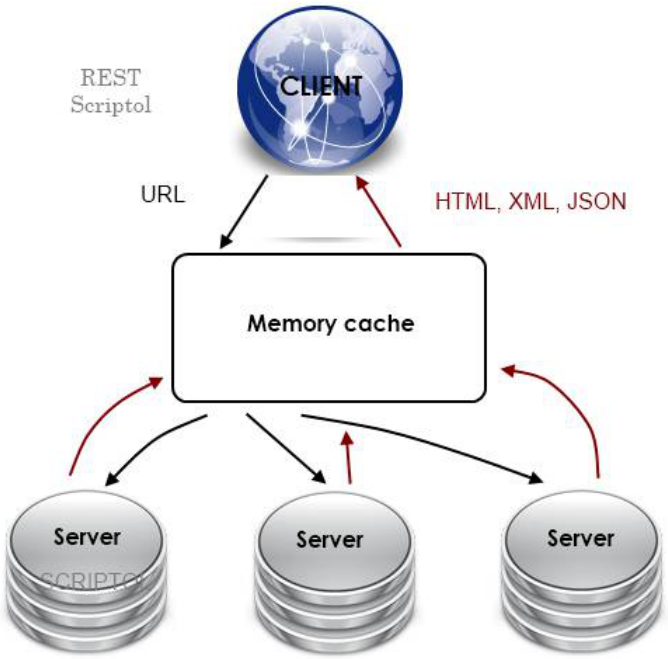

Figura 2.1.21: Modello di architettura REST.

Richieste e risposte si basano sul trasferimento di rappresentazioni di **risorse**.

L'esistenza delle risorse è un concetto importante in REST, in quanto esse sono fonti di informazioni a cui si può accedere tramite un identificatore globale (un URI). Per utilizzare le risorse, le componenti di una rete (client e server) comunicano attraverso una interfaccia standard (ad es. HTTP) e si scambiano rappresentazioni di queste risorse (il documento che trasmette le informazioni).

Un numero qualsiasi di connettori (client, server, cache, tunnel ecc.) può mediare la richiesta, ma ogni connettore interviene senza conoscere la storia passata delle altre richieste(stateless). Di conseguenza una applicazione può interagire con una risorsa conoscendo due cose: l'identificatore della risorsa e l'azione richiesta. L'applicazione deve conoscere il formato dell'informazione (rappresentazione) restituita, tipicamente un documento HTML, XML o JSON, ma potrebbe essere anche un'immagine o qualsiasi altro contenuto.

2.2 Tecnologie di Sviluppo

2.2.1 .NET Framework

Il **.NET Framework** è un software framework sviluppato da Microsoft che gira principalmente su sistema operativo Microsoft Windows. Include una vasta libreria e fornisce interoperabilità di linguaggio attraverso diversi linguaggi di programmazione. I programmi scritti per il il .NET Framework sono eseguiti in un ambiente software, conosciuto come *Common Language Runtime*, una macchina virtuale che fornisce servizi come sicurezza, gestione della memoria e delle eccezioni. La libreria delle classi e il Common Language Runtime costituiscono insieme il .NET Framework. La Base Class Library del .NET Framework fornisce interfacce utente, accesso ai dati, connettività al database, crittografia, sviluppo di applicazioni web, algoritmi numerici e comunicazioni di rete. I programmatori producono software combinando codice sorgente con .NET Framework e altre librerie. Microsoft fornisce, inoltre, un ambiente di sviluppo integrato per il software .NET chiamato *Visual Studio*. Nell'agosto 2012, è stato rilasciato il *.NET Framework 4.5*, che verrà utilizzato nella realizzazione del progetto di questa tesi.

Design

Le caratteristiche di design del .NET Framework sono:

- *Interoperabilità*: .Net Framework fornisce strumenti per accedere a funzionalità implementate in programmi più o meno recenti che sono eseguiti al di fuori dell'ambiente .NET, al fine di favorire l'interazione tra applicazioni;
- *Common Language Runtime (CLR)*: serve come motore per eseguire il framework .NET. Tutti i programmi .NET sono eseguiti sotto la supervisione del CLR, garantendo proprietà e comportamenti definiti riguardo sicurezza, memoria e gestione delle eccezioni;

- *Indipendenza dal linguaggio*: E' introdotto il *Common Type System* o CTS, che specifica tutti i possibili tipi di dati e costrutti di programmmazione supportati dal CLR e come essi possano o meno interagire con gli altri attraverso le specifiche del *Common Language Infrastructure*, trattato di seguito;

- *Base Class Library*: fornisce classi che incapsulano un gran numero di funzioni comuni, come creazione di interfacce utente, accesso ai file, rendering grafico, manipolazione documenti XML, connettività al database, crittografia, sviluppo di applicazioni web, algoritmi numerici e comunicazioni di rete. E' disponibile per tutti i linguaggi che utilizzano il .NET Framework;

- *Rilascio semplificato*: sono presenti tool che facilitano l'installazione del software sviluppato e assicurano che quest'ultimo non interferisca con software preinstallato e che sia conforme ai requisiti di sicurezza;

- *Sicurezza*: è fornito un modello di sicurezza comune per tutte le applicazioni, oltre a meccanismi di protezione da vari tipi di vulnerabilità;

- *Portabilità*: il framework .NET è *platform-agnostic* e sono disponibili implementazioni multipiattaforma per altri sistemi operativi. Il Common Language Infrastructure, i linguaggi C# e C++ sono tutti standard ECMA e ISO. Questo rende possibile la creazione di implementazioni compatibili col framework da terze parti.

Architettura

Il .NET Framework è costituito da:

- *Common Language Infrastructure*;

- *Sicurezza*;

- *Librerie di classi*;

- *Gestione della memoria*.

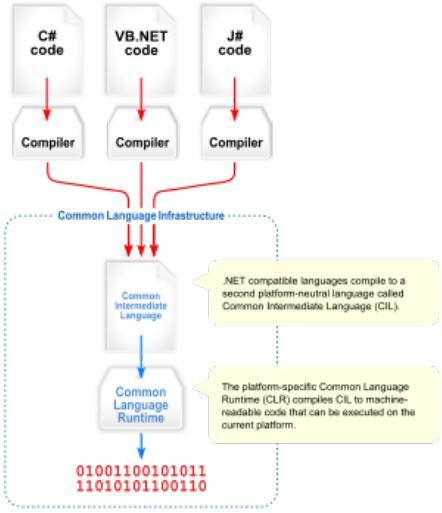

Figura 2.2.1: Panoramica della Common Language Infrastructure

che di seguito sono brevemente introdotti.

Common Language Infrastructure CLI fornisce una piattaforma neutra per lo sviluppo e l'esecuzione delle applicazioni. Tale piattaforma include funzioni per la gestione delle eccezioni, la *garbage collection*, la sicurezza e l'interoperabilità. Siccome gli aspetti basilari del .NET Framework sono stati implementati nell'ambito di CLI, questa funzionalità non è legata ad un singolo linguaggio ma è disponibile a tutti i linguaggi supportati dal framework. Il codice CLI è contenuto in file assembly, conservati in formato *Portable Executable* PE, comune alla piattaforma windows per tutti i file DLL e EXE. L'assembly consiste in uno o più file, uno dei quali contiene il manifest, che rappresenta il metadato dell'assembly. Il nome completo di un assembly contiene nome, numero di versione, *culture*(localizzazione) e chiave pubblica, quindi se due assembly hanno lo stesso nome completo, essi sono considerati uguali.

Sicurezza Il Framework .NET ha meccanismi propri di security con due caratteristiche fondamentali: Code Access Security (CAS) e validazione e verifica. Code Access Security è basato su un *evidenza* che è associata con uno specifico assembly. Tipicamente l'evidenza è la sorgente dell'assembly(qualora sia stato installato sulla macchina locale o sia stato

scaricato dalla rete). Code Access Security usa l'evidenza per determinare le autorizzazioni concesse al codice. Altre porzioni di codice possono chiedere se al codice chiamante sono concesse specifiche autorizzazioni; in tal caso, il CLR esegue un controllo allo stack ed ogni metodo presente nello stack viene controllato sulle autorizzazioni richieste. Se nessun assembly concede l'autorizzazionie, viene lanciata un'eccezione sulla sicurezza.

Librerie di classi Il Framework .NET include un insieme di librerie standard, organizzate in gerarchie di namespaces. La maggior parte delle API presenti sono parti dei namespaces *System.** o *Microsoft.** e implementano un gran numero di funzionalità comuni come, tra le altre, la lettura e la scrittura su file, il rendering grafico, l'interazione con i database e la manipolazione dei file XML. Le librerie sono divise in due macrocategorie: la Base Class Library e la Framework Class Library. La Base Class Library include un piccolo insieme dell'intera libreria ma contiene le API di base del Common Language Runtime. La Framework Class Library contiene la Base Class Library e si riferisce all'intera classe di librerie .NET, includendo, tra le altre, Windows Forms, ADO.NET, ASP.NET, Language Integrated Query (LINQ), Windows Presentation Foundation, Windows Communication Foundation.

Gestione della memoria Il Framework .NET libera lo sviluppatore dalla necessità di gestire l'allocazione e la deallocazione della memoria, dal momento che è presente un sistema che controlla quando la memoria può essere liberata in sicurezza. Le instanze degli oggetti .NET sono allocate nello heap, un pool di memoria gestito dal Common Language Runtime. Quando ci sono riferimenti a quell'oggetto, esso è considerato in uso, ma, quando i riferimenti non ci sono più, l'oggetto non è più raggiungibile e quindi viene cancellato. Il .NET Garbage Collector viene eseguito periodicamente, su un thread separato da quello dell'applicazione, solamente quando una definita quantità di memoria è stata utilizzata o c'è un alto utilizzo di memoria di sistema.

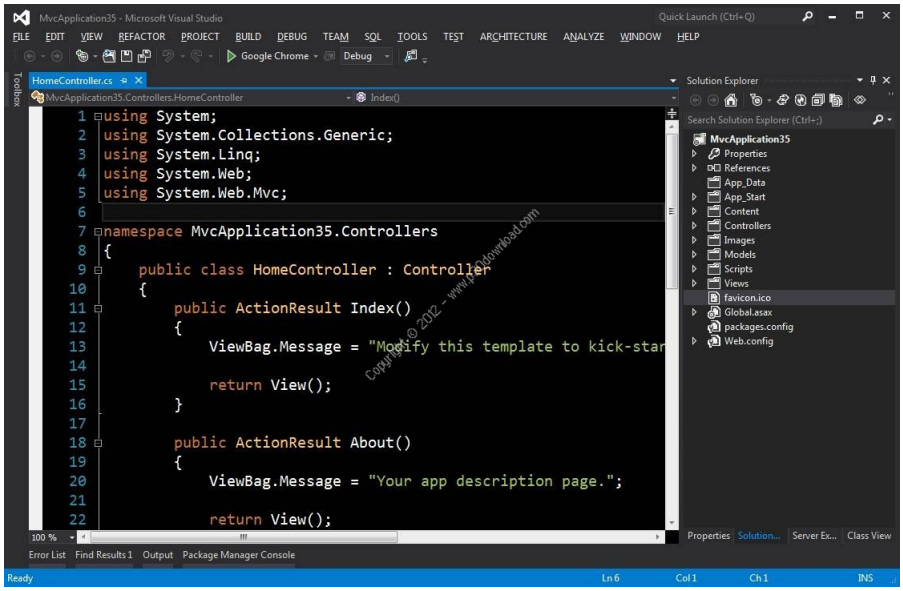

Figura 2.2.2: Schermata di Visual Studio 2012 Ultimate

Microsoft Visual Studio

Microsoft Visual Studio (*http://www.microsoft.com/visualstudio/ita*), rappresentata con uno screenshot in figura 2.2.2, permette lo sviluppo di applicazioni console, siti internet, applicazioni grafiche, applicazioni e servizi web, librerie di classi, servizi Windows in tutti i linguaggi di programmazione supportati dal framework .NET, oltre che in tutti i linguaggi di markup, JavaScript e CSS.

Visual Studio fornisce all'utilizzatore un potente strumento, **IntelliSense**, che permette l'autocompletamento del token che il programmatore intende scrivere, oltre che alla visualizzazione della documentazione e alla gestione delle disambiguità a video direttamente durante la scrittura del codice.

La versione utilizzata per il presente progetto è **Visual Studio 2013**, lanciata nel giugno 2013. Essa fornisce numerose nuove features, brevemente elencate:

- **Colorazione semantica**: è stata migliorata la colorazione della sintassi, compresi i tipi definiti dall'utente, le macro, i tipi enumerativi e le funzioni;
- **Evidenziazione dei riferimenti**: selezione di tutti i riferimenti di un simbolo, una

volta selezionato quest'ultimo;

- **Nuovo Solution Explorer**: il nuovo Solution Explorer consente agli utenti una visualizzazione per classi o gerarchie di file all'interno del progetto. E' supportata inoltre la ricerca di chiamate ad una funzione ed utilizzo delle classi;
- **Visualizzazione automatica della lista IntelliSense**: IntelliSense è automaticamente visualizzato mentre viene scritto il codice;
- **Filtraggio della lista degli elementi**: IntelliSense usa una *fuzzy logic* per determinare quali funzioni/variabili/tipi visualizzare nella lista;
- **Code snippets**: è incluso in Intellisense per generare automaticamente frammenti di codice basati sui parametri dell'utente.

TFS - Team Foundation Service

Visual Studio offre, tra i vari servizi, **Team Foundation Service**, uno strumento di gestione collaborativa del codice che consente di:

- controllare il proprio codice direttamente in *cloud*, rendendolo accessibile ovunque;
- gestire il versioning del progetto;
- gestire la collaborazione nel team;
- pianificare lo sviluppo agile;
- gestire il testing e la distribuzione del prodotto.

2.2.2 Metodologia agile

Con il termine *metodologia agile* si intende un insieme di metodi di sviluppo software basati sullo sviluppo iterativo ed incrementale, in cui i requisiti e le soluzioni evolvono attraverso una collaborazione tra team capaci di organizzarsi autonomamente e con esperienze e conoscenze diverse. Le metodologie agili sono contrapposte alle metodologie *pesanti* e *iterative* poiché promuovono una pianificazione adattabile, uno sviluppo e una consegna del software evolutivi, un approccio iterativo, ed incoraggiano ad una risposta al cambiamento rapida e flessibile.

Le metodologie agili sono state introdotte ufficialmente nel 2001 dal *Manifesto Agile*((25)), un documento dell'*Agile Alliance*, l'associazione che ha permesso la diffusione su ampia scala di tali metodologie. Il Manifesto Agile riporta quanto segue:

> *W*e are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> Individuals and interactions over processes and tools
>
> Working software over comprehensive documentation
>
> Customer collaboration over contract negotiation
>
> Responding to change over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more.

I metodi agili quindi preferiscono la comunicazione in tempo reale, preferibilmente faccia a faccia, a quella scritta (documentazione). Il team agile è composto da tutte le persone necessarie per terminare il progetto software. Come minimo il team deve includere i programmatori ed i loro clienti. (Con clienti si intendono le persone che definiscono come il prodotto dovrà essere fatto. Possono essere dei Product Manager, dei Business Analysts, o i clienti finali). L'obiettivo è la piena soddisfazione del cliente e non solo l'adempimento di un contratto. L'uso di queste metodologia, inoltre, serve ad abbattere i costi di

sviluppo del software e a ridurre al minimo la parte di progettazione che spesso era quella più dispendiosa. Essa è esplosa proprio in concomitanza con la crisi successiva al boom di Internet prendendo spunto dai metodi applicati in piccole software house. Sotto questo nome si raggruppano tecniche come Extreme Programming, SCRUM, Feature Driven Development, DSDM, Disciplined Agile Delivery, Crystal e Lean Software Development.

Processi di sviluppo agile

Così come per i processi di sviluppo tradizionali, anche durante lo sviluppo di software mediante metodologie agili, ci sono delle fasi predefinite: analisi dei requisiti, progettazione, sviluppo e testing. La differenza è che ad ogni iterazione lo sviluppatore ridefinisce e rielabora queste fasi. I requisiti sono, ad ogni iterazione, approfonditi e migliorati, così come è perfezionato il design. Inoltre è dato molta importanza alla rifattorizzazione, ovvero si modifica la struttura interna di porzioni di codice senza modificarne il comportamento esterno, così da migliorarne la leggibilità ed avere sempre codice di qualità. Il testing, nell'Agile, riveste un ruolo fondamentale, poiché sono previsti sia gli *Acceptance Test*, ovvero test black box che rappresentano dei risultati attesi dal sistema.

In figura 2.2.3 vi è una rappresentazione del processo di sviluppo agile.

La gran parte dei metodi agili tenta di ridurre il rischio di fallimento sia in termini economici, poiché si ha la possibilità di stabilire un tetto di spese limitate che è negoziato frequentemente e monitorato su base costante, sia inteso come forte riduzione del rischio che il cliente si ritrovi in mano funzionalità che non utilizzerà mai o molto raramente. Tutto ciò si può ottenere sviluppando il software in finestre di tempo limitate chiamate *iterazioni* che, in genere, durano qualche settimana. I fondamenti dei processi agili sono i seguenti:

- **iteratività**: prescrive che il processo di sviluppo debba essere ciclico, in modo che le varie fasi siano ripetute più volte in momenti temporali diversi. Questo permette di gestire in modo agile i cambiamenti delle specifiche durante il processo, e non

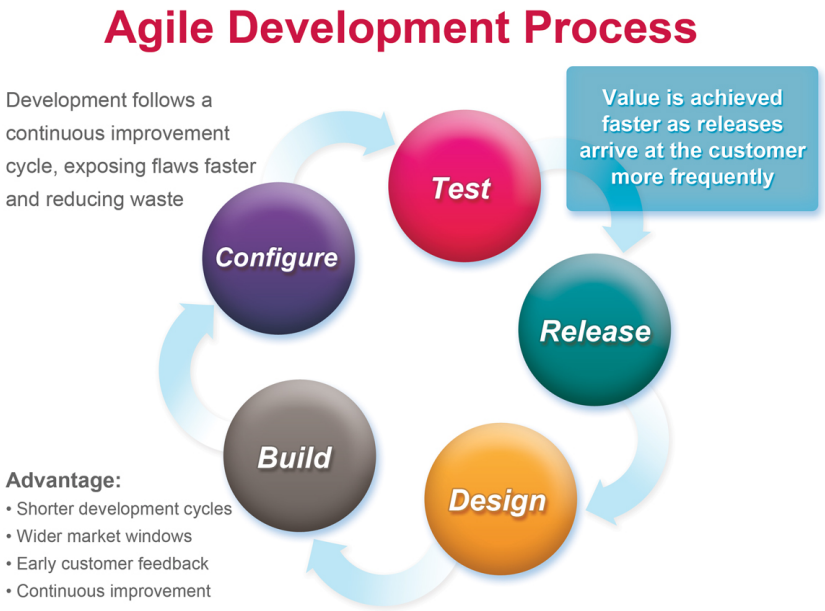

Figura 2.2.3: Processo di sviluppo agile

costringe ad aspettare il rilascio del prodotto per poi intraprendere subito una fase di manutenzione, come invece accade con i metodi tradizionali;

- **incrementalità**: è il continuo rilascio di versioni parziali del prodotto, le quali inglobano modifiche ed aggiornamenti risultati come necessari alle fasi precedenti. Questo meccanismo permette di rilevare i feedback del committente durante il processo di sviluppo e di adeguare opportunamente il software. In alcuni casi il software rilasciato nelle fasi intermedie è sottoposto anche agli utenti finali, in modo da coglierne le esigenze;

- **auto-organizzazione**: il team è lasciato libero di organizzarsi e di adottare di volta in volta le strategie più opportune. Questo favorisce la creatività degli sviluppatori, stimolandoli a trovare soluzioni innovative ai problemi che si presentano;

- **emergenza**: bisogna affrontare difficoltà ed imprevisti quando essi si presentano, senza cercare di predeterminarli o una prevenirli. Il principio tradizionale secondo cui un progetto solido deve tener conto dei possibili sviluppi futuri del software vie-

ne sovvertito, con la motivazione che si considera inutile spendere tempo e denaro per cercare di prevedere evoluzioni che potrebbero essere disattese.

Bibliografia

[argomento] Cognome, N., *Titolo*, Editore, Luogo, Data, ecc.

Siti consultati

[1] http://www.android.com/

[2] http://developer.android.com/sdk/

[3] http://harmony.apache.org/

[4] http//wiki.qemu.org/

[5] http://www.eclipse.org/

[6] http://www.apple.com/it/ios/

[7] http://developer.apple.com/

[8] http://www.windowsphone.com/it-it/

[9] http://dev.windowsphone.com/

[10] http://www.microsoft.com/silverlight/

[11] http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[12] http://www.w3.org/XML/

[13] http://www.json.org/

[14] http://yaml.org/

[15] http://www.gps.gov/

[16] http://www.3gpp.org/

[17] http://research.microsoft.com/en-us/projects/radar/

[18] http://www.ekahau.com/real-time-location-system/technology/

[19] http://www.aeroscout.com/technology/

[20] http://www.cisco.com/en/US/products/ps6386/index.html

[21] https://maps.google.it

[22] http://it.bing.com/maps/

[23] http://www.w3.org/

[24] http://www.w3.org/TR/soap/

[25] http://agilemanifesto.org/