POLITECNICO DI MILANO
Dipartimento di Ingegneria Informatica

APPLICAZIONI TABLETOP MULTITOUCH PER IL TURISMO CULTURALE: UN CASO DI STUDIO MULTIPIATTAFORMA

Relatore: Prof.ssa Franca Garzotto

Tesi di laurea di:
Jun Woo Lee Matr. 711629

Anno accademico 2010/2011

Ringraziamenti

Desidero innanzitutto ringraziare la prof.ssa Garzotto per la grande disponibilità e cortesia dimostratemi, per le opportunità che mi ha concesso, e per tutto l'aiuto fornito durante la stesura della tesi.
Un ringraziamento va sicuramente a Davide Luzzu e Roberto Cavallini di Microsoft per il supporto materiale e tecnico offertomi.
Infine un sentito ringraziamento ai miei genitori, che, con il loro incrollabile supporto morale ed economico, mi hanno permesso di raggiungere questo traguardo.

Sintesi

Il presente lavoro di tesi riguarda lo studio di tecnologie tabletop multi-touch e si propone di mostrare come queste possano essere impiegate in ambito turistico/culturale per promuovere i beni artistici di una città. Viene svolto uno studio critico delle potenzialità e dei limiti di tali tecnologie in ambito generale e nello specifico ambito turistico/culturale. Viene prima considerato un insieme ampio di dispositivi per poi concentrarsi su uno in particolare, Microsoft Surface.

Dopo uno studio dello stato dell'arte delle tecnologie tabletop multi-touch, viene progettato e implementato, in forma prototipale, un applicativo Surface per la città di Milano, che permetta ai visitatori di scoprire i beni artistici presenti in città. Successivamente viene effettuata un'operazione di portabilità dell'applicazione su un dispositivo tablet multitouch basato su Windows 7.

Vengono infine discusse le difficoltà incontrate e gli aspetti positivi e negativi riscontrati nello sviluppo di applicazioni Surface in ambito turistico/culturale e nella portabilità ad altre piattaforme multitouch.

Indice dei contenuti

Indice delle figure

Indice delle tabelle

CAPITOLO 1:
INTRODUZIONE

Nel corso degli anni, l'evoluzione dei computer è stata caratterizzata non solo da una maggiore potenza di calcolo e da una maggiore capacità di memorizzazione, ma anche dall'evoluzione dell'interfaccia utente. Quest'ultima presenta due tappe fondamentali nella sua storia. La prima grande rivoluzione è rappresentata dal passaggio da un'interfaccia a riga di comando, detta CLI (dall'inglese Command Line Interface), a un'interfaccia utente grafica, detta GUI (dall'inglese Graphical User Interface). La seconda ha portato in questi anni alla nascita di un'interfaccia utente naturale, detta NUI (dall'inglese Natural User Interface), chiamata così poiché a differenza dei predecessori l'utente non ha bisogno di imparare a utilizzare gli strumenti offerti, ma controlla l'applicazione attraverso movimenti e gesti naturali che scopre intuitivamente [27] . Gli strumenti di controllo principale per l'utente sono le mani e le dita, tramite le quali controlla l'applicazione o manipola i contenuti. Una delle tecnologie abilitanti principali per la NUI è il multi-touch (o multitouch). I dispositivi touch stanno vivendo un periodo di grande diffusione, soprattutto dopo l'uscita e il successo ottenuto col grande pubblico da parte dell'iPhone di Apple nel 2007 [7] . Solo nel 2009, la distribuzione nel mondo di tali display è aumentata del 29% rispetto all'anno precedente, raggiungendo i 606 milioni di unità, di cui circa il 30% multi-touch, e un profitto di circa 6 miliardi di dollari [7] . E' previsto inoltre che il numero di moduli touch-screen (definiti come combinazione di un sensore touch e di un controller) distribuiti nel mondo continui a crescere con un tasso di crescita composto (CAGR) del 17% raggiungendo nel 2015 circa 1,4 miliardi di unità distribuite per un giro d'affari di 9 miliardi di dollari [6] .

Questi display trovano impiego in ambiti anche molto diversi tra loro. Da molti anni sono utilizzati per bancomat, chioschi automatici (ad es. per le fototessere), in ambito industriale e per altri prodotti non-consumer. Negli ultimi anni stanno rapidamente penetrando nei settori della telefonia mobile, dei navigatori portatili e dei videogiochi. Gli utenti si stanno sempre di più abituando alle interfacce tattili, le quali vengono considerate attraenti e divertenti da utilizzare [6] .

Nonostante questa crescita e diffusione, l'interazione touch ha anche alcuni aspetti problematici. Il tocco non è un'alternativa a tastiera e mouse e non può essere utilizzato per tutti i tipi di applicazioni esistenti. Per esempio, può non essere la migliore soluzione per la stesura di un documento di testo. Il touch, e quindi il multi-touch, non è uno strumento per risolvere vecchi problemi in un nuovo modo, ma piuttosto uno strumento per esplorare nuove possibilità e risolvere nuovi problemi. La domanda da porsi è quindi: cosa posso fare con il multi-touch? Quali nuovi problemi posso risolvere?

Il lavoro svolto nella tesi consiste nello studio dell'utilizzo di tecnologie tabletop multi-touch e relative piattaforme per applicazioni in ambito turistico - culturale. Si vuole studiare quali siano le possibilità offerte da queste tecnologie e quali i limiti, e vedere se possano essere utilizzate per promuovere l'arte e la cultura di una città tramite la creazione di una User Experience (UX) efficace e soddisfacente per l'utente finale. Nello specifico caso si fa riferimento alla progettazione e sviluppo di un applicativo per Microsoft Surface, chiamato DiscoverMilano, che potrebbe essere utilizzato nel comune di Milano per promuovere i propri beni artistici e culturali.

La città di Milano è uno dei maggiori poli turistici d'Italia e d'Europa [8] . È molto conosciuta per gli affari e la moda, ma possiede anche un ampio patrimonio artistico e culturale da valorizzare.

Le soluzioni tecnologiche a supporto del turismo adottate finora dal Comune di Milano si limitano al portale www.visitamilano.it, e al sito www.turismo.milano.it. A livello territoriale, nei due centri di informazione e accoglienza turistica non sono presenti particolari servizi informatici offerti ai visitatori e il servizio si basa essenzialmente sulla presenza di personale disponibile a rispondere alle eventuali domande e sulla distribuzione di mappe cartacee e depliant contenenti informazioni su eventi imminenti.

Il presente progetto di tesi si propone di progettare e sviluppare un applicativo che possa incentivare e promuovere i beni artistici della città ed allo stesso tempo migliorare l'immagine della città e del suo comparto turistico.
Per affrontare il problema, è stato prima di tutto effettuato un lavoro di documentazione sullo stato dell'arte della tecnologia (fonte principale http://portal.acm.org/dl.cfm), studiando le caratteristiche, le potenzialità e i limiti finora analizzati del multi-touch, delle interfacce utente naturali e della configurazione tabletop. Dopo aver individuato in Microsoft Surface un buon candidato per lo studio, data la disponibilità di un SDK liberamente disponibile e di una documentazione abbastanza completa, è stato iniziato uno studio più approfondito dello specifico dispositivo e del relativo ambiente di sviluppo. Nel frattempo, assieme ad alcuni studenti della facoltà di design, è stata avviata la progettazione di alcune applicazioni e della loro User Experience. Conosciute meglio le potenzialità e i limiti di Microsoft Surface è stato selezionato un progetto da sviluppare e rendere oggetto dello studio. Il lavoro di sviluppo dell'applicazione è stato realizzato tramite un normale personal computer grazie alla presenza di un simulatore di Surface nell'SDK, ed è stato seguito da un processo di porting sia per rendere effettivamente touch l'applicazione sia per studiare la portabilità di sistemi Windows tra vari dispositivi.
La struttura del presente documento segue a grandi linee il percorso che è stato intrapreso nello svolgimento del lavoro di tesi.
I risultati del lavoro sono esposti a partire dal *capitolo 2* nel quale si presenta lo stato dell'arte delle tecnologie multi-touch, si introduce Microsoft Surface e si mostrano alcuni lavori di applicazioni multi-touch esistenti per il turismo e i beni culturali.
Nel *capitolo 3* si esamina la situazione corrente del turismo a Milano, si individuano gli stakeholders e i loro bisogni, e si espongono i risultati dello studio dei requisiti della User Experience.
Nel *capitolo 4* si descrive il design della UX con la presentazione dell'interfaccia e dei principali scenari d'interazione.
Nel *capitolo 5* si illustra il design implementativo, descrivendone l'architettura di alto livello e le varie componenti del pattern utilizzato.
Nel *capitolo 6* si descrive l'operazione di porting e i risultati ottenuti.

Nel *capitolo 7* si affronta una discussione sul lavoro svolto e sulle problematiche incontrate.

Nel *capitolo 8* trovano posto le conclusioni e i possibili sviluppi futuri.

Nell'*appendice A* si riportano estratti di codice di DiscoverMilano.

CAPITOLO 2:
STATO DELL'ARTE

In questo capitolo viene presentato lo stato dell'arte delle tecnologie multi-touch, in particolare del tabletop computer Microsoft Surface. Viene presentata la storia e l'evoluzione di queste tecnologie e il concetto di Natural User Interface (NUI). Seguono alcuni esempi di applicazione di queste tecnologie nell'ambito della diffusione e promozione dei beni culturali e del turismo. Infine viene introdotto Microsoft Surface e in particolare la piattaforma per lo sviluppo di applicazioni, Microsoft Surface SDK.

2.1 Le tecnologie multi-touch

I display touch hanno una storia lunga oltre 40 anni, il multi-touch e la gestualità a esso associata più di 25, ma il grande interesse per le interfacce tattili, soprattutto per quelle multi-touch, è esploso nel 2007 con l'annuncio di due prodotti come Microsoft Surface e l'iPhone di Apple. Da allora, la possibilità di interagire con un sistema con le proprie mani e dita è diventata quasi un must in alcuni segmenti di mercato come quello dei dispositivi mobili, dei desktop e dei laptop.

2.1.1 **Storia del multi-touch**

L'utilizzo del tocco per interagire con i computer inizia per lo meno dalla metà degli anni '60 da parte di IBM [1] a Ottawa, in Canada, e da parte della University of Illinois [3] . Nel 1972 i touch screen escono dai laboratori ed entrano nelle aule di alcune scuole elementari americane selezionate, come parte del sistema PLATO IV. Tra gli anni '70 e '80 vengono sviluppate molte tecnologie che supportano il touch e alcune compagnie cominciano a commercializzare queste tecnologie. Queste vengono divise in tre grandi categorie: chioschi (che includono i bancomat), dispositivi point-of-sale (per esempio nei ristoranti e nei negozi), e dispositivi mobili (in principio solo PDA, ma dal 1993 anche telefoni cellulari).
Molti di questi segmenti di mercato non avevano grandi requisiti in termini di ricchezza di tecniche d'interazione. Per esempio le operazioni necessarie per un bancomat sono solo semplici touch-to-select, in altre parole "tocca per selezionare" un comando.
Nel 1984 esce un PDA della Casio, il PF-8000, in cui la parte destra del dispositivo è costituita da un touchpad con cui si possono immettere caratteri alfanumerici. Nello stesso anno commercializzano anche il Casio AT-500, una calcolatrice con touch-screen in cui i numeri e gli operatori sono "scritti" sullo schermo. Una caratteristica di questi dispositivi è il fatto che è possibile digitare senza guardare, come succede con una classica tastiera QWERTY. Se si pensa alle interfacce di oggi questa possibilità di selezionare comandi senza contatto visivo non è molto praticabile. Nel 1993 esce il primo smartphone del mondo, il Simon [14] , realizzato da IBM in collaborazione con Bell South. Questo era dotato di soli due tasti fisici: quello per l'accensione e spegnimento del dispositivo e quello per il volume. Tutto il resto era controllato tramite un pennino o le dita.
Il primo utilizzo documentato di tecnologie multi-touch per il controllo manuale di un sistema digitale è stato effettuato da Nimish Metha [19] come parte della sua tesi all'Università di Toronto nel 1982. È anche il primo caso in cui è utilizzato un sistema di cattura di ombre tramite telecamere per riconoscere il tocco, anticipando alcuni sistemi multi-touch moderni come Microsoft Surface. Nello stesso periodo venivano sviluppati diversi tablet multi-touch, tra cui quello sviluppato da Bill Buxton e dal tuo team alla University of Toronto nel 1984 [3] o dalla Bell Labs a Murray Hill nel New Jersey [4] .

I primi dispositivi touch riconoscevano solo semplici operazioni poke-to-select, ovvero tocca per selezionare. Questo tipo d'interazione, sebbene utile, non rappresentava ancora l'obiettivo da raggiungere. Il desiderio era di estendere la gamma delle gestualità umane riconoscibili dalle tecnologie touch. Uno dei primi a espandere gli orizzonti della cattura della gestualità umana, creando le fondamenta di gran parte dei lavori correnti, è stato Myron Krueger [15] [16] . Il lavoro di Krueger era quasi esclusivamente incentrato alla cattura dei gesti umani e a dimostrare come potessero essere usati efficacemente nell'interazione con il sistema. La tecnologia alla base non era di per sé una tecnologia touch, ma ciononostante fu di fondamentale importanza nello sviluppo di queste, in quanto il suo lavoro fu immediatamente applicato a queste tecnologie. La tecnica del pinching, cioè del pizzicare con due dita per ridimensionare e traslare oggetti è tuttora utilizzata nei dispositivi moderni.

Il primo schermo tattile (e non tablet) multi-touch di cui si ha testimonianza, è quello sviluppato nel 1984 da Bob Boie [2] , Bell Labs, Murray Hill NJ. Esso consisteva in un array capacitivo trasparente di sensori touch sovrapposto a uno schermo CRT. Permetteva di manipolare oggetti grafici con le dita con ottimi tempi di risposta.

Nel 1991 viene presentato Digital Desk di Pierre Wellner [36] . È un primo sistema tabletop a proiezione frontale in cui si utilizzano tecniche ottiche e acustiche per riconoscere sia le mani e le dita che alcuni oggetti, soprattutto dati e controlli su carta. Vengono mostrati chiaramente alcuni concetti del multi-touch come il ridimensionamento e la traslazione di oggetti grafici con due dita, utilizzando i gesti di pizzicare con una mano (pinching) o con un dito per mano.

Esiste un'ulteriore classe di gesti, chiamati menù radiali, che sta avendo un notevole impatto nei sistemi touch di oggi. Comprende quei gesti la cui azione risultante dipende dalla posizione in cui avviene il tocco e dalla direzione in cui esso si muove dopo il contatto. Un esempio comune di questo tipo di gesti, che può essere trovato in molti dei moderni cellulari, è l'abilità di passare da un'immagine alla successiva o alla precedente toccando l'immagine e scorrendo il dito rapidamente verso destra o verso sinistra. Il Portfolio Wall [5] della Alias|wavefront (1999) ne è uno dei primi esempi.

Nei primi anni del nuovo millennio, comincia a prendere piede l'idea di sistemi in cui l'utente non è più uno, ma può essere un gruppo di persone che collaborano.

Questo viene dimostrato dall'uscita di sistemi come il Diamond Touch [9] della Mitsubishi Research Labs e di SmartSkin [31] di Jun Rekimoto della Sony Computer Science Laboratories di Tokyo in cui compaiono caratteristiche di riconoscimento multi-finger, multi-hand e multi-user. Altre caratteristiche sono il riconoscimento della provenienza di ogni tocco e della pressione esercitata da questo. Nel 2003 esce anche un paper [37] che descrive numerose tecniche per il multi-finger, multi-hand e multi-user su una singola superficie touch interattiva.
Negli anni successivi lo sviluppo e le sperimentazioni continuano e i primi dispositivi cominciano a essere commercializzati. Nel 2003 Jazz Mutant, un controller musicale, è il primo dispositivo con schermo multi-touch a diventare un prodotto commerciale.
Il 2007 è l'anno di uscita dell'iPhone di Apple che riscuoterà un grande successo commerciale. È uno smartphone che come il Simon del 1992 si basa su un'interfaccia quasi interamente basata sul tocco. Vengono implementate la tecnica del pinching introdotta da Krueger per zoomare foto e mappe e i menù radiali utilizzati nel Portfolio Wall di Alias per scorrere una sequenza d'immagini.
Il 2007 è anche l'anno dell'annuncio di Microsoft Surface, un tavolo interattivo basato su uno schermo orizzontale multi-touch pensato per essere multi-hand e multi-utente. È inoltre in grado di riconoscere alcuni oggetti e la loro posizione sulla sua superficie.
Negli ultimi anni i produttori di desktop e laptop hanno cominciato a fornire i loro sistemi di schermi multi-touch, soprattutto dopo l'uscita di Windows 7, ultimo sistema operativo di casa Microsoft, che supporta questo tipo di input.

2.1.2 Natural User Interfaces (NUI)

Le Natural User Interfaces, o NUI, sono interfacce utente disegnate in modo tale da poter interagire con i contenuti tramite gesti naturali degli utenti. L'interfaccia può diventare effettivamente invisibile e spesso il contenuto ne diviene l'interfaccia stessa. Sono chiamate 'naturali' perché a differenza delle interfacce grafiche (GUI) o di quelle a linea di comando (CLI) non si servono di controlli artificiali di cui bisogna imparare il funzionamento, ma sono progettate in modo tale che i movimenti e i gesti, da utilizzare per controllare l'applicazione o manipolarne i contenuti, appaiano naturali e che possano essere scoperti velocemente dall'utente. Caratteristica fondamentale delle NUI è l'abilità di riconoscere più tocchi da parte dell'utente, il

multi-touch, che rende l'interazione più naturale e fluida [17] [37] . Il controllo dell'applicazione deve essere guidato dal contesto in cui ci si trova, il che implica anche un restringimento del focus in cui analizzare il contesto. Il numero di percorsi che possono essere intrapresi da un utente in qualsiasi momento della sua esperienza deve essere limitato. I compiti meno rilevanti devono essere limitati o eliminati per potersi incamminare sui percorsi che portano ai compiti più importanti, che creano l'experience all'interno del software. Caratteristica delle NUI è l'abilità di portare la user experience a un livello successivo, combinando sapientemente i comportamenti e i concetti che esistono nel mondo reale con nuove idee che sono possibili solo nella tecnologia. Per esempio è possibile utilizzare i concetti di prendere, trascinare, lanciare foto su un tavolo combinandoli con l'idea di ingrandire e zoomare l'immagine. Ma il fattore differenziale e il cuore dell'esperienza basata sul tocco è la rimozione della barriera tra gli utenti e gli oggetti che questi vogliono manipolare, grazie alla natura fisica della manipolazione diretta dei contenuti. Questo crea un senso di controllo e delle sensazioni che non sono possibili da ottenere con altri paradigmi d'interazione [17] [18] [26] [37] .

2.1.3 **Tecnologie d'implementazioni del multi-touch**

I dispositivi multi-touch consistono in uno schermo tattile o touchpad unito a un software che riconosca simultaneamente multipli punti di contatto. Per implementare il multi-touch vengono utilizzate diverse tecnologie da vari produttori a seconda anche dei requisiti che ogni dispositivo possiede. Le tecnologie più utilizzate sono quelle resistive e quelle capacitive (projected-capacitive). Quest'ultime stanno vivendo un periodo di grande crescita negli ultimi anni da quando sono diventate popolari grazie all'iPhone e all'iPod Touch della Apple.

Tecnologia Capacitiva

I display capacitivi sono composti da un pannello di vetro ricoperto da un sottile strato di ossido metallico sulla parte esterna. Ai quattro angoli del pannello viene applicata una tensione che crea un campo elettrico uniforme su tutta la superficie dello schermo per via dell'ossido di metallo. Quando si verifica un contatto per opera di oggetti, animati o inanimati, che conducono corrente, il campo elettrico subisce

una variazione nel punto di contatto. L'oggetto, per esempio il dito, sfiorando il display, deforma il campo magnetico e questo cambiamento viene percepito da un controller che calcola le coordinate su schermo del contatto.

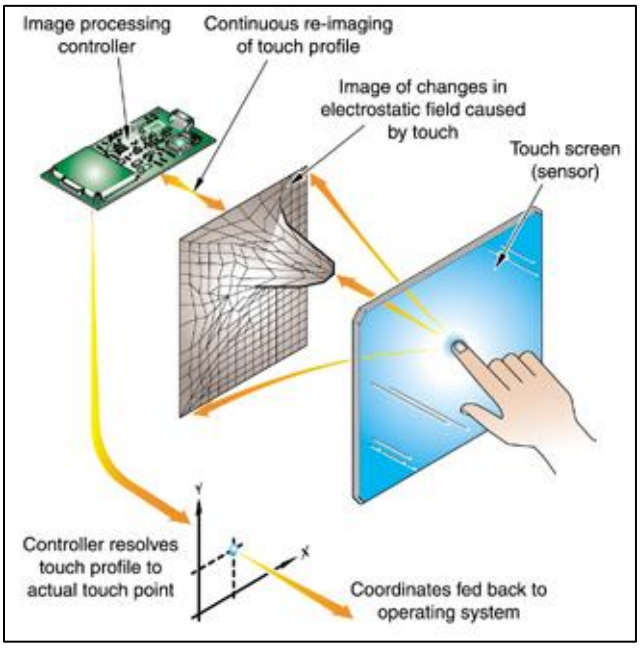

Figura 2.1 Schema di funzionamento della tecnologia capacitiva.

La tecnologia capacitiva è la più grande contendente degli schermi resistivi per il ruolo di tecnologia dominante nel panorama dei touch-screen. Ha una maggiore chiarezza d'immagine ed è meno esposto ai graffi, è resistente all'acqua e alla polvere, ma per ora può essere utilizzato solo con le dita e non funziona con i guanti, con i pennini o altri dispositivi di input, perciò risulta inappropriato per usi in cucina o in ambiente medico.

Tecnologia Resistiva

Da molti anni a dominare in questo settore è invece ancora la tecnologia resistiva. Nasce come tecnologia a singolo touch, ma dal 2009 è presente una versione multi-touch. Questa ha costi decisamente inferiori a quella capacitiva, supporta ora il multi-touch, ma soffre di minor durabilità e prestazioni ottiche. I display resistivi sono costituiti da due strati di materiale plastico, separati da uno spazio. Ognuno di questi strati sovrapposti ha la superficie interna ricoperta di materiale conduttore. Per recepire la selezione, il display deve essere premuto con

un pennino o con qualunque corpo solido, che avvicini i due strati fino a farli entrare in contatto come in Figura 2.2.

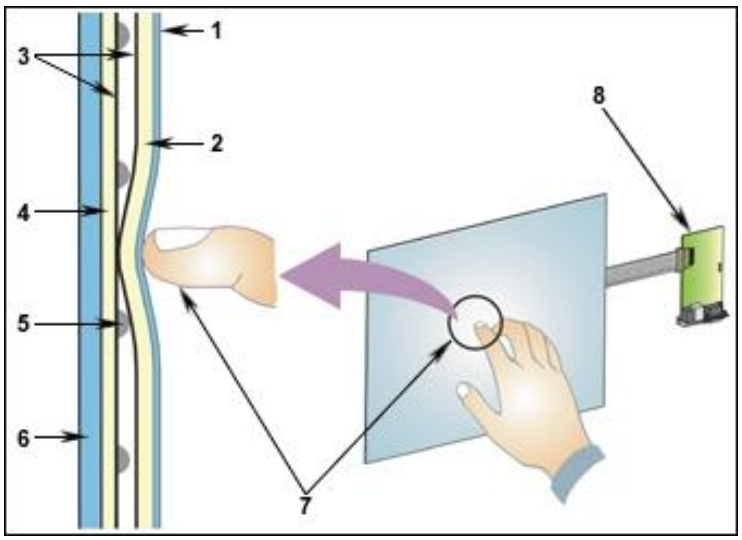

Figura 2.2 Contatto tra i due strati dello schermo resistivo.

Quando si crea un contatto tra due strati, si ha conduzione di elettricità. Il sistema traccia le coordinate di contatto e le traduce di conseguenza. Gli schermi resistivi possono essere usati con quasi qualsiasi dispositivo di input, ma perdono di chiarezza d'immagine alla luce del sole e possono essere graffiati facilmente.

Tecnologia a infrarossi

I touch screen basati su questa tecnologia sono formati da una superficie di vetro circondata da una cornice, trasparente agli infrarossi, contenente LED a infrarossi e fotorecettori installati su lati opposti, che creano una griglia di raggi proiettata sopra il display. Il touch viene rilevato quando un dito o un altro materiale blocca i raggi uscenti dai LED e impedisce ai fotorecettori corrispondenti di riceverli. Il controller calcola la posizione del tocco a seconda di quali recettori hanno riscontrato un blocco dei raggi. In genere con la tecnologia a infrarossi non si riescono a determinare più di due o tre contatti alla volta.

Tecnologia Optical Imaging

La tecnologia ottica o Optical Imaging consiste in due o più sensori d'immagini posti intorno ai vertici (di solito agli angoli) dello schermo. Il display è illuminato da luci infrarosse retroproiettate nel campo visivo delle telecamere. Un tocco è

riconosciuto come un'ombra e ogni coppia di telecamere può calcolare la posizione del tocco o anche misurare la dimensione dell'oggetto toccante.

Tecnologia Surface Acoustic Wave (SAW)

Gli schermi SAW sono costituiti da un pannello di vetro su cui sono trasmesse onde sonore. Ogni onda viene diffusa sullo schermo rimbalzando sui riflettori posti ai vertici dello schermo. Due ricevitori rilevano le onde e quando un utente tocca la superficie del vetro, il suo dito assorbe parte dell'energia dell'onda sonora e il controller è in grado di misurare la posizione del tocco. La tecnologia SAW è spesso utilizzata negli ATM, nei parchi divertimento, in applicazioni bancarie e finanziarie e nei chioschi.

2.2 Applicazione di tecnologie multi-touch nell'ambito dei beni culturali e del turismo

Nell'ambito del turismo e dei beni culturali l'utilizzo di tecnologie multi-touch non è ancora molto diffuso. Quasi sempre quando sono utilizzate, non vanno a sostituirsi agli strumenti già a disposizione, ma a integrarsi a essi per dare un valore aggiunto all'esperienza dell'utente. Questo perché le tecnologie multi-touch non sono pensate per sostituire i vecchi sistemi, ma per creare nuove esperienze.

2.2.1 Schlossmuseum di Linz

All'interno del Schlossmuseum di Linz, in Austria, è presente un'installazione permanente basata su Struktable, un tabletop multi-touch da 70". Vi sono installati diversi giochi multi-utente che hanno come obiettivo quello di educare i visitatori sull'energia solare, sul sistema ferroviario, stradale, idrico e sul turismo nel nord dell'Austria. Il primo gioco si chiama "Solar Land". I visitatori sono invitati a collocare dei pannelli solari sulla mappa del nord Austria (Figura 2.3 a sinistra), cercando di indovinare dove sarebbero più efficienti. Alla fine del collocamento i partecipanti possono iniziare una simulazione che calcola l'energia che avrebbero immagazzinato nel corso di un intero anno solare (Figura 2.3 a destra).

Figura 2.3 A sinistra: Posizionamento dei pannelli solari sulla mappa dell'Austria. A destra: calcolo dell'energia prodotta nelle varie stagioni e totale

I punteggi più alti vengono salvati e i giocatori sono invitati a giocare un'altra volta sfruttando le conoscenze appena acquisite [34] .

2.2.2 **Museo Nazionale di Scienze Naturali di Parigi**

Nell'ambito dell'esibizione "In the Shadows of Dinosaurs" del Museo Nazionale di Scienze Naturali di Parigi sono stati installati dei display interattivi per intrattenere e educare bambini e genitori in modo moderno e innovativo tramite giochi interattivi ispirati ai dinosauri [10] . Le soluzioni tecnologiche scelte sono basate su due tavoli multi-touch Illuminate da 55" della GestureTek (Figura 2.4).

Figura 2.4 Uno dei tavoli della GestureTek installati.

Sono stati creati due giochi semplici da giocare per aiutare i visitatori ad imparare la storia dei dinosauri e delle altre specie che vivevano nello stesso periodo. Un tavolo mostra le diverse specie di animali che vivevano al tempo dei dinosauri e l'altro spiega come questi animali convivessero tra di loro. Più bambini possono utilizzare un singolo tavolo contemporaneamente.

2.2.3 **Parks Canada**

Park Canada opera nel Lower Fort Garry National Historic Site of Canada. Il principale utilizzo delle tecnologie multi-touch è quello di condividere informazioni multimediali sulla storia del sito e sul commercio di pellicce in modo da fornire un'esperienza educativa divertente e memorabile. Vengono utilizzati due tabletop multi-touch Illuminate della GestureTek (Figura 2.5). L'interazione è divisa per tematiche ognuna delle quali offre agli utenti testi, foto, stampe e brevi video sull'argomento.

Figura 2.5 I due schermi tattili al Lower Fort Garry National Historic Site of Canada.

2.2.4 **New York City Visitor Information Center**

Il New York City Visitor Information Center (NYCVIC) si trova tra la 7° Avenue e la 53° strada di New York, al lato di Times Square. Serve visitatori stranieri e locali interessati a conoscere le ultime notizie sugli eventi e sulle attrazioni in città. A supporto di queste operazioni sono stati installati tre tabletop multi-touch della GestureTek, capaci di riconoscere sia il tocco sia gli oggetti. Con i gesti delle dita è possibile ottenere informazioni sulle destinazioni d'interesse, "volare" per le strade di New York tramite Google Earth Fly-Through, costruire un itinerario di viaggio personalizzato con tappe di carattere turistico, di divertimento, di cultura, di shopping e stamparlo, mandare un'e-mail o un sms. È possibile ottenere le notizie più recenti sugli eventi, le aperture, i ristoranti, lo shopping, il teatro e i mezzi pubblici grazie a un database aggiornato [11] .

Figura 2.6 I tavoli multi-touch al NYC Visitors Center. In fondo a destra è possibile vedere uno dei grandi schermi a muro con il lettore a forma di colonna di fronte.

Appoggiando dei dischetti taggati su una specifica località, questa viene memorizzata e spostando i dischetti su degli appositi lettori a forma di colonnina, parte sul grande schermo a muro di fronte (in fondo a destra in Figura 2.6) un video

di Google Earth Fly-Through sulla destinazione memorizzata accompagnata da audio ed effetti sonori che fuoriescono dalle casse interne alla colonnina e dal sistema di casse installato sul muro al di sopra dello schermo.

2.2.5 **Marshall Space Flight Center di Huntsville, Alabama**

Il Marshall Space Flight Center situate a Huntsville in Alabama appartiene alla NASA (National Aeronautics and Space Administration). Un tabletop computer multi-touch, Illuminate della GestureTek, è stato utilizzato per arricchire l'esibizione che ha come oggetto la stazione lunare della NASA. Il lavoro è stato affidato a Inhance Digital Corporation in collaborazione con GestureTek, i quali hanno sviluppato un gioco multi-utente che insegni ai giocatori le basi del funzionamento di uno degli avamposti lunari sviluppati dalla NASA. Ci sono quattro punti di partenza per supportare fino a quattro giocatori simultaneamente. In ciascuna sessione, ogni giocatore ha accesso a diverse componenti della stazione lunare, che possono ruotare e zoomare per ottenere maggiori informazioni.

Figura 2.7 Scelta del componente tra due giocatori al tavolo multi-touch installato al Marshall Space Flight Center di Huntsville

Ogni utente ha la possibilità di trascinare e rilasciare al centro del tavolo pezzi della stazione lunare. Insieme, gli utenti collaborano e assemblano i vari pezzi fino a ottenere la stazione lunare. Successivamente viene presentato un video in cui sono spiegati ulteriori dettagli circa il lavoro svolto alla NASA.

2.2.6 **Eureka Tower di Melbourne, Australia**

L'Eureka Tower è l'edificio più alto di Melbourne. Al piano terra è situato Serendipity, il tavolo multi-touch più grande del mondo. Lungo 6 metri, presenta sulla sua superficie 60 storie diverse riguardo alla mitologia, la cultura, la storia e gli eventi contemporanei di Melbourne. Questi racconti si muovono in un flusso continuo sul tavolo sotto forma d'icone e una volta toccate, si materializzano in forma di finestre informative (information windows) contenenti testo, immagini, video e animazioni (vedi Figura 2.8).

Figura 2.8 Il tavolo multi-touch Serendipity al piano terra dell'Eureka Tower di Melbourne. Gli agglomerati che si trovano al centro del tavolo fluttuano lungo la superficie e quando un utente li tocca, si materializzano le finestre informative che si vedono sulla parte destra dell'immagine.

Molti utenti alla volta possono accedere alle informazioni contemporaneamente semplicemente toccando le icone che fluttuano sullo schermo. La natura dell'interazione è, come la definiscono i loro realizzatori, "serendipitous", ovvero fortunosa e permette al visitatore di fare scoperte per caso piuttosto che seguendo un preciso ordine di storie sequenziali o di narrazione.

2.2.7 **Helsinki CityWall**

Il CityWall [29] di Helsinki è un grande display multi-touch installato in una zona centrale della città. Si propone come interfaccia interattiva e giocosa per il paesaggio sempre in evoluzione della città. Presenta immagini, video, descrizioni che possono essere utilizzati per scoprire qualcosa di più sulla città e permette anche di inviare le immagini agli amici. Il tema dominante è la relazione tra Helsinki e la natura e come questa relazione possa creare sia benefici sia disturbi a entrambi. Per esempio un singolo albero può essere sia un utile riparo, un elemento apprezzabile nel paesaggio ma anche fonte di allergeni e un pericolo per il traffico. Sono presenti anche discussioni a riguardo con l'obiettivo di creare consapevolezza tra le persone riguardo ai disturbi ai quali bisognerebbe adattarsi e a quelli che invece dovrebbero essere controllati o combattuti. CityWall inoltre scarica automaticamente contenuti video e immagini da Flickr e YouTube che siano taggati come 'helsinki', 'citywall' o 'cwhki' consultabili in una sezione separata del CityWall chiamata 'World'.

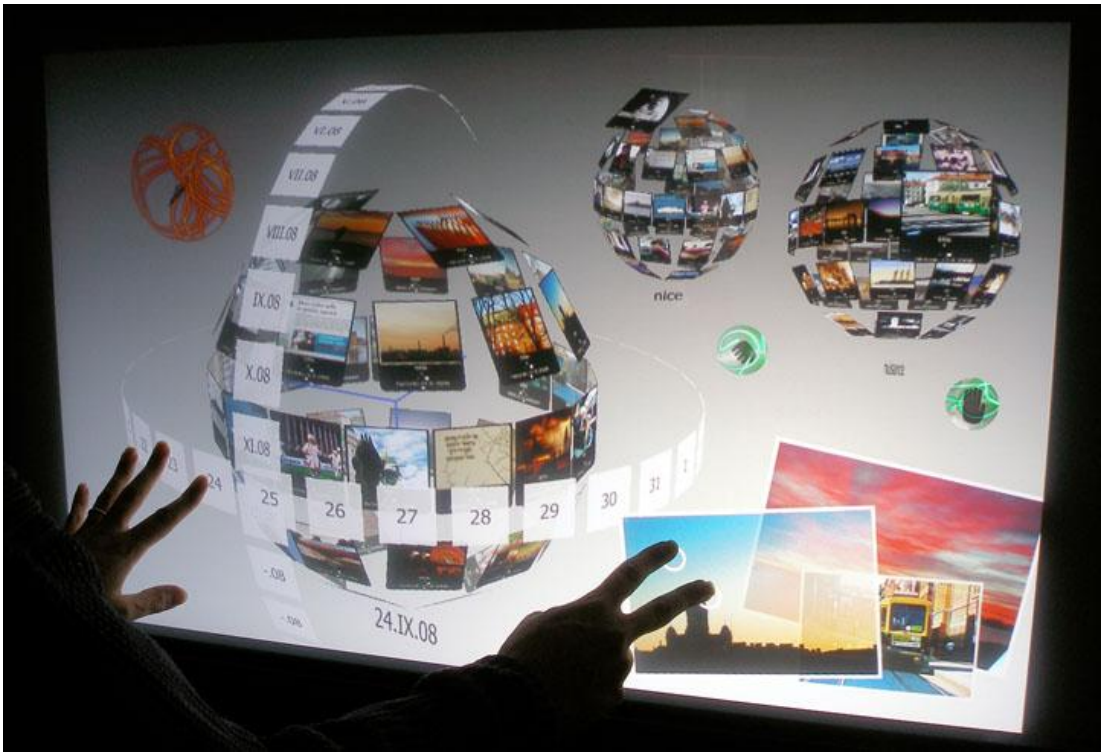

Figura 2.9 CityWall di Helsinki.

2.2.8 **Cambridge Tourist Information Centre**

Ricercatori della Open University hanno disegnato un'applicazione per Microsoft Surface per aiutare le famiglie e i visitatori di Cambridge a sfruttare al meglio il loro tempo nella città e programmare un itinerario per la loro visita. L'applicazione è ospitata da Visit Cambridge presso il Cambridge Tourist Information Centre e mostra una varietà di attrazione turistiche al visitatore, suggerendogli informazioni per sfruttare al massimo il proprio soggiorno [35] .
Il programma permette a fino a quattro persone di decidere le proprie preferenze individualmente per poi decidere insieme, a quali dare la priorità, organizzare un itinerario combinato e stamparlo. Normalmente questo tipo d'interazione non funziona bene tra gruppi di persone e uno degli obiettivi del gruppo di ricerca è di studiare se questo tipo di tecnologie possono aiutare ad aumentare l'experience delle persone.

2.3 Microsoft Surface

Microsoft Surface è un tabletop computer multi-touch con tecnologia a optical imaging sviluppato da Microsoft che risponde a gesti delle mani e a oggetti del mondo reale. Ha un'interfaccia a 360° e può essere utilizzato da uno o più utenti che possono interagire con i contenuti digitali e tra di loro. Fu annunciato pubblicamente il giorno 29 maggio 2007 da Steve Ballmer, CEO di Microsoft, in occasione della conferenza 'D: All Thing's Digital' del The Wall Street Journal a Carlsbad, California [23] e fu rilasciato per la prima volta il giorno 17 Aprile 2008 [20] in collaborazione con AT&T, una delle più grandi aziende di telecomunicazioni americane. Lo schermo ha una diagonale di 30 pollici e la sua superficie non è direttamente sensibile al tocco, ma viene utilizzato un sistema di telecamere per riconoscere diverti tipi di oggetti, che posso essere i polpastrelli delle dita, oggetti taggati o altre forme. L'input viene poi elaborato dal computer e l'interazione viene mostrata sullo schermo tramite retroproiezione con un proiettore. Per lo sviluppo di applicazioni, è stato reso disponibile gratuitamente alla comunità degli sviluppatori un software development kit, Microsoft Surface SDK, che comprende un simulatore per lo sviluppo da workstation tradizionali.

2.3.1 **Architettura di Microsoft Surface SDK**

Microsoft Surface è una piattaforma software e hardware per sviluppare applicazioni multi-input abilitate al tocco. Lo schema ad alto livello dell'architettura della piattaforma di sviluppo è mostrato in Figura 2.10.

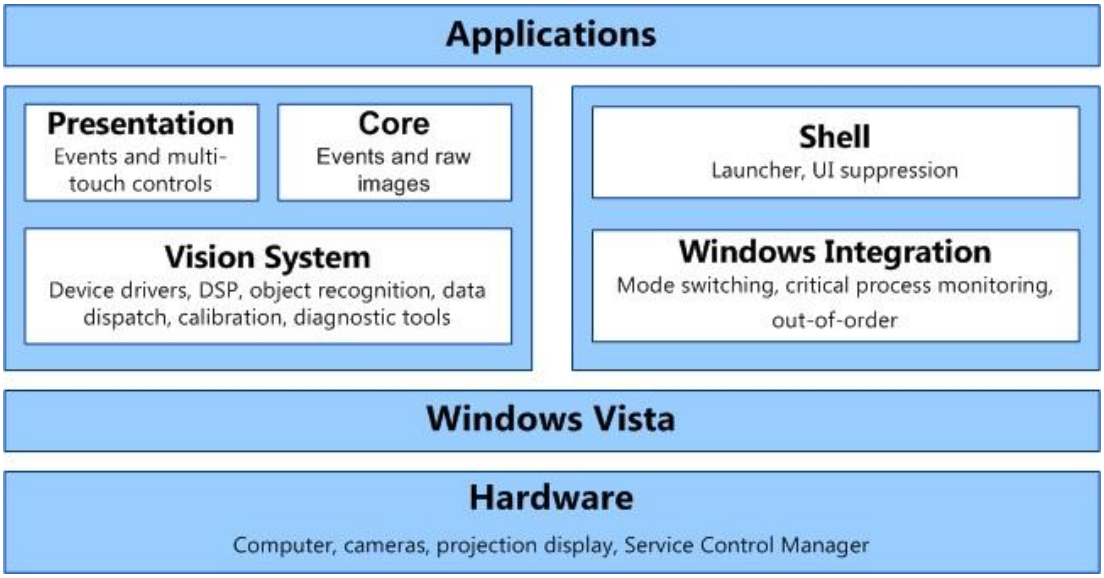

Figura 2.10 Architettura della piattaforma di sviluppo di Microsoft Surface

La piattaforma hardware è costituita da un sistema di videocamere, un display a proiezione e un computer. Questo computer è basato su sistema operativo Windows Vista, il che permette agli sviluppatori e agli amministratori di sfruttare tutte le funzionalità di Windows, come per esempio la possibilità di connettere l'unità a reti, stampanti, lettori di schede, dispositivi mobili e altro (modalità amministratore).
Surface Shell e Surface Window Integration svolgono tutte quelle operazioni necessarie per eseguire applicazioni Surface ma che non interessano l'utente finale. Quando un utente delle applicazioni utilizza Surface (modalità utente), non deve rendersi conto che sta utilizzando un computer. Perciò l'interfaccia utente di Windows viene soppressa e sostituita con l'interfaccia di Surface dalla Surface Shell. È possibile passare dalla modalità utente a quella amministratore (e viceversa) tramite le funzionalità di Windows Integration. Più che un modulo a sé stante, Windows Integration rappresenta l'integrazione tra Microsoft Surface e il sistema operativo Windows, che permette di monitorare processi critici di Microsoft Surface

e gestirne gli errori critici. Tutte le applicazioni Surface si devono integrare con la Surface Shell per essere lanciate.
Le parti del programma che hanno a che fare con l'utilizzo da parte dell'utente finale sono quelle presenti nella parte sinistra dello schema. Il Presentation layer e il Core layer sono i due set di API che possono essere utilizzati per lo sviluppo dell'applicazione e il Vision System è rappresenta tutto ciò che permette alle immagini raccolte dalle videocamere di essere trasformate in informazioni utilizzabili dalle API.
I due layer espongono le stesse funzionalità ma sono pensati per due differenti modelli di sviluppo e lo sviluppatore può scegliere uno solo dei due set da utilizzare. Il Presentation layer si basa su Microsoft Windows Presentation Foundation (WPF) e risulta l'opzione standard per chi voglia sviluppare applicazioni touch perché facilita molti compiti durante lo sviluppo, come la creazione di controlli personalizzati abilitati al touch oltre che di elementi d'interfacce utente classici come Button, Label e barre di scorrimento. Il Presentation layer contiene una suite di controlli WPF standard in versione Surface e supporta l'uso di file XAML (eXtensible Application Markup Language) per uno sviluppo più rapido dell'interfaccia utente. Uno dei controlli che vengono messi a disposizione si chiama ScatterView e permette di manipolare i contenuti inseriti tramite gesti di slittamento e pinching delle mani. Il Core layer al contrario non dispone di questi controlli precostruiti. Espone eventi e dati sui contatti specifici per Microsoft Surface in modo che possano essere sviluppate applicazioni tramite un qualsiasi framework basato su HWND, come Microsoft XNA development Platform, Microsoft Managed DirectX o Microsoft Windows Forms. È consigliato l'utilizzo del Presentation Layer per lo sviluppo di applicazioni che non necessitano di grafica high-end quali complesse animazioni 3D, e quando si vogliano sfruttare i controlli a disposizione. Il Core Layer invece è consigliato in caso in cui l'applicazione da sviluppare necessiti di grafica high-end o rendering con pixel shaders personalizzati. Inoltre è necessario utilizzare il Core layer se si ha bisogno di accedere ai dati delle immagini raw del Vision System.
Il Vision System si occupa di processare i dati video catturati dall'hardware e di convertirli in dati utilizzabili dagli sviluppatori tramite le API.
La Tabella 2.1 descrive più in dettaglio i componenti che caratterizzano l'architettura di Microsoft Surface.

Tabella 2.1 Descrizione delle varie componenti della piattaforma Microsoft Surface.

Componente	Descrizione
Windows Vista	Microsoft Surface gira sul sistema operativo Windows Vista. Vista fornisce tutte le funzionalità di amministrazione, sicurezza e di file system dell'unità Surface. Gli sviluppatori e gli amministratori che lavorano su un'unità Microsoft Surface hanno pieno accesso alle funzionalità di Windows (in modalità amministratore). Tuttavia, quando gli utenti interagiscono con applicazioni Microsoft Surface, l'interfaccia utente di Windows è completamente soppressa (in modalità utente).
Hardware	L'hardware di un'unità Microsoft Surface include il sistema di videocamere, un display a proiezione e un computer con sistema operativo Windows Vista. L'hardware cattura il video dei contatti sullo schermo a uno specifico frame rate.
Vision System	Il software di Vision System processa i dati video catturati dall'hardware e li converte in dati che possono essere utilizzati dagli sviluppatori tramite le API di Surface SDK. È possibile utilizzare uno strumento di calibrazione per configurare le telecamere per performance ottimali. Esistono due tipi di calibrazione: base e completa. Quella completa è utilizzata ogni qual volta che l'unità Surface viene spostata in una nuova locazione. La calibrazione base invece viene utilizzata ogni qual volta le condizioni di luce della locazione cambiano.
Presentation and Core Layers	Microsoft Surface SDK informa le applicazioni quando sulla finestra dell'applicazione di Microsoft Surface appaiono dei contatti. Appena gli utenti appoggiano dei contatti sul display e li manipolano, Microsoft Surface SDK notifica le applicazioni in modo che possano aggiornare la loro interfaccia. Per ogni contatto, le applicazioni possono determinarne la posizione, l'orientamento, la superficie e l'ellisse centrale. Per contatti ottenuti tramite oggetti taggati, le applicazioni possono determinarne il valore di tag. Microsoft Surface SDK espone due set di API: il Presentation layer e il Core layer. Solo un set può essere utilizzato per sviluppare una determinata applicazione Microsoft Surface: • Il Presentation layer si integra con Windows Presentation Foundation (WPF) e include una suite di controlli abilitati a Microsoft Surface. • Il Core layer può essere utilizzato con pressoché ogni framework di interfacce utente.
Surface Shell	Surface Shell è l'elemento che gestisce le applicazioni, le finestre, l'orientamento e le sessioni utente e fornisce altre funzionalità. Ogni applicazione Microsoft Surface si deve integrare con la Surface Shell.
Surface and Windows Integration	L'integrazione tra Microsoft Surface e Windows fornisce funzionalità a tutto il sistema in cima al sistema operativo Windows. Bisogna utilizzare queste funzionalità per supportare aspetti unici dell'esperienza di Microsoft Surface, come per esempio gestire le sessioni utenti, monitorare processi critici di Microsoft Surface, gestire errori critici.

La versione corrente della piattaforma software è Microsoft Surface 1.0 Service Pack 1. Microsoft Surface è compatibile con molti programmi e strumenti già utilizzati da sviluppatori e designer come Microsoft Windows Presentation Foundation (WPF), Microsoft XNA, Visual C# 2008, Visual Studio 2008 SP1, Microsoft Expression Blend 2.

Il Software Development Kit di Surface dispone inoltre di un simulatore che permette lo sviluppo di applicazioni anche ai non possessori di un'unità fisica Surface. Il simulatore però funziona solo su display con risoluzione di almeno 1280 x 960 o con widescreen di almeno 1440 x 900 [21] rendendo impossibile lo sviluppo con portatili che non abbiano uno schermo di almeno 17" con la risoluzione indicata. Inoltre anche con display più grandi (per es. 22") la dimensione del simulatore rimane invariata e non viene permesso di ingrandirne la dimensione, rendendo di fatto difficile studiare e valutare la user experience del software in sviluppo in cui alcune dimensioni (per es. di caratteri e finestre) necessitano di valori minimi per rendere il contenuto fruibile.

2.3.2 **La piattaforma hardware**

La piattaforma hardware è costituita da un computer, un sistema di videocamere a infrarossi e uno schermo a retroproiezione da 30 pollici. Di seguito in Tabella 2.2, le specifiche hardware come dichiarate dal produttore[22] .

Tabella 2.2 Specifiche Hardware Microsoft Surface

Display	Monitor: 30-pollici XGA DLP Scheda video: ATI X1650 con 256MB di memoria Risoluzione massima: 1024 x 768 Durata media della lampada: 6000+ ore Massima pressione sul display: 3.5kg per cm^2 Carico massimo: 14kg
Dispositivi di Input	Sistema di visione basato su videocamere con illuminazione LED diretta
Sistema di computazione	Processore: Intel Core 2 Duo a 2.13GHz Memoria: 2GB DDR2 Dual Channel Storage: minimo 250GB SATA
Audio	Tipo di output: Altoparlanti stereo incorporati Input: nessuno
Protocolli e Standard di rete	Adattatore di rete: Intel Gb LAN Connettività Wireless LAN supportata: Sì Protocolli di rete e dati: IEEE802.11b, IEEE802.11g, Bluetooth 2.0, Gigabit Ethernet
Connessioni I/O	2 jack 3.5mm 6 porte USB 2.0 Uscita video RGB Component Uscita video S-VGA Component Audio Porta Ethernet (Scheda Gigabit Ethernet [10/100/1000]) Porta monitor esterno Vano per cavi Pulsante di accensione/standby
AC input	100-240 VAC, 50/60Hz, 10A, 650W

Le dimensioni fisiche e il peso dell'unità Surface sono come da Tabella 2.3.

Tabella 2.3 Dimensioni fisiche e peso di Microsoft Surface

Dimensioni	Dimensione unità Surface (LxWxH): 108 x 69 x 54 cm
Peso	Con pannelli metallici: 82kg (disponibile solo in U.S. e Canada) Con pannelli acrilici: 90kg

2.4 Ambiente di sviluppo

La piattaforma Microsoft Surface aggiunge a Microsoft Windows Presentation Foundation un set di controlli abilitati al tocco che permettono di creare un nuovo mondo d'interfacce utente. L'Extensible Application Markup Language (XAML) mette a disposizione una sintassi dichiarativa a markup per la creazione di elementi WPF. Al posto di WPF, gli sviluppatori possono utilizzare la piattaforma di sviluppo Microsoft XNA per sviluppare applicazioni per Microsoft Surface. La scelta di quale

piattaforma utilizzare dipende dai requisiti dell'applicazione. Nello sviluppo di applicazioni Microsoft Surface, gli sviluppatori possono disporre di IDE (Integrated Development Environment) come Microsoft Visual C# 2008 Express Edition o Microsoft Visual Studio 2008. Entrambi mettono a disposizione degli sviluppatori gli stessi strumenti utilizzati per creare applicazioni con il Microsoft .NET Framework, quali editor avanzati, gestione dei progetti, debugging e altro.

2.4.1 .NET Framework

Microsoft Surface SDK si basa su Microsoft .NET Framework. .NET Framework è un componente di Windows che supporta la compilazione e l'esecuzione di applicazioni e servizi Web di nuova generazione. I componenti principali di .NET Framework sono il Common Language Runtime (CLR), che gestisce la memoria, l'esecuzione del codice e altri servizi di sistema, e la libreria di classi .NET Framework, che include ADO.NET, ASP.NET, Windows Form e Windows Presentation Foundation (WPF). Common Language Runtime rappresenta la base di .NET Framework e rappresenta l'implementazione da parte di Microsoft dello standard Common Language Infrastructure, e definisce un ambiente runtime per il codice del programma. Gli sviluppatori che utilizzano la CLR scrivono codice in linguaggi come C# o VB.NET e a tempo di compilazione questo viene tradotto in codice CIL (Common Intermediate Language). A runtime il just-in-time compiler di CLR converte il codice CIL in codice nativo del sistema operativo. CLR inoltre fornisce servizi di base quali gestione della memoria, di thread e servizi remoti.
La libreria di classi di .NET Framework è una libreria di classi, interfacce e tipi di valori inclusi in Windows Software Development Kit (SDK). Questa libreria fornisce l'accesso alle funzionalità del sistema ed è progettata come base per la generazione di controlli, componenti e applicazioni .NET Framework.

2.4.2 Windows Presentation Foundation (WPF)

Windows Presentation Foundation (WPF) fa parte del .NET Framework dalla versione 3.0. Obiettivo principale è quello di fornire strumenti per la creazione di applicazioni client per Windows che abbiano un forte impatto visivo. Si poggia su un motore di rendering basato su vettori e indipendente dalla risoluzione, costruito per trarre vantaggio dall'hardware delle moderne schede grafiche. Costruito su DirectX,

riesce a sfruttare l'accelerazione hardware e a fornire funzioni delle moderne interfacce come le trasparenze, i gradienti e le trasformazioni.
WPF offre un nuovo linguaggio di markup chiamato XAML che permette di definire l'interfaccia utente e l'integrazione tra i suoi elementi dichiarativamente. Grazie a WPF è possibile integrare nell'interfaccia elementi come grafica 2-D e 3-D, animazioni, documenti di testo ed elementi multimediali in modo abbastanza semplice. Altre caratteristiche del WPF includono il data binding, il layout, i controlli, gli stili, i template e la tipografia. Il grande vantaggio di WPF è che, occupandosi degli aspetti meno importanti per l'utente finale ma fondamentali per il corretto funzionamento del programma, permette agli sviluppatori di concentrarsi sullo sviluppo della user experience. È possibile sviluppare sia applicazioni web sia stand alone tra cui applicazioni per Microsoft Surface, per il quale esistono controlli specifici abilitati al multi-touch e funzioni apposite. I controlli possono essere pulsanti, finestre di dialogo, documenti, caselle di input, menu, controlli di selezione (checkbox, combo box, listbox, radiobutton, slider), layout (pannelli, griglie, finestre, ecc.) o media (immagini, video, suoni). Esiste inoltre la possibilità di modificare e personalizzare profondamente i comandi.

2.4.3 **Visual Studio 2008**

L'ambiente di sviluppo utilizzato è Microsoft Visual Studio 2008, un ambiente di sviluppo integrato (IDE) di Microsoft. Visual Studio include un code editor che supporta IntelliSense e la rifattorizzazione del codice. Presenta un debugger interno che può lavorare sia a livello di codice sorgente che a livello di codice macchina. Visual Studio supporta diversi tipi di linguaggi tra cui C/C++ (via Visual C++), VB.NET (via Visual Basic .NET), C# (via Visual C#) e F#. Esistono versioni di Visual Studio specifiche per un linguaggio: Microsoft Visual Basic, Visual J#, Visual C# e Visual C++. Inoltre sono presenti le versioni "Express" di Visual Basic, Visual C#, Visual C++ e Visual Web Developer, disponibili gratuitamente.
Per sviluppare applicazioni Surface è necessario disporre di Microsoft Visual C# 2008 Express Edition o di Visual Studio 2008. Per il lavoro corrente è stato utilizzata la versione Visual Studio 2008 Professional Edition.

CAPITOLO 3: DISCOVERMILANO: REQUISITI DELLA USER EXPERIENCE (UX)

In questo capitolo viene presentato lo studio effettuato per individuare i requisiti della User Experience di DiscoverMilano. Identificare i requisiti della User Experience significa collezionare informazioni sulle condizioni in cui bisogna operare. Nel caso del presente lavoro queste condizioni si riferiscono alla tipologia di utente che andrà a utilizzare l'applicativo e all'ambiente in cui verrà inserito e quindi i turisti e il sistema turistico di Milano. I dati utilizzati sono presi da ricerche svolte dall'Osservatorio del Turismo di Milano in collaborazione con le università Bocconi, IULM e Bicocca di Milano, nello specifico da [8] [28] e [30] . Il capitolo è suddiviso nel seguente modo:

- Nel paragrafo 3.1 viene esaminata la situazione del turismo a Milano. Si analizzano la dimensione e i fattori che caratterizzano il settore turistico milanese, viene effettuata una profilazione dei visitatori e infine viene esaminata la presenza e l'utilizzo di tecnologie informatiche a supporto del turismo
- Nel paragrafo 3.2 e 3.3 vengono studiati quali possono essere gli stakeholders e i loro bisogni
- Infine nel paragrafo 3.4 sono individuati e sintetizzati i requisiti da rispettare per soddisfare i bisogni degli utenti

3.1 Situazione del turismo a Milano

La città di Milano è uno dei maggiori poli turistici d'Italia e d'Europa. Con i suoi 3,3 milioni di visitatori ogni anno è la seconda destinazione italiana per presenze dopo Roma, con una capacità di attrazione turistica superiore persino a Venezia e Firenze [8] . Il 70% delle presenze è rappresentato da turismo non-leisure e arriva a Milano per motivi di business, per fiere, congressi o spinti da motivazioni diverse dalla pura vacanza. E' interessante notare come di conseguenza oltre il 60% degli arrivi sia gestito da strutture alberghiere a 4 o 5 stelle. Il 63% dei visitatori presi in considerazione, afferma di non essere alla prima visita in città e la permanenza media in città è in media di 2 notti.
Il 50% dei visitatori proviene dall'estero e i maggiori paesi di provenienza sono Germania, Giappone, Francia, Stati Uniti d'America e Regno Unito mentre i visitatori italiani arrivano principalmente dalle regioni di Lombardia, Lazio e Veneto. Il grado di scolarizzazione è alto con quasi l'80% dei visitatori che possiede almeno un diploma di scuola superiore e circa il 15% in possesso di laurea o dottorato [30] . Tra questi c'è una leggera predominanza di turisti di sesso maschile e della fascia d'età tra i 25 e i 50 anni. Non dovrebbero esserci perciò particolari problemi d'informatizzazione. La tendenza è di viaggiare in compagnia (famiglia, amici, colleghi) e di organizzare autonomamente il soggiorno senza passare da agenzie di viaggio, e il fatto che solo il 16,6% delle richieste all'ufficio informazioni riguardi le strutture ricettive della città (alberghi, ostelli, campeggi) rivela come spesso la maggior parte dei visitatori arrivi già sistemata sotto questo punto di vista. Tra i motivi predominanti della visita a Milano oltre agli affari ci sono il divertimento e la cultura. Infatti, le richieste d'informazioni maggiori presso i centri di accoglienza turistica sono state quelle riguardo alle attività culturali - musei, teatri e spettacoli dal vivo - (27,6%), e quelle riguardanti i servizi turistici - cartine, visite guidate, guide - (28,1%), seguite a distanza da quelle riguardanti strutture ricettive (16,6%) [28] .

3.1.1 L'utilizzo di tecnologie informatiche a supporto del turismo nel comune di Milano

La soluzione offerta dal comune di Milano a supporto del turismo è il portale www.visitamilano.it disponibile sia in italiano sia in lingua inglese [30] . È pensato come un luogo d'incontro tra la domanda e l'offerta turistica. Il Sistema Turistico, inteso come insieme di attori che progettano in modo integrato lo sviluppo turistico del territorio, può pubblicare eventi, luoghi e itinerari d'interesse turistico e i visitatori possono contribuire con propri commenti.
Per quanto riguarda tecnologie sul territorio, non ci sono particolari servizi informatici offerti. Nei due centri di informazione e accoglienza turistica il servizio offerto ai visitatori si basa essenzialmente sulla presenza di personale disponibile a rispondere alle eventuali domande e sulla distribuzione di mappe cartacee e depliant contenenti informazioni su eventi imminenti.
Sono assenti tecnologie tabletop multitouch o postazioni informatiche, dove potersi informare autonomamente e programmare e personalizzare la propria permanenza in città.

3.2 Stakeholders

Dopo lo studio della situazione del turismo a Milano effettuato nel paragrafo precedente, sono stati individuati come stakeholders di DiscoverMilano i seguenti gruppi:

- Stakeholders principali: utenti finali
    - turisti italiani e stranieri: cultura medio/medio-alta, età dai 12 anni in su, con familiarità con le tecnologie, amanti o interessati all'arte e alla cultura.
- Altri stakeholders:
    - centri di informazione e accoglienza turistica
    - centri di arrivo visitatori: stazioni e aeroporti
    - grandi alberghi
    - luoghi d'arte: musei, spazi espositivi, gallerie d'arte
    - eventuali sponsor

3.3 Bisogni

Agli stakeholders individuati, sono stati associati i seguenti bisogni, riassunti poi come matrice Stakeholders/Needs in Tabella 3.1:

- Avere una visione globale dei beni artistici della città
- Possibilità di progettare facilmente un itinerario personalizzato
- Ottenere informazioni sugli eventi culturali e sui mezzi di trasporto
- Incentivare e promuovere l'arte e la cultura di una città
- Rendere l'immagine della città e del suo reparto turistico più accattivante e moderna
- Offrire un servizio unico e di valore ai clienti/visitatori

Tabella 3.1 Matrice Stakeholders/Need di DiscoverMilano

Stakeholders \ Needs		Visione Globale Beni artistici	Itinerario personalizzato	Info su eventi culturali	Promuovere arte e cultura	Marketing della città	Offrire servizi innovativi
Utenti finali	Turisti italiani e stranieri	X	X	X			
Altri stakeholder	Centri di informazione turistica				X	X	X
	Centri di arrivo visitatori: stazioni e aeroporti						X
	Grandi alberghi						X
	Luoghi d'arte				X		X

3.4 Requisiti funzionali

In base ai bisogni da soddisfare individuati, risulta un requisito fondamentale per la user experience la presenza di informazioni, immagini e video dei luoghi di maggior interesse turistico e la presenza di una mappa della città su cui distribuire tali contenuti in modo da avere una visione globale sia dei contenuti che della

posizione geografica di ciascun luogo. La possibilità di creare un itinerario personalizzato e di poterlo fare in collaborazione con i compagni di viaggio può dare un grande valore aggiunto nel creare una forte experience e coprire i bisogni di varie categorie di utenti: membri di gruppi o famiglie con interessi diversi possono mettersi intorno al tavolo e apportare ognuno il proprio contributo all'itinerario; businessman con meno tempo da spendere per le visite possono selezionare i luoghi più vicini al loro hotel; persone che abbiano già visitato la città in precedenza possono selezionare solo i luoghi non ancora visitati.
La possibilità di stampare l'itinerario in formato cartaceo, creando una mini-guida personalizzata, dovrebbe essere un requisito fondamentale fintantoché la possibilità di caricare e consultare l'itinerario sul proprio smartphone non ottenga una diffusione tale da essere in grado di sostituire la stampa.
C'è una forte presenza di turismo straniero a Milano, per cui la possibilità di passare dall'italiano all'inglese (ed eventualmente in altre lingue) è un requisito fondamentale. Le ulteriori lingue a cui si potrebbe pensare di fare una traduzione dovrebbero essere in ordine d'importanza il giapponese, il tedesco, lo spagnolo e il francese per l'alta percentuale di turisti provenienti da questi paesi.

3.4.1 **Requisiti "di ambiente"**

Gli ambienti in cui si vanno a disporre i dispositivi possono essere i vari luoghi d'arte da cui individuare la prossima destinazione o le hall dei grandi hotel e i due uffici di Informazione e Accoglienza Turistica (IAT) situati in piazza Duomo e in Stazione Centrale per programmarsi tutto l'itinerario. A questo scopo può essere utile una funzione "You are here" indicante la posizione in cui si trova lo specifico tavolo Surface.
È importante che il tavolo Surface non sia posizionato in un angolo, ma che sia posizionato in modo che lo spazio intorno sia sufficiente ad ospitare più utenti e permettere l'utilizzo contemporaneo dell'applicazione.
Le location possono essere luoghi più o meno illuminati, ma soprattutto più o meno rumorosi. Si rende quindi opportuno non dare troppo focus all'audio che dovrà essere solo di accompagnamento e di feedback, ma piuttosto focalizzarsi sulla parte visiva dell'applicazione.

Per stampare l'itinerario in formato cartaceo è necessario che vi sia una stampante collegata a Surface. In questo modo è possibile dotare i visitatori di una mini guida turistica personalizzata.

3.4.2 **Sintesi dei requisiti**

I requisiti individuati per la user experience sono dunque:

- Disponibilità di una mappa della città da consultare con geolocalizzazione
- Disponibilità di informazioni sui luoghi d'arte e sugli eventi culturali
- Possibilità di collaborazione tra più utenti
- Possibilità di creazione di itinerari personalizzati
- Disponibilità di un sistema multilingue
- Disponibilità di un collegamento a una stampante
- Collocazione favorevole a un utilizzo multiutente

CAPITOLO 4:
DISCOVERMILANO: DESIGN DELLA UX

Quando si parla di design della User Experience, ci si riferisce alla definizione dei vari ingredienti che contribuiscono a creare questa esperienza. Nel capitolo precedente è stata presentata l'analisi che ha portato alla definizione dei requisiti funzionali di DiscoverMilano. L'analisi dei requisiti è un passo preliminare nel design della User Experience e aiuta a individuare tutte le funzionalità che l'applicazione deve avere. Per creare una User Experience che sia soddisfacente, divertente e coinvolgente è necessario che l'applicazione sia non solo funzionale, ma che sia anche gradevole alla vista e riesca a creare un'affinità con gli utenti e a renderli partecipi di una storia. Nel caso di Microsoft Surface, la possibilità di rendere l'esperienza multiutente contribuisce ulteriormente alla creazione di una forte User Experience. In questo capitolo viene presentata come è stata progettata l'interfaccia per soddisfare i vari requisiti. Segue la presentazione di due tipici scenari di utilizzo.

4.1 Design dell'interfaccia

Data l'importanza di avere un'idea generale dei luoghi d'arte presenti nel territorio, una mappa della città è stata posta al centro dell'applicazione Surface e occupa la maggior parte dello spazio disponibile. La mappa è interattiva e può essere mossa, ruotata e zoomata a piacimento e contiene i pushpin indicanti la posizione dei vari luoghi d'interesse. Per una maggiore pulizia e chiarezza dell'interfaccia sono stati implementati pushpin a comparsa che possono essere resi visibili o meno a

seconda dei filtri selezionati dall'utente. Sono presenti filtri per tipologia di luogo (chiese, musei, monumenti, ecc.), filtri per artista (Leonardo da Vinci, Bramante, ecc.) e filtri temporali. La particolarità di DiscoverMilano risiede proprio nella possibilità di effettuare una scelta degli itinerari sia lungo una dimensione tematica – per tipologia di luogo o di artista – sia soprattutto lungo quella temporale. È infatti possibile ripercorrere la storia dei beni culturali di Milano dal periodo dell'Impero Romano ai giorni nostri e rendere partecipe l'utente di questa storia mostrandogli come si è evoluto il patrimonio culturale della città nel tempo. I controlli per attivare tali filtri sono posizionati tutt'intorno alla mappa. In Figura 4.3 è possibile osservare un prototipo della schermata principale del programma. Si può notare come l'interfaccia sia a 360° in modo che possa essere utilizzato da vari utenti posizionati in diversi punti intorno al tavolo. Inoltre vari colori sono utilizzati sia per rendere l'interfaccia più gradevole, sia per distinguere i vari periodi storici.

Figura 4.1 Schermata principale di DiscoverMilano

Sui lati lunghi dell'interfaccia di Surface è presente la linea temporale indicante tutti i periodi storici/artistici relativi ai beni culturali della città. Selezionando il segnalibro del periodo interessato, ad esempio il periodo del Comune, verranno automaticamente indicati sulla mappa tutti i luoghi d'arte dell'XI e XII secolo, dando più importanza a quelli di maggiore rilevanza artistica tramite distinzione grafica (pushpin di maggiore o minore dimensione).

Su uno dei lati corti dell'interfaccia sono presenti i segnalibri relativi alle diverse tipologie di luoghi d'arte presenti sul territorio milanese (chiese, monumenti, palazzi, musei, ecc.). Selezionando uno di questi segnalibri, ad esempio "Musei", vengono segnalati sulla mappa tutti i musei di Milano (sempre rendendo visibile a colpo d'occhio quelli con maggiore rilevanza artistica). La ricerca può essere eseguita anche incrociando i sistemi di accesso (periodi storici e tipologie di luoghi d'arte presenti sul territorio).

Sull'altro lato corto dell'interfaccia, vedi Figura 4.2, è disposta l'altra tipologia di segnalibri, ognuno dei quali corrispondente a uno dei maggiori artisti italiani, le cui opere/interventi sono presenti sul territorio.

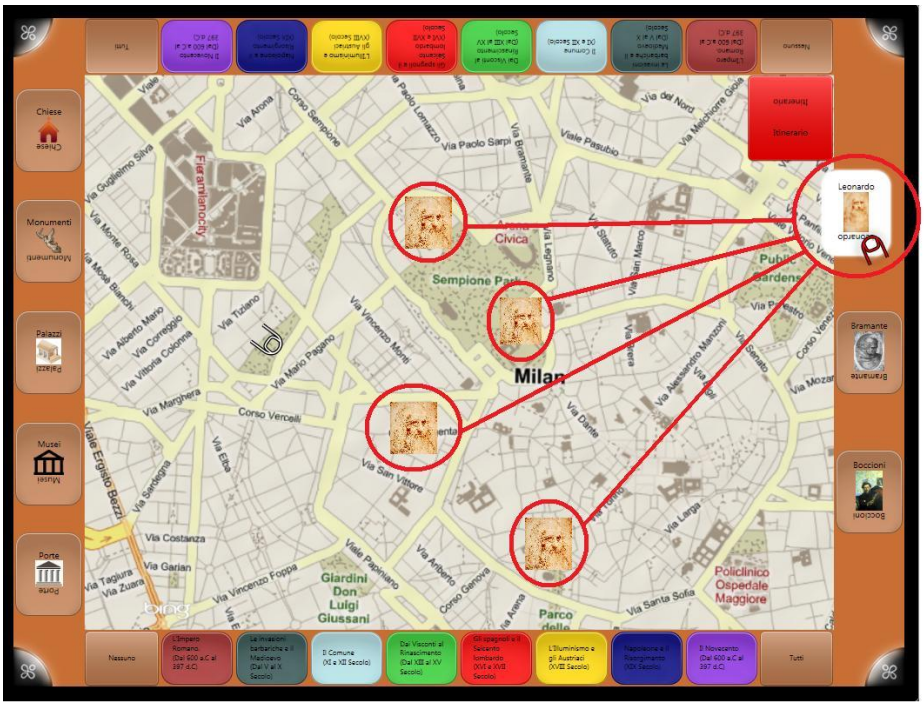

Figura 4.2 Ricerca per artista.

Selezionando un artista vengono visualizzati sulla mappa i punti in cui è possibile trovare le sue opere; in questo modo, il turista appassionato ad esempio di Leonardo

da Vinci, può crearsi un itinerario che comprenda principalmente luoghi concernenti Leonardo.

Figura 4.3 Consultazione finestre informative.

Per visualizzare informazioni riguardanti un sito d'interesse basta premere il relativo pushpin per far apparire una finestra informativa (Figura 4.3), la quale potrà essere mossa, ruotata e ingrandita a piacimento. Queste contengono varie informazioni riguardo al luogo scelto: orari d'ingresso, eventuali costi del biglietto, telefono, indirizzo, descrizione, storia, curiosità, aneddoti e quant'altro possa interessare il visitatore. Oltre alle varie informazioni sono presenti anche altre schede contenenti gallerie d'immagini e video selezionabili dai menù laterali. Durante la fase di ricerca, se un utente è intento a visualizzare le schede riguardanti un sito, un altro ha la possibilità di continuare la sua ricerca modificando i filtri, facendo scorrere o scalando la mappa e selezionando il pushpin desiderato. Inoltre i diversi utenti possono scambiarsi le schede per decidere insieme se aggiungere o no una determinata tappa all'itinerario.

Quando si desidera aggiungere una destinazione è sufficiente trascinare la relativa finestra informativa sulla cartella degli itinerari o premere il tasto Add presente sulla

finestra stessa. Alla fine del processo di selezione, la cartella conterrà il percorso con tutti i luoghi aggiunti dai vari utenti.

Per consultare e stampare l'itinerario creato si preme l'omonimo tasto presente nell'interfaccia e compare la schermata con la lista delle destinazioni scelte come in Figura 4.4.

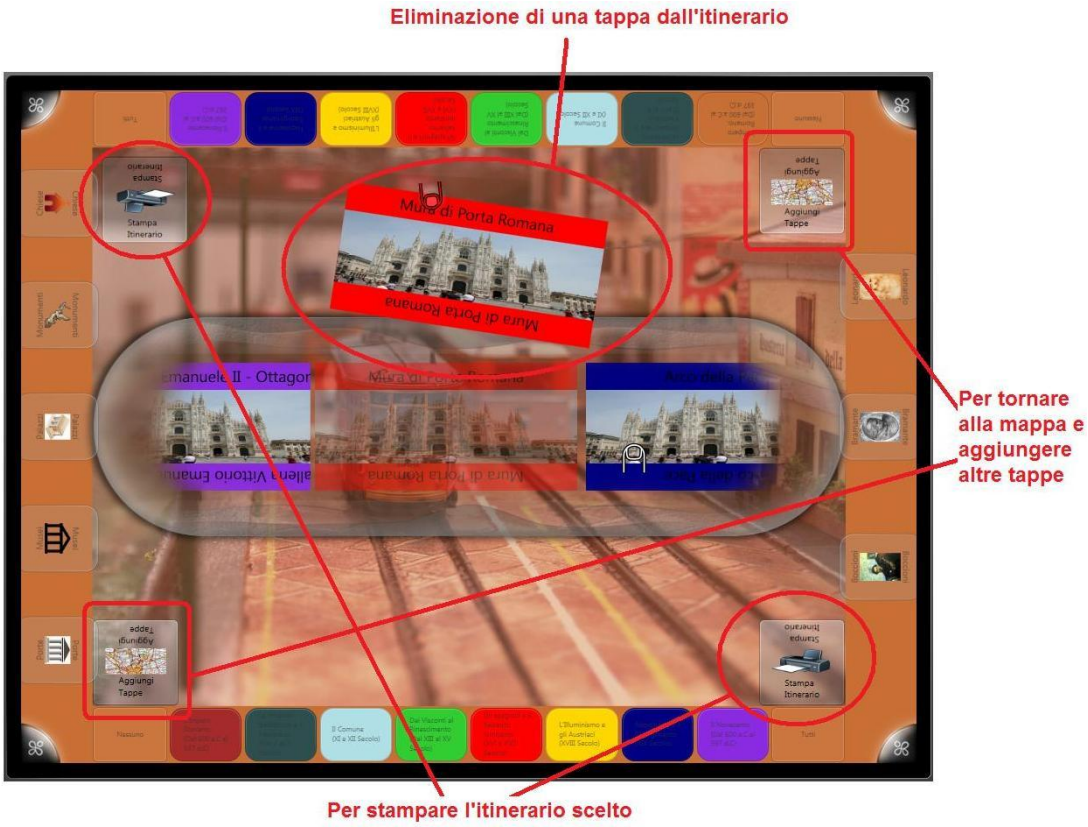

Figura 4.4 Consultazione/modifica itinerario e stampa

È possibile scorrere gli elementi della lista per vedere quali tappe sono state aggiunte all'itinerario. Durante la consultazione è possibile eliminare una tappa trascinandola fuori dalla lista. Per aggiungere ulteriori tappe invece è possibile tornare alla schermata precedente premendo i tasti Aggiungi Tappe. Una volta deciso l'itinerario definitivamente sarà possibile stampare una piccola guida turistica con le informazioni necessarie ad accompagnare il visitatore, premendo il pulsante Stampa Itinerario.

4.2 Scenari di interazione

Discutiamo ora un esempio di come si può svolgere un insieme tipico di interazioni con DiscoverMilano. Per rendere più vivida la presentazione viene utilizzato il concetto di "scenario" [32] , cioè di "storia d'uso". Uno scenario è una descrizione narrativa di quali azioni gli utenti compiono e quali esperienze provano nell'interagire con un'applicazione o sistema. È utile per rappresentare, analizzare e programmare le modalità con cui un sistema influenza il comportamento degli utenti. Nei paragrafi seguenti vengono mostrati due scenari che rappresentano i due insiemi di interazione tipici di DiscoverMilano: la ricerca e l'esplorazione dei beni artistici e culturali di Milano e la consultazione e modifica dell'itinerario scelto.

4.2.1 Ricerca ed esplorazione

I visitatori arrivati intorno al tavolo Surface e iniziano a toccare e a capire il funzionamento di DiscoverMilano e cominciano a scegliere un itinerario. Un utente, Alfredo, è interessato a capire quali luoghi turistici appartengano al periodo rinascimentale e quali tra questi siano chiese mentre un altro, Bernardo, è più interessato ai monumenti del Novecento, perciò utilizzano i filtri combinati per semplificare la ricerca, come mostrato in Figura 4.5.

Figura 4.5 Esempio di ricerca incrociata e di come possano interagire più utenti simultaneamente.

Bernardo trova un monumento che era interessato a visitare e senza pensarci su lo aggiunge all'itinerario trascinando la relativa scheda informativa sulla cartella itinerario. Alfredo invece scopre una chiesa che non conosceva che gli pare interessante. Approfondisce allora la conoscenza sfogliando le informazioni, le immagini o i video presenti nella finestra informativa (Figura 4.6).

Figura 4.6 Consultazione delle finestre informative e aggiunta di luoghi d'interesse nell'itinerario.

Navigando nelle diverse schede della finestra informativa Alfredo scopre un particolare interessante e pensa che possa interessare anche il suo compagno di viaggio.

Alfredo avverte l'amico Bernardo e trascinando con un dito la scheda informativa gliela passa in modo che possa dare un'occhiata anche lui.

Bernardo concorda sul fatto che sia un sito interessante da visitare e aggiunge anche questa destinazione all'itinerario, premendo il tasto Add presente sulla finestra informativa.

L'interazione continua similmente con altre ricerche e altre aggiunte alle destinazioni da visitare: i due amici leggono le storie e le descrizioni di altri luoghi,

sfogliano album di immagini, guardano qualche video e infine aggiungono altre destinazioni all'itinerario.

4.2.2 **Consultazione itinerario**

I visitatori hanno scelto alcuni luoghi d'arte da visitare e decidono di dare un'occhiata all'itinerario prima di stamparlo. Viene premuto il pulsante raffigurante la cartella dell'itinerario e si apre la schermata della lista delle destinazioni aggiunte fino a quel momento (Figura 4.7).

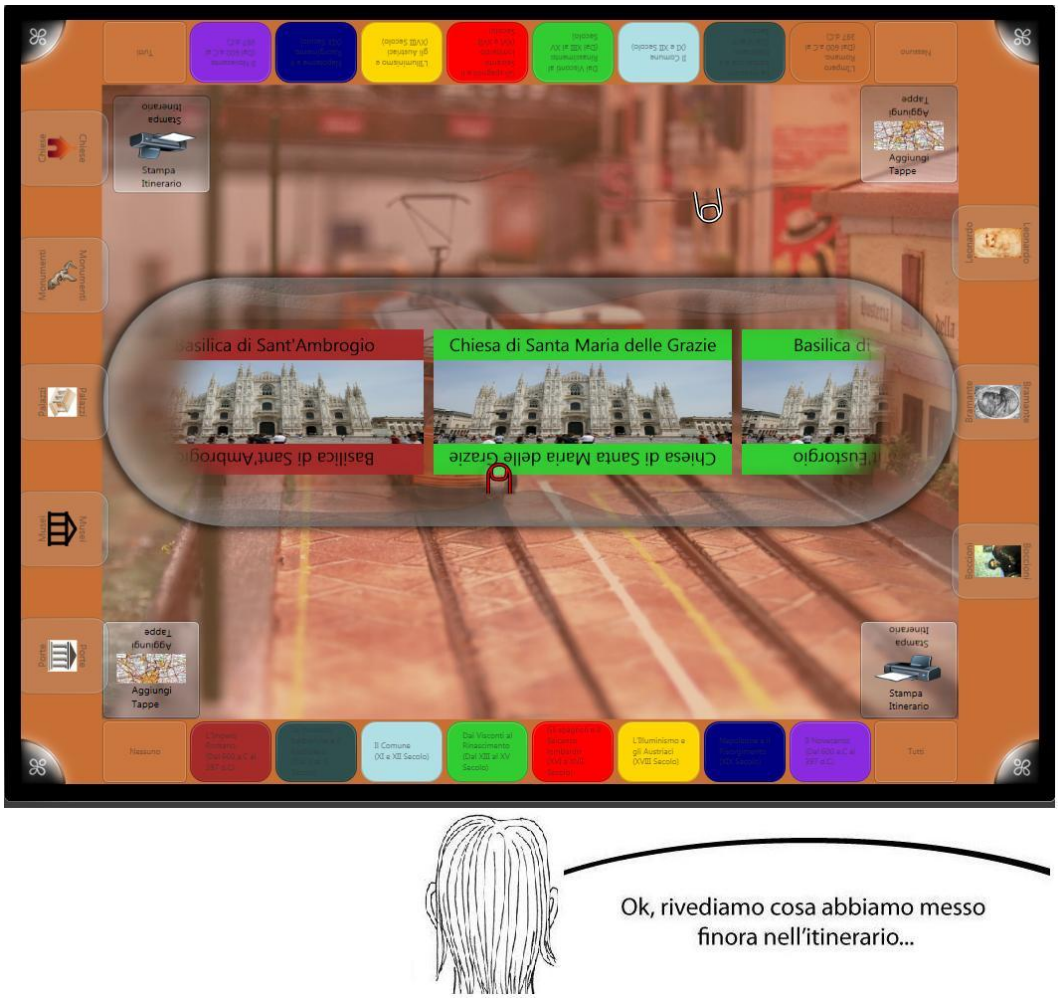

Figura 4.7 Visualizzazione e controllo dell'itinerario scelto.

Notano che hanno inserito troppe tappe e che alcune sono meno interessanti di altre. Decidono allora di eliminarne alcune finché non sono soddisfatti dell'itinerario risultante (Figura 4.8).

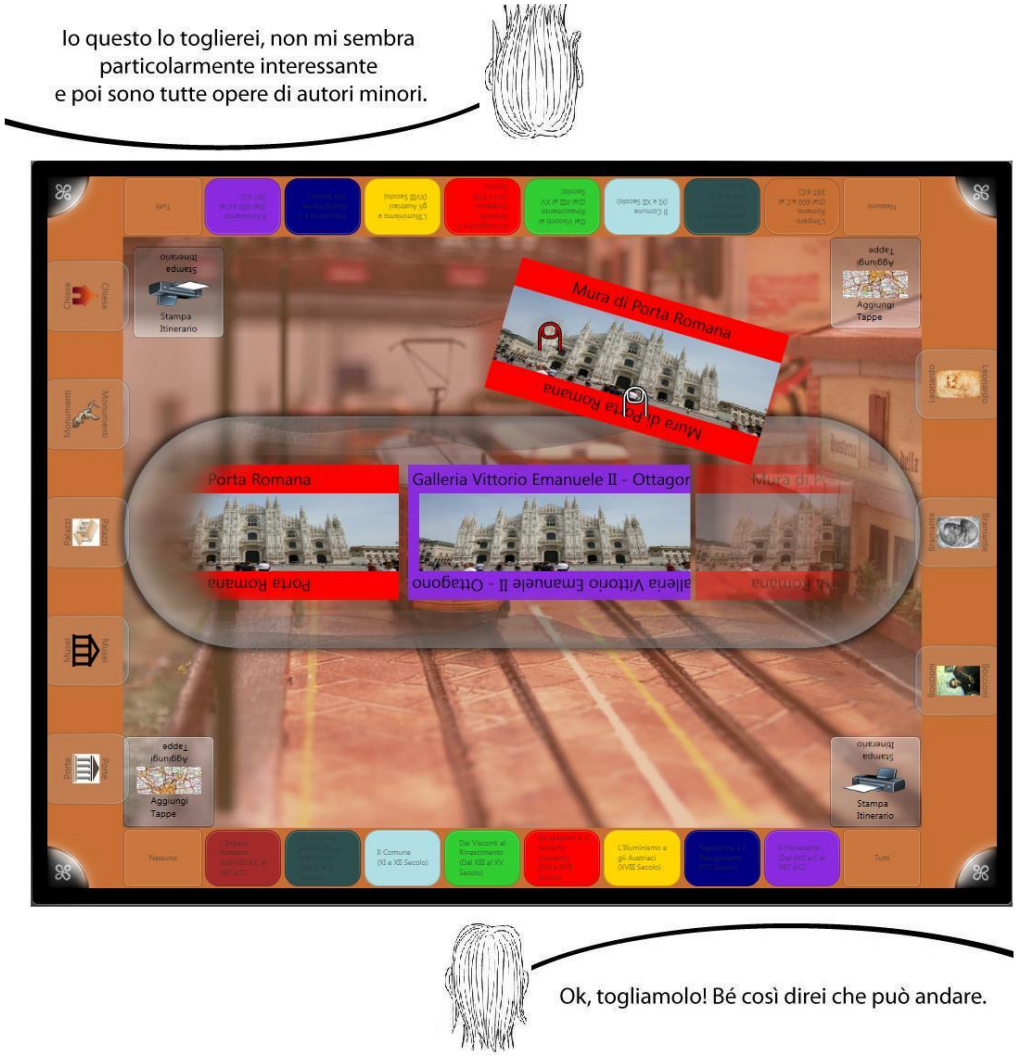

Figura 4.8 Modifica dell'itinerario.

L'itinerario è ora pronto per la stampa. Uno degli utenti preme il pulsante "Stampa Itinerario", lo ritira e insieme escono soddisfatti della loro mini-guida personalizzata e vanno a visitare i luoghi scelti.

CAPITOLO 5: DISCOVERMILANO: DESIGN IMPLEMENTATIVO

In questo capitolo viene mostrata l'architettura implementativa di DiscoverMilano basato sul pattern Model-View-ViewModel. Si mettono in mostra le relazioni tra i vari elementi del pattern e come questi si mappano con le classi dell'applicazione. Successivamente sono presentati alcuni dettagli riguardanti tali classi e su come sono connesse tra loro. Infine vengono mostrate le strutture dati utilizzate per ospitare i contenuti.

5.1 Architettura implementativa

Per lo sviluppo di DiscoverMilano è stato utilizzato il set di API del Presentation layer poiché si basa su Microsoft Windows Presentation Foundation (WPF). WPF risulta la scelta standard per lo sviluppo di applicazioni Surface grazie alla presenza di molti elementi d'interfaccia disponibili in una versione adattata al tabletop di Microsoft e la possibilità di personalizzare i controlli in modo più semplice rispetto al Core layer (si veda Paragrafo 2.3). Per lo sviluppo di applicazioni WPF esiste un design pattern chiamato Model-View-ViewModel (MVVM), una variante del pattern MVC che sfrutta le caratteristiche di WPF [12] [33] . In MVVM, il controller di MVC è sostituito dal viewmodel, che, come suggerisce il nome, rappresenta un modello per la view, ovvero espone i dati e i comandi di cui la view ha bisogno. Si può pensare al viewmodel come un contenitore dove la view va ad attingere dati e

comandi. Esso inoltre non è un oggetto generico che può servire diverse view, ma è specifico per ogni vista. Il viewmodel a sua volta preleva i dati dal model. La caratteristica principale del MVVM pattern è l'utilizzo del binding, che è lo strumento tramite il quale i dati tra view e viewmodel vengono legati e si mantengono sincronizzati.
L'architettura di DiscoverMilano si ispira al pattern MVVM anche se non viene seguito alla lettera in quanto trasferisce parte della logica di business nella classe SurfaceWindows1.xaml.cs, il cosiddetto Code Behind della view. Lo schema dell'architettura di DiscoverMilano è mostrato in Figura 5.1.

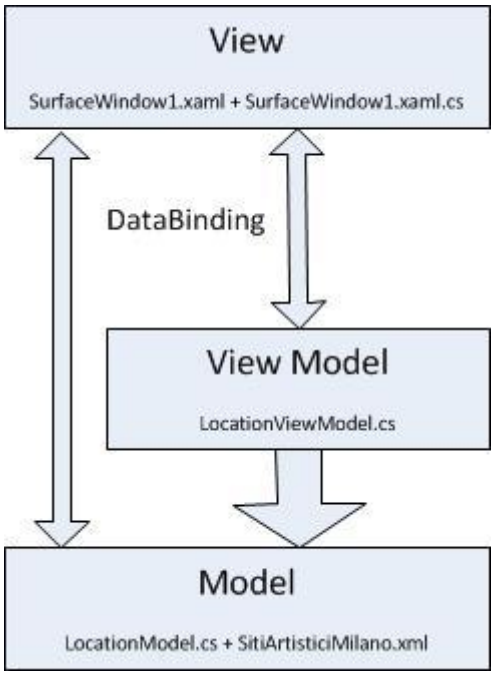

Figura 5.1 Schema architettura implementativa.

Il model è rappresentato dai dati relativi ai luoghi d'interesse artistico e contiene informazioni quali le coordinate, il tipo di luogo (museo, monumento, chiesa, ecc.), il periodo storico di appartenenza, gli orari di apertura, la descrizione, le immagini e i video. Tutti questi dati sono presenti nel file SitiArtisticiMilano.xml di cui la classe LocationModel.cs ne rappresenta un'astrazione.
Il view è rappresentato dalla classe SurfaceWindow1.xaml che accoglie l'interfaccia utente e riunisce tutti gli elementi descritti nel paragrafo 4.1. Inoltre contiene eventi e data-bindings che rappresentano il punto d'integrazione e sincronizzazione con il

viewmodel e il model. Assume un ruolo attivo in quanto esegue un push dei dati verso gli altri componenti, ma è tutto gestito dal framework tramite i binding.
Il viewmodel è rappresentato dalla classe LocationViewModel.cs e si occupa principalmente di trasferire i comandi dalla view al model e di rendere disponibili i dati del model alla view, tranne per i dati in cui view e model sono direttamente legati. Inoltre è responsabile per l'aggiornamento dello stato della vista. Essendo basato su WPF, DiscoverMilano utilizza i linguaggi di programmazione xaml e C# per l'implementazione.
Il diagramma UML delle classi è mostrato in Figura 5.2. Nei paragrafi successivi vengono presentate le singole classi e mostrati i loro comportamenti tramite la spiegazione di parti del codice o di diagrammi UML. Per una maggior leggibilità del documento, le porzioni di codice troppo lunghe vengono inserite in Appendice e ove possibile viene presentata una visione più concettuale della struttura dell'applicazione tramite diagrammi UML.

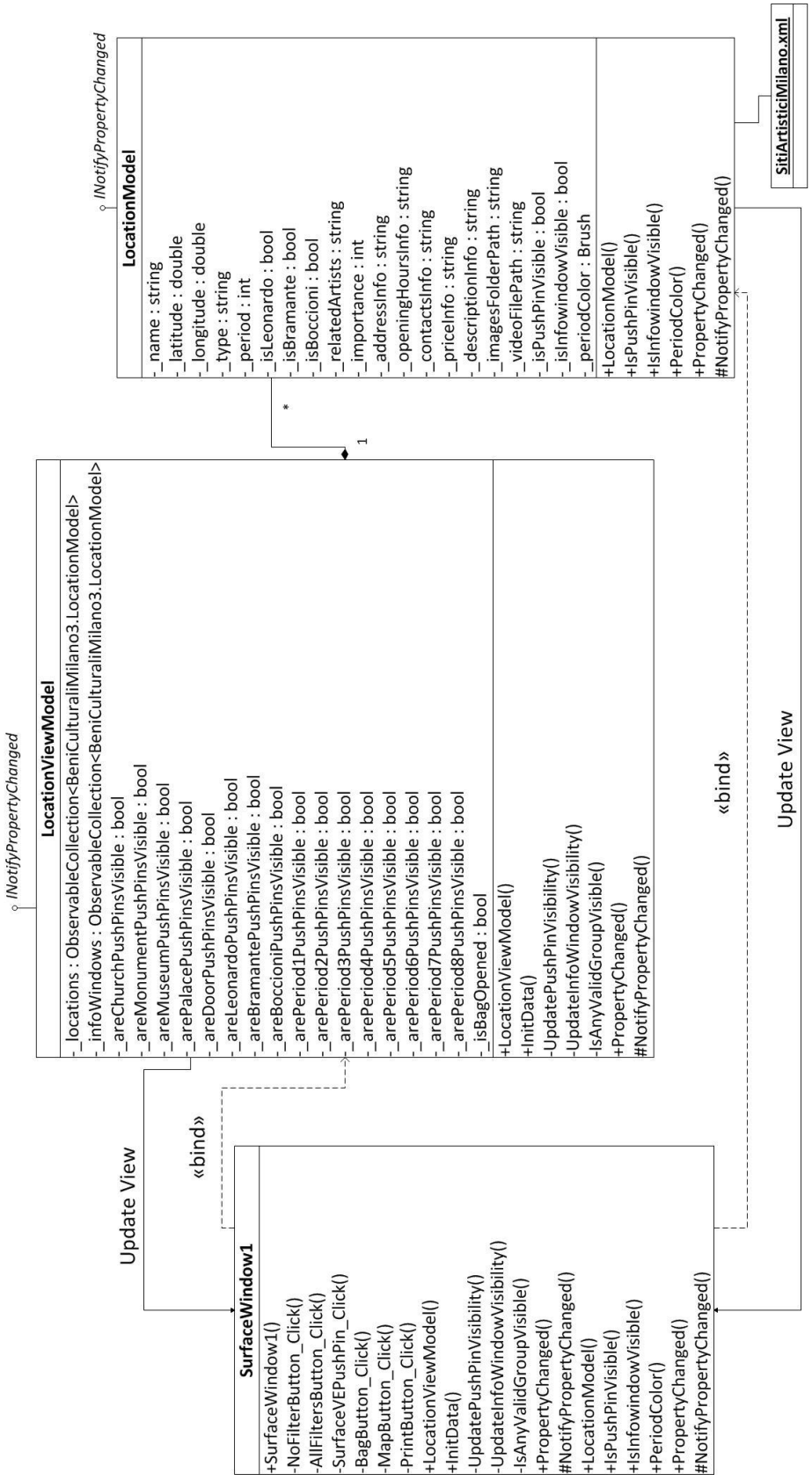

Figura 5.2 DiscoverMilano: diagramma UML delle classi

5.2 Model

Il dominio dei dati è rappresentato dalle informazioni riguardanti i vari luoghi d'arte. Ogni luogo d'arte è caratterizzato da un nome, coordinate cartesiane, tipologia di luogo, periodo storico di riferimento, rilevanza artistica, informazioni generali e di contatto, descrizione, aneddoti, curiosità e una serie di foto e video. La classe LocationModel rappresenta il modello di DiscoverMilano e contiene:

- le proprietà rappresentanti tutte queste informazioni (name, latitude, longitude, ecc.)
- il costruttore
- una variabile booleana, IsPushPinVisible, per segnalare se il singolo pushpin che si riferisce a un determinato luogo è visibile oppure meno sulla mappa.

Un oggetto LocationModel contiene quindi tutte le informazioni riguardanti un singolo luogo d'arte. In Figura 5.3 è possibile vedere la rappresentazione UML di questa classe.

Figura 5.3 Rappresentazione della classe LocationModel in UML

In Appendice A è possibile trovare il codice sorgente della classe, in cui si può vedere come la visibilità dei pushpin venga inizializzata a false per non avere un numero ingestibile di pushpin sullo schermo all'avvio dell'applicazione. Nel paragrafo 5.4 invece viene mostrato come viene aggiornato il valore di IsPushPinVisible ogni volta che si aggiorna lo stato dell'applicazione.

5.3 View

La vista di DiscoverMilano è rappresentata dai controlli che formano l'interfaccia ed è codificata nel file SurfaceWindow1.xaml. Xaml è un linguaggio che deriva da Xml ed è utilizzato per creare dichiarativamente interfacce utente.

L'interfaccia di DiscoverMilano è formata da una "cornice" di pulsanti SurfaceToggleButton utilizzati per modificare lo stato della vista, e dal contenuto centrale rappresentato dalla mappa interattiva e da un layer di oggetti ScatterView, un controllo Surface che permette di muovere, ruotare e ridimensionare oggetti liberamente. In Appendice A è possibile consultare un estratto del codice presente in SurfaceWindow1.xaml che rappresenta la struttura della view di DiscoverMilano.
Per visualizzare la mappa viene utilizzato il servizio di Microsoft chiamato Bing Maps. Per renderlo abilitato al multi-touch e alla sua gestualità e integrarlo con WPF è stato utilizzato un controllo esterno chiamato InfoStrat.VE, sviluppato da Information Strategies (InfoStrat) [13] . Per utilizzare il controllo SurfaceVEMap messo a disposizione da InfoStrat è necessario aggiungere le librerie referenziando due file dll (InfoStrat.VE.dll e InfoStrat.VE.NUI.dll) in Visual Studio e inserire il riferimento allo spazio dei nomi come segue:

Tabella 5.1 Xml Namespace per SurfaceVEMap

```
<s:SurfaceWindow
…
xmlns:ve="clr-namespace:InfoStrat.VE.NUI;assembly=InfoStrat.VE.NUI"
…>
```

Infine è sufficiente aggiungere il controllo SurfaceVEMap al codice XAML utilizzando il nome della libreria stabilito come in Tabella 5.2:

Tabella 5.2 Controllo SurfaceVEMap in SurfaceWindow1.xaml

```
<Viewbox Stretch="UniformToFill">
   <ve:SurfaceVEMap Name="map"
                    LatLong="45.468913167715065, 9.181029796600341"
                    Altitude="7000" MapStyle="Road"
                    ItemsSource="{Binding Locations}"
                    ItemTemplateSelector="{StaticResource
                    templateSelector}" >
   </ve:SurfaceVEMap>
</Viewbox>
```

Per visualizzare le finestre informative viene utilizzato il controllo ScatterView come mostrato in Tabella 5.3:

Tabella 5.3 Finestre informative implementate tramite ScatterView

```
<s:ScatterView x:Name="InfoWindowsSV"
               ItemsSource="{Binding InfoWindows}"
               ItemTemplate="{StaticResource InfowindowTemplate}">
                …
            </s:ScatterView>
```

Scatterview è un controllo fornito per Surface e fornisce le funzionalità per muovere, ruotare e ridimensionare con le dita i suoi elementi.
Si può notare come la mappa "prelevi" i dati riguardanti i luoghi d'arte tramite il binding con la proprietà Locations e applichi i template per definire la struttura di controlli utilizzata per presentare tali dati. Lo stesso vale per il layer ScatterView delle finestre informative. Questi dati verranno forniti dal viewmodel. Per i pushpin della mappa viene utilizzato un TemplateSelector per selezionare un template diverso a seconda del tipo di luogo d'arte per ottenere un aspetto diverso a seconda che esso sia una chiesa, un monumento, un museo, ecc. Per le finestre informative viene applicato direttamente il template scelto. Il template è ciò che definisce l'aspetto e gestisce la visibilità del singolo pushpin tramite il binding tra l'attributo PinVisible del SurfaceVEPushPin e la proprietà IsPushPinVisible del LocationModel (Tabella 5.4). Questo significa che a ogni cambiamento del valore di IsPushPinVisible nel modello, corrisponderà un cambiamento di visibilità del PushPin sulla mappa. Per quanto riguarda le finestre informative invece, la visibilità viene implementata aggiungendo o rimuovendo un elemento dalla collezione InfoWindows, la fonte di dati di InfoWindowsSV (Tabella 5.3). Nella Tabella 5.4 è possibile vedere il template di un singolo pushpin e si può notare come tutti gli attributi non abbiano un valore esplicito, ma siano "legati" a una proprietà del model.

Tabella 5.4 Esempio di Template e TemplateSelector

```
<DataTemplate x:Key="ChurchPushPinTemplate">
   <ve:SurfaceVEPushPin Latitude="{Binding Latitude}"
                        Longitude="{Binding Longitude}"
                        PushPinBackground="{Binding PeriodColor}"
                        PinVisible="{Binding IsPushPinVisible}"
                        Height="{Binding Importance}"
                        Width="{Binding Importance}"
                        Click="SurfaceVEPushPin_Click">
      <Image Source="C:\...\icon-church.png" Stretch="Uniform"/>
   </ve:SurfaceVEPushPin>
</DataTemplate>

<local:PushPinTemplateSelector x:Key="templateSelector"
      TemplateChurch="{StaticResource ChurchPushPinTemplate}"
      TemplateMonument="{StaticResource MonumentPushPinTemplate}"
      TemplateMuseum="{StaticResource MuseumPushPinTemplate}"
      TemplatePalace="{StaticResource PalacePushPinTemplate}"
      TemplateDoor="{StaticResource DoorPushPinTemplate}"
      StandardTemplate="{StaticResource StandardPushPinTemplate}"/>
```

Un altro esempio di binding viene dai filtri temporali, utilizzati per gestire la visibilità dei pushpin:

Tabella 5.5 Binding per aggiornamento visibilità PushPins

```
<s:SurfaceToggleButton x:Name="Period1Button_Nord"
IsChecked="{Binding ArePeriod1PushPinsVisible}" … >
    <TextBlock…> … </TextBlock>
</s:SurfaceToggleButton>

…

<s:SurfaceToggleButton x:Name="ChurchButton" IsChecked="{Binding
AreChurchPushPinsVisible}" … >
    <Grid > … </Grid>
</s:SurfaceToggleButton>
```

In questo caso gli attributi sono legati a proprietà del ViewModel, implementate per gestire lo stato dell'applicazione

Le finestre informative che vengono aperte selezionando i vari pushpin funzionano secondo lo stesso schema. Un template definisce la struttura dei controlli che il binding popola di dati:

Tabella 5.6 Estratto del template per le finestre informative

```
<DataTemplate x:Key="InfowindowTemplate" >
    <Grid Background="{Binding PeriodColor}" >
        <Grid.RowDefinitions>…</Grid.RowDefinitions>
        <Grid Grid.Row="0"> … </Grid>
        <Grid Grid.Row="1">
            <TabControl>
                <TabItem Header="Descrizione">
                    …
                    <TextBlock Text="{Binding Name}"…/>
                    …
                    <TextBlock Text="{Binding AddressInfo}"…/>
                    …
                    <TextBlock Text="{Binding DescriptionInfo}"…/>
                    …
        </Grid>
    </Grid>
</DataTemplate>
```

Per fare in modo che il binding funzioni e che la view possa accedere ai dati del viewmodel, è necessario indicare a quale contesto dati fare riferimento assegnando al DataContext della view il viewmodel. Nel file SurfaceWindow1.xaml.cs si troveranno quindi le seguenti linee di codice:

Tabella 5.7 Impostazione DataContext

```
public partial class SurfaceWindow1 : SurfaceWindow
{
      LocationViewModel locationViewModel;
      public SurfaceWindow1()
        {
            InitializeComponent();
            AddActivationHandlers();

            locationViewModel = new LocationViewModel();
            this.DataContext = locationViewModel;
        }
}
```

In questo modo, ogni volta che si presenta un binding, WPF andrà a cercare nel DataContext la proprietà da "legare".
Ora per visualizzare i PushPin bisogna solo fornire i dati richiesti dalla view tramite binding e questo compito è riservato alla classe LocationViewModel.cs.

5.4 ViewModel

La classe LocationViewModel rappresenta il cuore dell'applicazione. Lo schema UML rappresentante la classe LocationViewModel è mostrato in Figura 5.4. Contiene:

- Una collezione di oggetti di tipo LocationModel per rappresentare i pushpin: Locations
- Una collezione di oggetti di tipo LocationModel per rappresentare le finestre informative: InfoWindows
- Una serie di variabili Are_PuhsPinsVisible per gestire lo stato di visibilità dei pushpin
- Una funzione InitData() di inizializzazione
- Le funzioni UpdatePushPinVisibiltiy() e IsAnyValidGroupVisible() per aggiornare i dati riguardanti la visibilità di ogni singolo pushpin.

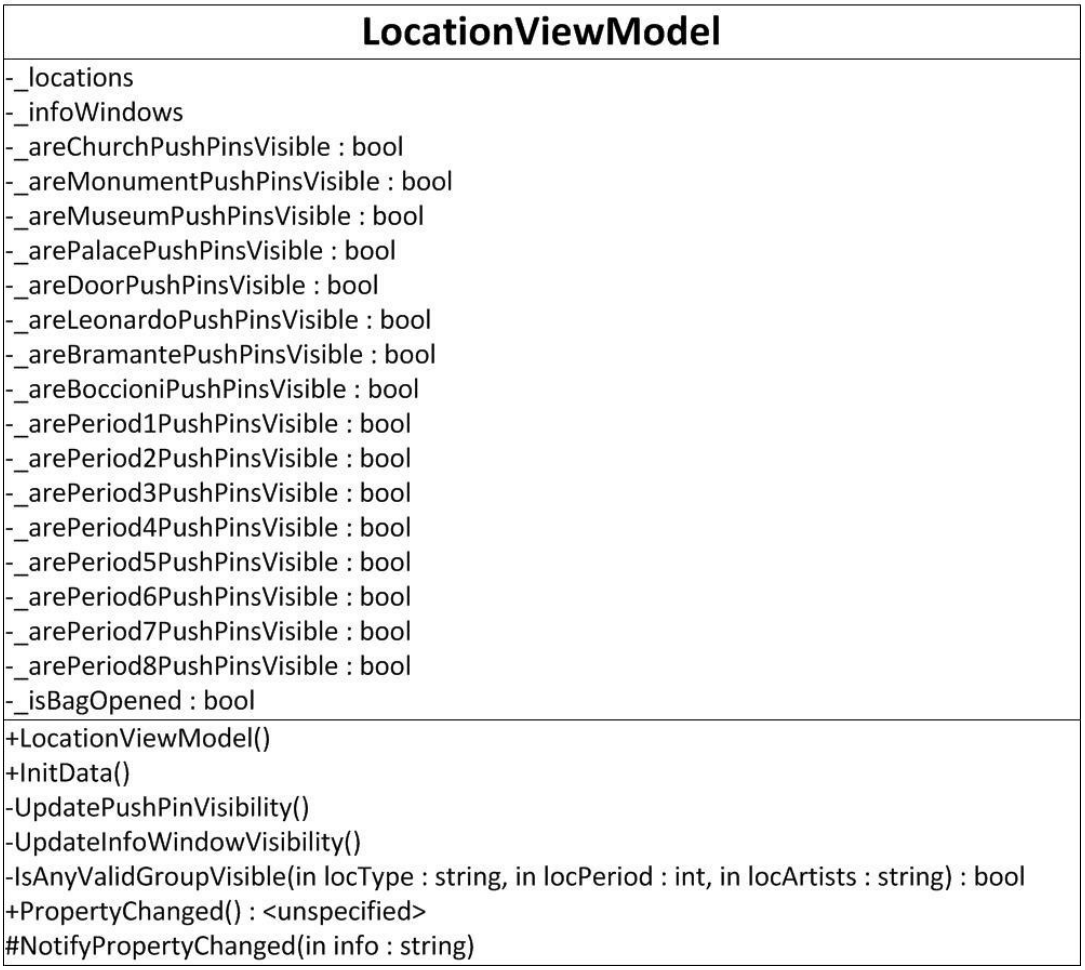

Figura 5.4 Rappresentazione della classe LocationViewModel in UML

La funzione InitData() si occupa di inizializzare e popolare il model con i dati prelevati dal file SitiArtisticiMilano.xml. Questi dati vengono inseriti in una collezione, Locations, di oggetti di tipo LocationModel (Tabella 5.8).

Tabella 5.8 Collezione Locations di oggetti di tipo LocationModel

```
public class LocationViewModel : INotifyPropertyChanged
{
    #region Properties

    private ObservableCollection<LocationModel> _locations;
    public ObservableCollection<LocationModel> Locations
    {
        get
        {
            return _locations;
        }
    }
    …
}
```

Come visto precedentemente in Tabella 5.2 Locations è legato all'attributo ItemsSource della mappa, e quindi rappresenta la fonte di dati da cui vengono prelevate le informazioni necessarie alla visualizzazione dei PushPin. Per prelevare dati dal file xml e popolare la collezione di LocationModel viene utilizzato LINQ to XML, un componente del .NET Framework che aggiunge capacità di query al linguaggio utilizzato. Un esempio di codice LINQ to XML applicato a DiscoverMilano è fornito in Tabella 5.9.

Tabella 5.9 Prelievo dati da XML

```
public void InitData()
    {
        //prelievo dati da xml
        ObservableCollection<LocationModel> newLocations = new
ObservableCollection<LocationModel>();
        XDocument loaded =
XDocument.Load(@"../../Resources/SitiArtisticiMilano.xml");
        var query = from p in loaded.Descendants("Location")
                select new
                {
                    Name = p.Element("Name").Value,
                    Latitude = p.Element("Latitude").Value,
                    Longitude = p.Element("Longitude").Value,
                    …
                    VideoSource = p.Element("VideoSource").Value
                };
//per ogni nodo "Location" trovato nel file, creo un'istanza di
LocationModel e la aggiungo a LocationViewModel.Locations
        foreach (var location in query)
        {
            LocationModel loc = new LocationModel( … );

            newLocations.Add(loc);
        }
        _locations = newLocations;
        NotifyPropertyChanged("Locations");
    }
```

La classe LocationViewModel si occupa di aggiornare lo stato dell'applicazione a seconda degli eventi che vengono rilevati nella view. Gli eventi vengono lanciati dai vari componenti della view, tra cui i vari SurfaceToggleButton che implementano i filtri dell'applicazione per la visibilità dei pushpin.

Come si può vedere dalla Figura 5.4 sono presenti numerose proprietà Are_PushPinsVisible nel ViewModel le quali, grazie al binding, vengono impostate a True o False a seconda che il SurfaceToggleButton relativo sia in stato selezionato (IsChecked) oppure no. In Tabella 5.10 si può vedere l'esempio del caso

AreChurchPushPinsVisible "legato" all'attributo IsChecked del tasto ChurchButton (come visto in Tabella 5.5).

Tabella 5.10 Aggiornamento stato della visibilità dei pushpin

```
private bool _areChurchPushPinsVisible;
    public bool AreChurchPushPinsVisible
    {
        get
        {
            return _areChurchPushPinsVisible;
        }
        set
        {
            _areChurchPushPinsVisible = value;
            NotifyPropertyChanged("AreChurchPushPinsVisible");
            UpdatePushPinVisibility();
        }
    }

#region Methods

    private void UpdatePushPinVisibility()
    {
        Locations.ToList().ForEach(loc => loc.IsPushPinVisible =
IsAnyValidGroupVisible(loc.Name));
    }
private bool IsAnyValidGroupVisible(string locationName)
    {
        switch (locationName)
        {
            case "CastelloSforzescoPushPin":
                return AreMuseumPushPinsVisible ||
AreLeonardoPushPinsVisible || ArePeriod1PushPinsVisible;

            ……

            case "YouAreHerePushPin":
                return true;
            default:
                return false;
        }
    }
    #endregion
```

Premendo il tasto Chiese per visualizzare tutte le chiese di Milano la proprietà IsChecked del controllo assume valore True e come conseguenza del binding rende il valore di AreChurchPushPinsVisible True. Una volta settato il valore, il setter chiama il metodo UpdatePushPinVisibility() che aggiorna il valore di IsPushPinVisible di ogni Location secondo lo stato della visibilità delle categorie (tipo di luogo, periodo artistico, artista) che lo riguardano. Il binding tra

IsPushPinVisible e PinVisible fa il resto e trasmette il nuovo valore di IsPushPinVisible a PinVisible e rende i pushpin delle chiese visibili.

Premendo il pulsante Itinerari, cambia la visuale dell'interfaccia. I pulsanti ai bordi vengono disabilitati e viene mostrata l'interfaccia per controllare e modificare le tappe dell'itinerario scelto. Il meccanismo è abbastanza semplice: si lavora sulla visibilità dei componenti a seconda dello stato della proprietà IsBagOpen, che indica se la cartella delle destinazioni selezionate è aperta oppure no. In questo modo il controllo che contiene la mappa (chiamato MapContainer) sparisce insieme al tasto Itinerario (BagButton) quando questo viene premuto, lasciando spazio al controllo LibraryBar, che contiene le destinazioni selezionate (SelectedDestinationsLibraryBar), e ai tasti per tornare alla schermata principale e per stampare.

Tabella 5.11 Passaggio da mappa a itinerari selezionati e viceversa

```
//per andare alla schermata delle destinazioni selezionate ed
eventualmente rimuoverne alcune e stampare
private void BagButton_Click(object sender, RoutedEventArgs e)
        {
            if (!locationViewModel.IsBagOpen)
            {
                MapContainer.Visibility = Visibility.Collapsed;
                BagButton.Visibility = Visibility.Collapsed;
                MapButton1.Visibility = Visibility.Visible;
                MapButton2.Visibility = Visibility.Visible;
                PrintButton1.Visibility = Visibility.Visible;
                PrintButton2.Visibility = Visibility.Visible;
                SelectedDestinationsLibraryBar.Visibility =
Visibility.Visible;
                locationViewModel.InfoWindows.Clear();

                locationViewModel.IsBagOpen = true;
             }
//per tornare alla schermata della mappa e aggiungere altri
itinerari
private void MapButton_Click(object sender, RoutedEventArgs e)
        {
            if (locationViewModel.IsBagOpen)
            {
                MapContainer.Visibility = Visibility.Visible;
                BagButton.Visibility = Visibility.Visible;
                MapButton1.Visibility = Visibility.Collapsed;
                MapButton2.Visibility = Visibility.Collapsed;
                PrintButton1.Visibility = Visibility.Collapsed;
                PrintButton2.Visibility = Visibility.Collapsed;
                SelectedDestinationsLibraryBar.Visibility =
Visibility.Collapsed;
```

```
                locationViewModel.IsBagOpen = false;
            }
        }
```

5.5 Strutture dati

Per lo sviluppo del presente progetto i dati concernenti i beni culturali sono stati strutturati in un file XML aggiunto come risorsa di progetto tramite Visual Studio. Il file XML è stato preferito a un database vero e proprio per la maggiore semplicità d'interfacciamento con l'applicazione e per la natura prototipale del progetto in cui una quantità contenuta di dati viene letta una sola volta all'inizio dell'applicazione e il file non subisce altri accessi.

Il file si compone di vari nodi `<Location>` contenenti tutti i dati riguardanti il luogo d'arte che possono essere sfruttati o no dall'applicazione. Di seguito, in Tabella 5.12, viene presentata la struttura di un nodo, mentre in Appendice A è mostrato un esempio di nodo completo di dati:

Tabella 5.12 Struttura dati DiscoverMilano

```
<Location>
    <Name></Name>
    <Latitude></Latitude>
    <Longitude></Longitude>
    <Address></Address>
    <OpeningHours></OpeningHours>
    <Prices></Prices>
    <Contacts></Contacts>
    <Description></Description>
    <Curiosities></Curiosities>
    <MustSee></MustSee>
    <UsefulDetails></UsefulDetails>
    <ImagesFolderPath></ImagesFolderPath>
    <VideoFilePath></VideoFilePath>
    <Type></Type>
    <Period></Period>
    <IsLeonardo></IsLeonardo>
    <IsBramante></IsBramante>
    <IsBoccioni></IsBoccioni>
    <Importance></Importance>
    <Artists></Artists>
  </Location>
```

Questi dati vengono prelevati tramite LINQ to XML come visto in Tabella 5.9 per popolare il modello dei dati di DiscoverMilano.

CAPITOLO 6:
PORTING SU ALTRI DISPOSITIVI

Durante lo sviluppo dell'applicativo Surface si è resa disponibile la possibilità di utilizzare un dispositivo tablet multi-touch su cui poter provare a sviluppare DiscoverMilano. In questo capitolo vengono mostrate le motivazioni per cui si è deciso di intraprendere un cammino di porting dell'applicazione da una piattaforma all'altra, seguite da alcune considerazioni riguardo alla portabilità di DiscoverMilano e dei sistemi basati su WPF in generale. Vengono poi mostrati i vincoli che sono stati introdotti con l'operazione di porting. Infine vengono presentati gli aspetti implementativi legati al porting.

6.1 Motivazioni

Il lavoro di porting è stato effettuato non con l'obiettivo di creare un'applicazione che si adatti alle caratteristiche di un tablet, diverse da quelle di un tabletop computer come Microsoft Surface, ma a scopo dimostrativo e per poter creare un'esperienza tattile. Durante il lavoro d'implementazione la presenza di un simulatore si è resa utile per poter originare eventi legati al contatto delle dita e non al click del mouse e ciò è stato fondamentale per progettare un'applicazione per Surface. La necessità però di utilizzare mouse e trackpad per simulare il tocco delle dita delle mani rende l'interazione poco naturale e limitata. Inoltre la disponibilità del tablet si è rivelata un'ottima occasione per studiare la portabilità di applicazioni Windows tra diversi dispositivi.

DiscoverMilano è basato sul .NET Framework di Microsoft e in quanto tale teoricamente può essere facilmente adattato a qualsiasi piattaforma che supporti questa tecnologia. La natura dell'interazione di DiscoverMilano è tattile, ed è progettato per dispositivi e sistemi operativi che supportino il tocco. Ciò rende naturale la possibilità di portare DiscoverMilano su altri dispositivi touch basati su sistema operativo Microsoft 7 e la disponibilità di un tablet basato su tale sistema operativo ha dato la possibilità di poter sperimentare il porting di un'applicazione Surface su Windows 7.

6.2 Vincoli

Come si può vedere dalla Figura 6.1, l'applicazione per Windows 7 presenta alcune differenze rispetto alla versione originale.

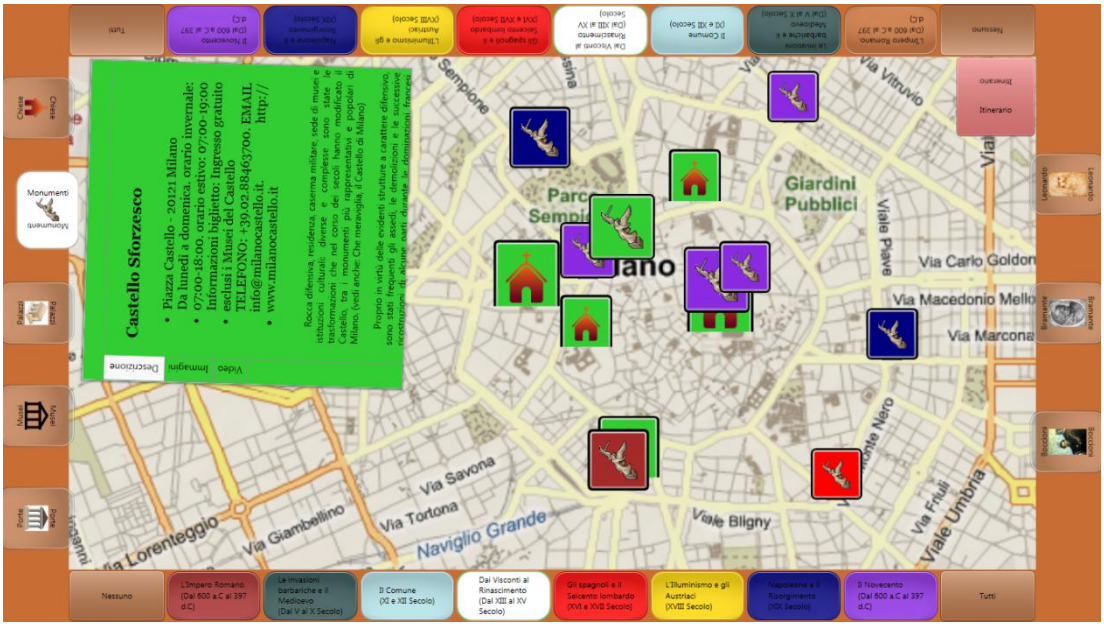

Figura 6.1 Screenshot DiscoverMilano su Windows 7

Le proporzioni e le dimensioni dello schermo sono diverse: si passa da un display 30" in formato 4:3 di Surface (o circa 15" del simulatore) con risoluzione di 1024 x 768 pixel, a uno schermo da 11,6" in formato 16:9 e con risoluzione 1366 x 768 del tablet che risulta dunque in uno schermo più piccolo e allungato. I pushpin inoltre, forse per qualche problema di incompatibilità, vengono renderizzati senza la base a forma di triangolo che indica con precisione la posizione del luogo indicato.

I pushpin non sono gli unici elementi a soffrire di problemi a causa del porting. Il controllo LibraryBar utilizzato per visualizzare l'itinerario scelto, viene istanziato con dimensioni ridotte che non si riescono a cambiare. In Figura 6.2 è possibile vedere come il controllo non riesca a occupare tutta la larghezza dello spazio come faceva prima del porting (Figura 4.4).

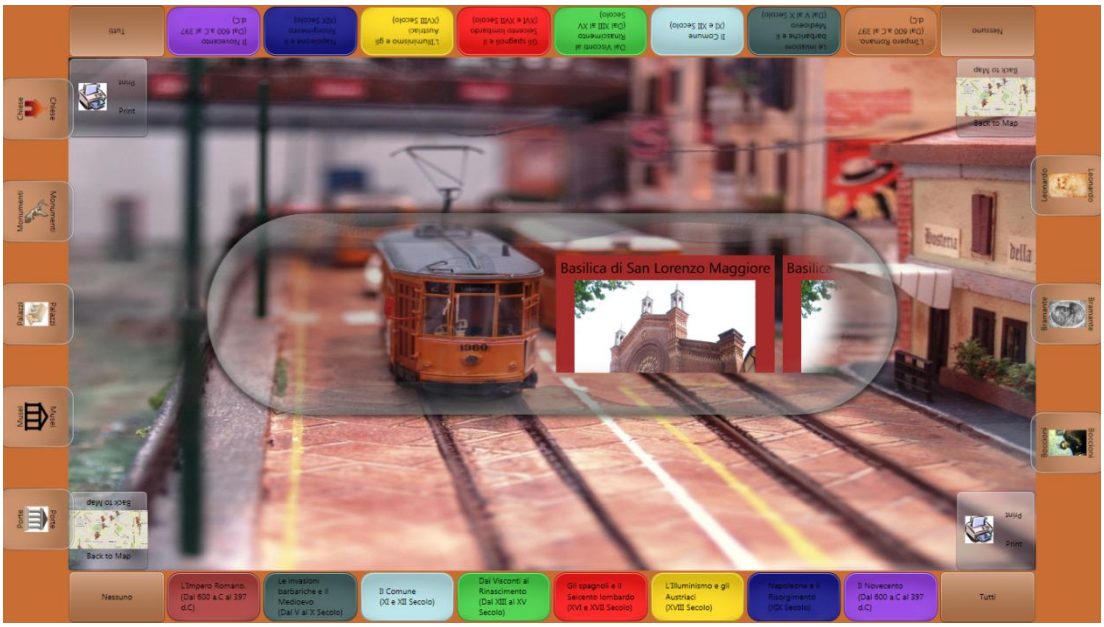

Figura 6.2 Screenshot schermata itinerario

Un'altra differenza sostanziale è che il tablet è sì multi-touch ma riconosce un numero limitato di contatti con cui non è possibile un utilizzo multiutente. Sparisce così la possibilità di interazione tra utenti e di conseguenza l'utilità di un'interfaccia a 360°, requisiti fondamentali nel caso della versione Surface.

6.3 Aspetti implementativi

Portare un'applicazione Surface su Windows 7 è stato in generale relativamente semplice, ma sono comunque stati necessari alcuni interventi sul codice e sull'ambiente di sviluppo e nonostante ciò, si sono presentati alcuni problemi nella visualizzazione dei controlli. L'ambiente di sviluppo utilizzato è lo stesso dei capitoli precedenti, ma con Visual Studio 2008 e Surface SDK installati in questo caso sopra il sistema operativo Windows 7. Per poter utilizzare appieno tutte le funzionalità presenti su Surface è stato necessario aggiungere alcune librerie al progetto tra le quali Microsoft.Surface.dll e Windows7.Multitouch.dll. Inoltre essendo l'utilizzo delle mappe basato su un pacchetto esterno pensato per girare su Surface è stato

necessario cambiarlo con la versione per Windows 7 fornita sempre da InfoStrat, modificando di conseguenza ogni elemento del pacchetto SurfaceVEMap con il corrispettivo di Win7TouchVEMap. Infine è stato necessario modificare i namespace per adattarli al nuovo progetto e alle nuove librerie.
Il processo di porting, non ha richiesto grosse modifiche al codice e all'ambiente di lavoro, ma non è possibile utilizzare lo stesso identico codice per due piattaforme diverse come Surface e Windows 7, anche per questioni di dimensioni e caratteristiche diverse dei dispositivi e si rende perciò necessaria una rivalutazione dei requisiti della UX.

CAPITOLO 7: DISCUSSIONE

Il pacchetto offerto da Microsoft per lo sviluppo di applicazioni dal contenuto ricco e di grande impatto per la user experience si è rivelato un buon insieme di strumenti. WPF soprattutto si è rivelato uno strumento facile da imparare e allo stesso tempo molto potente, in grado di semplificare notevolmente il lavoro dello sviluppatore attento al design della user experience. Il Software Development Kit di Microsoft Surface include un simulatore che permette di sviluppare applicativi per Microsoft Surface sulla propria workstation Windows che se da un lato permette lo sviluppo di applicazioni Surface sul proprio PC, presenta anche difetti importanti, mostrati nel sottoparagrafo 2.3.1, tra cui spicca l'impossibilità di ridimensionare il simulatore per ottenere una superficie di lavoro di dimensioni che si avvicinino di più al dispositivo simulato. A causa di ciò sono state incontrate alcune difficoltà nella progettazione dell'interfaccia soprattutto per quanto riguarda l'assegnazione di valori assoluti come la grandezza minima dei caratteri e delle finestre di dialogo. Nonostante sia possibile fare affidamento su grandezze relative nella maggior parte dei casi, ci sono situazioni in cui l'assegnazione di una dimensione minima assoluta è fondamentale per l'usabilità del prodotto. Per esempio i caratteri testuali devono avere una dimensione di almeno 12 pixel perché possano essere leggibili su Surface [24] e le finestre informative devono avere una dimensione minima accettabile per poter mostrare il loro contenuto adeguatamente. Impostando i valori minimi necessari per una visualizzazione corretta sullo schermo da 30" di Surface ciò che si ottiene sul simulatore dalle dimensioni pressoché dimezzate è un'applicazione con alcuni elementi sproporzionatamente più grandi in relazione al resto dell'interfaccia.

Essendo ancora una tecnologia relativamente recente che non ha avuto ancora un grande successo commerciale, non ci sono molte fonti da cui trarre informazioni anche in rete, e spesso molte domande rimangono senza risposta. A supporto dello sviluppatore, su MSDN Library è possibile trovare una documentazione abbastanza completa sull'SDK. Gli esempi e i campioni di codice presenti sono però ancora pochi e riguardano solo le funzionalità più pubblicizzate. Non esistono inoltre soluzioni ufficiali per problemi comuni come l'integrazione delle Bing Maps su Surface, se non tramite controlli esterni o pacchetti chiusi non modificabili come Microsoft Surface Globe incluso nel Touch Pack per Windows 7, che non possono essere utili nello sviluppo di applicazioni proprie. Affidandosi a soluzioni esterne è inevitabile incontrare errori, limitazioni e problemi nell'utilizzo, come nel caso dei pushpin nella versione tablet. Il problema della poca documentazione disponibile è stato riscontrato anche durante il processo di porting da Surface a Windows 7 e in generale può essere considerato il fattore che ha creato maggiori difficoltà nello svolgimento del progetto.

CAPITOLO 8: CONCLUSIONI

L'interazione tabletop, combinata con il multi-touch, offre grandi potenzialità per quanto concerne la User Experience. Grazie alla sua posizione orizzontale e alla possibilità di implementare interfacce a 360°, la configurazione tabletop si presta molto bene a un utilizzo collaborativo multi-utente, elemento di grande influenza per l'experience [25] . L'interazione multi-touch offre semplicità e divertimento di utilizzo e senso di controllo dell'applicazione. I limiti risiedono soprattutto nella giovinezza di questa tecnologia, nell'ancora poca diffusione della comunità di sviluppatori e nell'assenza di uno standard nello sviluppo di applicazioni multi-touch dovuto soprattutto al fatto che si applica ad ambiti molto diversi tra loro e che spesso anche all'interno di uno stesso settore ogni produttore utilizza tecnologie proprietarie per lo sviluppo.

Nell'ambito del turismo sono state individuate diverse applicazioni di tecnologie multi-touch già sviluppate. In tutte, l'utilizzo di tali tecnologie non va a sostituire i vecchi strumenti utilizzati in precedenza, ma ad arricchire l'esperienza dell'utente con uno strumento di grande impatto. Questo a conferma del fatto che il multi-touch è un'evoluzione tecnologica il cui valore aggiunto non è permettere di fare cose vecchie in modo nuovo, ma è un elemento aggiuntivo che permette agli utenti di interagire con i contenuti in una nuova e unica maniera.

Per questo motivo può essere un utile strumento per rilanciare e promuovere i beni artistici e culturali di una città, può essere un valore in più che si va ad aggiungere senza sconvolgere un sistema d'incentivazione e promozione del turismo già esistente.

Nel caso specifico di Microsoft Surface, questo gode di tutti i vantaggi dei dispositivi multi-touch e tabletop sopracitati ma soffre anche dei difetti quali l'utilizzo di un ambiente chiuso e proprietario di sviluppo e di conseguenza la limitatezza di una comunità di sviluppatori. Un grosso vantaggio invece è la presenza nel cuore di Surface di un computer completo di sistema operativo Microsoft Windows Vista che mette a disposizione degli sviluppatori tutte le comodità di un sistema operativo completo, tra cui la possibilità di collegare una stampante, come nel caso analizzato, o altri dispositivi a una delle porte USB 2.0 a disposizione e la presenza di numerosi driver. Inoltre la presenza di connessioni Bluetooth 2.0 e WiFi permette collegamenti senza fili con altri dispositivi come gli smartphone o i nuovi tablet come l'iPad di Apple. Infine la vasta diffusione di prodotti Microsoft soprattutto tra PC e smartphone con Windows 7 e Windows Mobile (e ora il nuovo Windows Phone 7), unita alla buona portabilità tra tali sistemi, permette di adattare l'applicazione a diversi tipi di dispositivi.

Al momento DiscoverMilano è un prototipo da validare e un primo sviluppo futuro potrebbe essere proprio quello di validazione tramite user testing.

Ulteriori sviluppi futuri si possono dividere in due macro categorie:

- Sviluppo multicanalità
- Estensione dei contenuti e dei servizi

Nel primo caso, la diffusione di smartphone multitouch e l'uscita del nuovo sistema operativo mobile Windows Phone 7, uniti alla portabilità tra sistemi Microsoft, rendono interessante il porting su un dispositivo mobile basato su tale sistema operativo, eventualmente combinato con l'integrazione tra l'applicativo mobile e quello Surface. Per la versione mobile si ritiene utile integrare anche contenuti audio in modo da fornire i visitatori di un'utile guida da ascoltare durante la visita.

Nel secondo caso, un possibile sviluppo futuro potrebbe essere l'integrazione con i servizi, se disponibili, dell'azienda di trasporti cittadina (nel caso di Milano, l'ATM) in modo da poter offrire un itinerario non solo automobilistico e pedonale offerto dai servizi di mappe online attuali, ma anche indicazioni sui mezzi di trasporto da utilizzare per spostarsi in città. Questo andrebbe a coprire un'altra fetta di bisogni che appartiene ai visitatori, spesso non automuniti.

Riferimenti bibliografici

[1] Betts, P., Brown, C.J., Lynott, J.J., Martin, H.F., "Light Beam Matrix Input Terminal", IBM Technical Disclosure Journal, 9(5), 1965, pp. 493-494.

[2] Boie, R.A., "Capacitive Impedance Readout Tactile Image Sensor", Proceedings of the IEEE International Conference on Robotics, 1984, pp. 370-378.

[3] Buxton, B., "31.1: Invited Paper: A Touching Story: A Personal Perspective on the History of Touch Interfaces Past and Future", Microsoft Research, One Microsoft Way, Redmond, WA, USA, Society for Information Display (SID) Symposium Digest of Technical Papers, Maggio 2010, volume 41(1), session 31, pp. 444-448.

[4] Buxton, B., "Multi-Touch Systems that I Have Known and Loved", Microsoft Research, http://www.billbuxton.com/multitouchOverview.html, 9 Ottobre 2009. Data visita: Marzo 2010.

[5] Buxton, W., Fitzmaurice, G., Balakrishnana, R., Kurtenbach, G., "Large Displays in Automotive Design", IEEE Computer Graphics and Applications, 20(4), 2000, pp. 68-75.

[6] Colegrove, J., "The State of the Touch-Screen Market in 2010", in Information Display, Vol. 26, No. 3, Marzo 2010, pp. 22-24.

[7] Colegrove, J., Hsieh, C., "Touch Panel Market Analysis", DisplaySearch, http://www.displaysearch.com/cps/rde/xchg/displaysearch/hs.xsl/touch_panel_market_analysis.asp. Data visita: Gennaio 2010

[8] De Carlo, M., "Competitività della destinazione Milano – Secondo rapporto, Marzo 2007", Camera di Commercio di Milano, Fondazione IULM, Marzo 2007.

[9] Dietz, P. and Leigh, D., "DiamondTouch: a multi-user touch technology", in Proceedings of the 14th annual ACM symposium on User interface software and technology (UIST '01), Orlando, Florida, 2001, ACM Press, pp. 219-226.

[10] GestureTek, “Illuminate Series – Business Cases”, Illuminate MultiTouch Display Screens, Tables and Windows/Interactive Projection System and Surface Computing Technology, http://www.gesturetek.com/illuminate/businesscases.php. Data visita: Marzo 2010

[11] GestureTek, “New York City Visitor’s Center”, Illuminate MultiTouch Display Screens, Tables and Windows/Interactive Projection System and Surface Computing Technology http://www.gesturetek.com/illuminate/businesscases/nyc-visitors-center.php. Data visita: Marzo 2010

[12] Gossman, J., “Introduction to Model/View/ViewModel pattern for building WPF apps”, MSDN Blogs – Tales from the Smart Client, http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx, 08 Ottobre 2005. Data ultima visita: Settembre 2010.

[13] Information Strategies, “InfoStrat.VE - Bing Maps 3D for WPF and Microsoft Surface”, http://bingmapswpf.codeplex.com/. Data ultima visita: Ottobre 2010.

[14] Lewis, J.R., “Reaping the benefits of modern usability evaluation: The Simon story”, Advances in Applied Ergonomics: Proceedings of the 1st International Conference on Applied Ergonomics (ICAE ’96), 1996, pp. 752-757.

[15] M.W. Krueger, “Artificial Reality”, Addison-Wesley, Reading, MA, 1983.

[16] M.W. Krueger, T. Gionfriddo, K. Hinrichsen, “VIDEOPLACE - An Artificial Reality,” Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI’85), 1985, pp. 35 – 40.

[17] Malik, S., “An Exploration of Multi-Finger Interaction on Multi-Touch Surfaces”, University of Toronto, 2007.

[18] Marshall, P., Hornecker, E., Morris, R., Dalton, S. and Rogers, Y., “When the fingers do the talking: A study of group participation for different kinds of shareable surfaces”, in Preceedings of IEEE Tabletops and Interactive Surfaces ’08, 1-3 Oct 2008, Amsterdam, Netherlands.

[19] Metha, N., “A Flexible Machine Interface”, MSc. Thesis, Department of Electrical Engineering, University of Toronto, 1982.

[20] Microsoft Corporation, “AT&T First to Introduce Microsoft Surface in Retail Stores to Enhance Mobile Shopping Experience”, Microsoft News Center, http://www.microsoft.com/presspass/press/2008/apr08/04-01surfaceretailpr.mspx, 1 Aprile 2008. Data visita: Ottobre 2009.

[21] Microsoft Corporation, “Developing Microsoft Surface Applications on a Separate Workstation”, MSDN Library, http://msdn.microsoft.com/en-us/library/ee804897(Surface.10).aspx. Data ultima visita: Novembre 2010.

[22] Microsoft Corporation, “Experience Microsoft Surface”, Data sheet.

[23] Microsoft Corporation, “Look What’s Surfacing at Microsoft”, Microsoft News Center, http://www.microsoft.com/presspass/features/2007/may07/05-29Surface.mspx, 29 Maggio 2007. Data visita: Ottobre 2009.

[24] Microsoft Surface TM, “User Experience Guidelines – User Interaction and Design Guidelines for Creating Microsoft Surface Applications”, 2 Giugno 2009.

[25] Morris, M.R., Huang, A., Paepcke, A. and Winograd, T. (2006) “Cooperative gestures: Multi-user gestural interactions for co-located groupware.” Proc. CHI '06, New York, 2006, ACM Press, pp. 1201-1210.

[26] Moscovich, T., “Principles and Applications of Multi-touch Interaction”, Brown University, 2007.

[27] Natural User Interface Group, “Natural User Interfaces”, http://wiki.nuigroup.com/Natural_User_Interface. Data visita: Dicembre 2009.

[28] Osservatorio del Turismo di Milano, “I flussi turistici a Milano e Provincia. Prime evidenze anno 2009”, Milano, 2009.

[29] Peltonen, P., Kurvinen, E., Salovaara, A., Jacucci, G.,Ilmonen, T., Evans, J., Oulasvirta, A., and Saarikko, P., “It's Mine, Don't Touch!: interactions at a large multi-touch display in a city centre”, in Proceedings of CHI’08, Firenze, 2008, pp. 1285-1294.

[30] Provincia di Milano, “Luoghi da vivere – Sistema Turistico Metropolitano”, Documento Strategico e Programma Sviluppo Turistico 2009, Milano, 2009.

[31] Rekimoto, J., “SmartSkin: an infrastructure for freehand manipulation on interactive surfaces”, in Proceedings of the SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves, Minneapolis, Minnesota, USA, 2002, ACM Press, pp. 113-120.

[32] Rosson, M.B. e Carroll, J.M., “Scenario-Based Design”, in Jacko, J.A. e Sears, A., The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications, Lawrence Erlbaum Associates, Inc., Mahwah, NJ, 2002.

[33] Smith, J., “WPF Apps With The Model-View-ViewModel Design Pattern”, MSDN Magazine, http://msdn.microsoft.com/en-us/magazine/dd419663.aspx, Febbraio 2009. Data ultima visita: Ottobre 2010.

[34] Strukt GmbH, “Schlossmuseum Linz Multitouch Installation/Strukt”, http://strukt.com/2010/schlossmuseum-multitouch/, 2010. Data Visita: Marzo 2010.

[35] The Open University, “OU interactive tabletop computer helps tourists explore Cambridge”, http://www3.open.ac.uk/media/fullstory.aspx?id=18770, Cambridge. Data visita: Maggio 2010.

[36] Wellner, P., “The DigitalDesk calculator: tangible manipulation on a desk top display”, in Proceedings of the 4th annual ACM symposium on User interface software and technology, Hilton Head, South Carolina, USA, 1991.

[37] Wu, M. and Balakrishnan, R., “Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays” in Proceedings of the 16th annual ACM symposium on User interface software and technology (UIST '03), Vancouver, Canada, 2003, pp. 193-202.

APPENDICE A:
CODICE DISCOVERMILANO

Per migliorare la leggibilità del documento di tesi, la maggior parte del codice dell'applicazione è stato trasferito in appendice. Di seguito si trovano vari estratti di codice suddivisi per funzione (Model, View o ViewModel), come nel Capitolo 5.

A.1 Codice Model

Tabella 8.1 Estratto della classe LocationModel.cs

```
public class LocationModel : INotifyPropertyChanged
{
    #region Properties

    private string _name;
    public string Name
    {
        get
        {
            return _name;
        }
        set
        {
            _name = value;
            NotifyPropertyChanged("Name");
        }
    }
    private double _latitude;
    public double Latitude
    { get …  set …  }
    private double _longitude;
    public double Longitude
    { get …  set …  }
    private string _type;
    public string Type
```

```
    { get …  set …  }
    private int _period;
    public int Period
    { get …  set …  }
    private bool _isLeonardo;
    public bool IsLeonardo
    { get …  set …  }
    private bool _isBramante;
    public bool IsBramante
    { get …  set …  }
    private bool _isBoccioni;
    public bool IsBoccioni
    { get …  set …  }
    private string _relatedArtists;
    public string RelatedArtists
    { get …  set …  }
    private Int32 _importance;
    public Int32 Importance
    { get …  set …  }
    private string _addressInfo;
    public string AddressInfo
    { get …  set …  }
    private string _openingHoursInfo;
    public string OpeningHoursInfo
    { get …  set …  }
    private string _contactsInfo;
    public string ContactsInfo
    { get …  set …  }
    private string _priceInfo;
    public string PriceInfo
    { get …  set …  }
    private string _descriptionInfo;
    public string DescriptionInfo
    { get …  set …  }
    private string _imagesFolderPath;
    public string ImagesFolderPath
    { get …  set …  }
    private string _videoFilePath;
    public string VideoFilePath
    { get …  set …  }
    private bool _isPushPinVisible;
    public bool IsPushPinVisible
    {
        get
        {
            return _isPushPinVisible;
        }
        set
        {
            _isPushPinVisible = value;
            NotifyPropertyChanged("IsPushPinVisible");
        }
    }
    #endregion
//Costruttore
    public LocationModel( string name, double latitude, double
```

```
    longitude, string type, int period, bool isLeonardo, bool
    isBramante, bool isBoccioni, string importance, string address,
    string openinghours, string contactsInfo, string description,
    string imagesListURI, string videoURI)
    {
        this.Name = name;
        this.Latitude = latitude;
        this.Longitude = longitude;
        …
        this.IsPushPinVisible = false;
    }
```

A.2 Codice View

Tabella 8.2 Estratto della classe SurfaceWindow1.xaml

```
<s:SurfaceWindow x:Class="BeniCulturaliMilano3.SurfaceWindow1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:s="http://schemas.microsoft.com/surface/2008"
xmlns:ve="clr-namespace:InfoStrat.VE.NUI;assembly=InfoStrat.VE.NUI"
xmlns:converter="clr-namespace:BeniCulturaliMilano3.Converters"
xmlns:local="clr-namespace:BeniCulturaliMilano3"
Title="BeniCulturaliMilano3">
  <s:SurfaceWindow.Resources>
    ……
  </s:SurfaceWindow.Resources>
    <Grid Background="{StaticResource WindowBackground}" >
        <Grid.RowDefinitions>
            <RowDefinition Height="auto"/>
            <RowDefinition />
            <RowDefinition Height="auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="auto"/>
            <ColumnDefinition />
            <ColumnDefinition Width="auto"/>
        </Grid.ColumnDefinitions>

        <!--Mappa e contenuto centrale (scatterviewitems)-->
        <Grid x:Name="Grid11" Grid.Row="1" Grid.Column="1" …>
            …
           <Viewbox x:Name="MapContainer" Stretch="UniformToFill">
                <ve:SurfaceVEMap Name="map" LatLong="…"
                                 Altitude="7000" MapStyle="Road"
                                 ItemsSource="{Binding Locations}"
                              ItemTemplateSelector="{StaticResource
templateSelector}"/>
            </Viewbox>
            <!—Layer delle finestre informative-->
            <s:ScatterView x:Name="InfoWindowsSV"
                           ItemsSource="{Binding InfoWindows}"
                           ItemTemplate="{StaticResource
InfowindowTemplate}">
                …
```

```
            </s:ScatterView>
            <!—Pulsanti per passare da una schermata all’altra e
per stampare-->
            <s:SurfaceButton x:Name="BagButton" Grid.Row="1"
Grid.Column="1" …>…</s:SurfaceButton>
            <s:SurfaceButton x:Name="MapButton2" Grid.Row="1"
Grid.Column="1" …>…</s:SurfaceButton>
            <s:SurfaceButton x:Name="MapButton1" Grid.Row="1"
Grid.Column="1" …>…</s:SurfaceButton>
            <s:SurfaceButton x:Name="PrintButton1" Grid.Column="1"
Grid.Row="1" …>…</s:SurfaceButton>
            <s:SurfaceButton x:Name="PrintButton2" Grid.Column="1"
Grid.Row="1" …>…</s:SurfaceButton>
            <s:LibraryBar x:Name="SelectedDestinationsLibraryBar"…>
                …
            </s:LibraryBar>

        </Grid>
        <!--Barra temporale nord-->
        <Grid Grid.Row="0" Grid.Column="1" MaxHeight="100">
            <Grid Name="Filtro_Temporale_Nord"
RenderTransformOrigin="0.5,0.5">
                <Grid.RenderTransform>
                    <RotateTransform Angle="180"/>
                </Grid.RenderTransform>
                …
                <s:SurfaceButton x:Name="NoFilterButton_Nord" …
Click="NoFilterButton_Click" >…</s:SurfaceButton>
                <s:SurfaceToggleButton x:Name="Period1Button_Nord"
Style="{StaticResource FiltersStyle}" Grid.Column="1"
IsChecked="{Binding ArePeriod1PushPinsVisible}"
Background="{Binding PeriodColors.Period1Color}">
…</s:SurfaceToggleButton>
                …
            </Grid>
        </Grid>
        <!--Barra Temporale Sud-->
        uguale a Nord girato di 180°

        <!--Filtri Tipologia di Luogo-->
        <Grid Name="Filtro_Tipologia_Luogo" Grid.Row="1"
Grid.Column="0" >
            …
            <s:SurfaceToggleButton x:Name="ChurchButton"
Grid.Row="0" Style="{StaticResource ArtTypeButtonStyle}"
IsChecked="{Binding AreChurchPushPinsVisible}" >
                …
            </s:SurfaceToggleButton>
            …
        </Grid>
        <!--Filtri per Artista-->
        <Grid Name="Filtro_Artista" Grid.Row="1" Grid.Column="2">
            …
            <s:SurfaceToggleButton x:Name="LeonardoButton"
Grid.Row="0" Style="{StaticResource ArtistButtonStyle}"
IsChecked="{Binding AreLeonardoPushPinsVisible}">
```

```
                …
            </s:SurfaceToggleButton>
            …
        </Grid>
    </Grid>
</s:SurfaceWindow>
```

A.3 Codice ViewModel

Tabella 8.3 Estratto della classe LocationViewModel.cs

```
public class LocationViewModel : INotifyPropertyChanged
    {
        #region Properties

        private ObservableCollection<LocationModel> _locations;
        public ObservableCollection<LocationModel> Locations
        {
            get
            {
                return _locations;
            }
        }

        private ObservableCollection<LocationModel> _infoWindows;
        public ObservableCollection<LocationModel> InfoWindows
        {
            get
            {
                return _infoWindows;
            }
            set
            {
                _infoWindows = value;
                NotifyPropertyChanged("InfoWindows");
                UpdateInfoWindowVisibility();
            }
        }


        private bool _areChurchPushPinsVisible;
        public bool AreChurchPushPinsVisible
        {
            get
            {
                return _areChurchPushPinsVisible;
            }
            set
            {
                _areChurchPushPinsVisible = value;
                NotifyPropertyChanged("AreChurchPushPinsVisible");
                UpdatePushPinVisibility();
            }
        }
```

```
        private bool _areMonumentPushPinsVisible;
        public bool AreMonumentPushPinsVisible
        { get …  set …  }
        private bool _areMuseumPushPinsVisible;
        public bool AreMuseumPushPinsVisible
        { get …  set …  }
        private bool _arePalacePushPinsVisible;
        public bool ArePalacePushPinsVisible
        { get …  set …  }
        private bool _areDoorPushPinsVisible;
        public bool AreDoorPushPinsVisible
        { get …  set …  }
        private bool _areLeonardoPushPinsVisible;
        public bool AreLeonardoPushPinsVisible
        { get …  set …  }
        private bool _areBramantePushPinsVisible;
        public bool AreBramantePushPinsVisible
        { get …  set …  }
        private bool _areBoccioniPushPinsVisible;
        public bool AreBoccioniPushPinsVisible
        { get …  set …  }
        private bool _arePeriod1PushPinsVisible;
        public bool ArePeriod1PushPinsVisible
        { get …  set …  }
        .
        .
        .
        private bool _arePeriod8PushPinsVisible;
        public bool ArePeriod8PushPinsVisible
        { get …  set …  }

        private bool _isBagOpen;
        public bool IsBagOpen
        {
            get
            {
                return _isBagOpen;
            }
            set
            {
                _isBagOpen = value;
                NotifyPropertyChanged("IsBagOpen");
            }
        }
        #endregion

        #region INotifyPropertyChanged Members

        public event PropertyChangedEventHandler PropertyChanged;
        protected void NotifyPropertyChanged(String info)
        {
            if (PropertyChanged == null)
                return;
            PropertyChanged(this, new
PropertyChangedEventArgs(info));
```

```
        }
        #endregion

        #region Constructors

        public LocationViewModel()
        {
            _areBoccioniPushPinsVisible = false;
            _areBramantePushPinsVisible = false;
            _areChurchPushPinsVisible = false;
            _areDoorPushPinsVisible = false;
            _areLeonardoPushPinsVisible = false;
            _areMonumentPushPinsVisible = false;
            _areMuseumPushPinsVisible = false;
            _arePalacePushPinsVisible = false;
            _arePeriod1PushPinsVisible = false;
            _arePeriod2PushPinsVisible = false;
            _arePeriod3PushPinsVisible = false;
            _arePeriod4PushPinsVisible = false;
            _arePeriod5PushPinsVisible = false;
            _arePeriod6PushPinsVisible = false;
            _arePeriod7PushPinsVisible = false;
            _arePeriod8PushPinsVisible = false;
            _isBagOpen = false;

            InitData();
            UpdatePushPinVisibility();
        }

         public void InitData()
        {   //prelievo dati da xml
            ObservableCollection<LocationModel> newLocations= new
ObservableCollection<LocationModel>();
            XDocument loaded =
XDocument.Load(@"../../Resources/SitiArtisticiMilano.xml");
            var query = from p in loaded.Descendants("Location")
                        select new
                        {
                            Name = p.Element("Name").Value,
                            Latitude = p.Element("Latitude").Value,
                            Longitude =
p.Element("Longitude").Value,
                            Type = p.Element("Type").Value,
                            Period = p.Element("Period").Value,
                            IsLeonardo =
p.Element("IsLeonardo").Value,
                            IsBramante =
p.Element("IsBramante").Value,
                            IsBoccioni =
p.Element("IsBoccioni").Value,
                            Artists = p.Element("Artists").Value,
                            Importance =
p.Element("Importance").Value,
                            AddressInfo =
p.Element("Address").Value,
                            Prices = p.Element("Prices").Value,
```

```
                                OpeningHoursInfo =
p.Element("OpeningHours").Value,
                                ContactsInfo =
p.Element("Contacts").Value,
                                DescriptionInfo =
p.Element("Description").Value,
                                ImagesListSource =
p.Element("ImagesFolderPath").Value,
                                VideoSource =
p.Element("VideoFilePath").Value
                            };
            //per ogni nodo "Location" trovato nel file, creo
un'istanza di LocationModel e la aggiungo a
LocationViewModel.Locations
            foreach (var location in query)
            {
                LocationModel loc = new LocationModel(
Convert.ToString(location.Name),
Convert.ToDouble(location.Latitude,NumberFormatInfo.InvariantInfo),
Convert.ToDouble(location.Longitude,NumberFormatInfo.InvariantInfo),
Convert.ToString(location.Type),
Convert.ToInt32(location.Period),
Convert.ToBoolean(location.IsLeonardo),
Convert.ToBoolean(location.IsBramante),
Convert.ToBoolean(location.IsBoccioni),
Convert.ToString(location.Artists),
Convert.ToString(location.Importance),
Convert.ToString(location.AddressInfo),
Convert.ToString(location.OpeningHoursInfo),
Convert.ToString(location.ContactsInfo),
Convert.ToString(location.Prices),
Convert.ToString(location.DescriptionInfo),
Convert.ToString(location.ImagesListSource),
Convert.ToString(location.VideoSource)
                );
                newLocations.Add(loc);
            }
            _locations = newLocations;
            _infoWindows = new
ObservableCollection<LocationModel>();
            NotifyPropertyChanged("Locations");
        }

        #endregion

        #region Methods

        private void UpdatePushPinVisibility()
        {
            Locations.ToList().ForEach(loc => loc.IsPushPinVisible =
IsAnyValidGroupVisible(loc.Type, loc.Period, loc.RelatedArtists));
        }

        private bool IsAnyValidGroupVisible(string locType, Int32
locPeriod, string locArtists)
        {
```

```
            switch (locType)
            {
                case "Museum":
                    if (AreMuseumPushPinsVisible)
                    { return true; }
                    break;
                case "Monument":
                    if (AreMonumentPushPinsVisible)
                    { return true; }
                    break;
                case "Palace":
                    if (ArePalacePushPinsVisible)
                    { return true; }
                    break;
                case "Door":
                    if (AreDoorPushPinsVisible)
                    { return true; }
                    break;
                case "Church":
                    if (AreChurchPushPinsVisible)
                    { return true; }
                    break;
            }
            switch (locPeriod)
            {
                case 1:
                    if (ArePeriod1PushPinsVisible)
                    { return true; }
                    break;
                case 2:
                    if (ArePeriod2PushPinsVisible)
                    { return true; }
                    break;
                case 3:
                    if (ArePeriod3PushPinsVisible)
                    { return true; }
                    break;
                case 4:
                    if (ArePeriod4PushPinsVisible)
                    { return true; }
                    break;
                case 5:
                    if (ArePeriod5PushPinsVisible)
                    { return true; }
                    break;
                case 6:
                    if (ArePeriod6PushPinsVisible)
                    { return true; }
                    break;
                case 7:
                    if (ArePeriod7PushPinsVisible)
                    { return true; }
                    break;
                case 8:
                    if (ArePeriod8PushPinsVisible)
                    { return true; }
```

```
                    break;
            }

            if (locArtists.Contains("Leonardo da Vinci"))
            {
                if (AreLeonardoPushPinsVisible)
                { return true; }
            }
            if (locArtists.Contains("Bramante"))
            {
                if (AreBramantePushPinsVisible)
                { return true; }
            }
            if (locArtists.Contains("Boccioni"))
            {
                if (AreBoccioniPushPinsVisible)
                { return true; }
            }
            return false;
        }
        #endregion
    }
```

A.4 Struttura dati: esempio di nodo Location

Tabella 8.4 Esempio di nodo in SitiArtisticiMilano.xml

```
<Location>
    <Name>Basilica di San Lorenzo Maggiore</Name>
    <Latitude>45.45818500220775</Latitude>
    <Longitude>9.182079881429672</Longitude>
    <Address>Corso di Porta Ticinese, 39 - 20123 Milano</Address>
    <OpeningHours>Da lunedì a sabato: 07:30-12:30, 14:30-18:30.
Domenica 07:30-18:30</OpeningHours>
    <Prices>Ingresso gratuito</Prices>
    <Contacts>Contatti: Segreteria parrocchiale. TELEFONO:
+39.02.89.40.41.29</Contacts>
    <Description>
      La magnifica basilica paleocristiana …
      All'esterno è ancora ben conservato un meraviglioso
colonnato.
    </Description>
    <Curiosities>L'altezza delle colonne è di metri 8,50;
      …protettore della Confraternita dei facchini.
    </Curiosities>
    <MustSee>
      La Corale di San Lorenzo … musiva paleocristiana a Milano.
    </MustSee>
    <UsefulDetails>Durante le celebrazioni non è consentita la
visita turistica.</UsefulDetails>
    <ImagesFolderPath>C:\...\Immagini\</ImagesFolderPath>
    <VideoFilePath>C:\...\Video Castello Sforzesco\Castello
Sforzesco Milano.wmv</VideoFilePath>
```

```
    <Type>Church</Type>
    <Period>1</Period>
    <IsLeonardo>False</IsLeonardo>
    <IsBramante>False</IsBramante>
    <IsBoccioni>False</IsBoccioni>
    <Importance>Low</Importance>
    <Artists>Martino Bassi</Artists>
  </Location>
```