MARCO GUIDI

MODELLO A&A PER CONTEXT-AWARE APPLICATION

RELAZIONE DEL CORSO: SISTEMI MULTI AGENTI LS, A.A. 2009/2010
Febbraio 2010

Marco Guidi: *Modello A&A per Context-Aware Application*
Relazione del corso: Sistemi Multi Agenti LS, A.A. 2009/2010

E-MAIL:
marco.guidi86@gmail.com

SOMMARIO

In questo articolo si introduce come il modello A&A possa essere utilizzato per la progettazione dei sistemi Context-Aware. Gli Artefatti individuano il mezzo tramite il quale gli Agenti percepiscono il contesto in cui si trovano ed in base a tali percezioni, secondo i propri desideri, gli Agenti generano determinate intenzioni per raggiungere un particolare obbiettivo, a tal fine dovranno eseguire determinate azioni nell'ambiente le quali vengono attuate sempre attraverso gli Artefatti.

INDICE

1 INTRODUZIONE

1.1 INTRODUZIONE

Le applicazioni Context-Aware sono sempre più diffuse e soprattutto stanno assumendo un ruolo molto importante nel campo del mobile computing. La conoscenza del contesto in cui si trova l'utente permette di offrire una vasta gamma di servizi che lo aiutano nella sua vita quotidiana, lavorativa o privata, per gestire al meglio il proprio tempo.

Le applicazioni Context-Aware sono sistemi il cui comportamento è governato dal contesto corrente dell'utente diversamente dalle tradizionali applicazioni dove il comportamento è deciso dagli input forniti dall'utente. Le applicazioni Context-Aware possono percepire l'ambiente e la situazione in cui l'utente si trova attraverso sensori hardware o software e decidere automaticamente il loro comportamento in accordo a delle regole predefinite.

Nel seguente elaborato, si motiva e descrive come il modello di programmazione ad Agenti & Artefatti possa essere utilizzato nello sviluppo di applicazioni Context-Aware.

L'articolo è suddiviso nel seguente modo. Nel primo capitolo si definisco i concetti fondamentali legati alla context-awareness, nel secondo capitolo si cerca di spiegare perché il modello di programmazione ad Agenti possa portare dei vantaggi nello sviluppo di sistemi context-aware in particolare facendo riferimento al modello Agenti e Artefatti (A&A); infine nell'ultimo capito si descrive quale sia il ruolo delle ontologie nella context-awareness.

2 CONCETTI

2.1 CONCETTI

In questa sezione si voglio definire i concetti fondamentali legati alla context-awareness.

2.1.1 *Contesto*

Come prima cosa è importante capire il significato di contesto. Il concetto di contesto è ancora oggetto di discussione e negli anni diverse definizioni sono state proposte. Come definito in [1], *"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."* ed inoltre, *"The context acts like a set of constraints that influence the behavior of a system (a user or a computer) embedded in a given task"* [2].

Questa definizione rende facile per uno sviluppatore individuare il contesto per un determinato scenario applicativo. Se una certa informazione è utile per caratterizzare la situazione di un partecipante in una interazione significa che questa informazione sia di contesto.

Esempio, aver annotato nella propria agenda un appuntamento significa che in tale giorno l'utente dovrà effettuare qualcosa legato a quell'appuntamento.

Si potrebbero utilizzare le informazioni fornite dall'agenda con informazioni spazio temporali correnti dell'utente per dedurre che esso stia partecipando ad una riunione ed in particolar caso si potrebbe selezionare sul telefonino dell'utente il profilo che filtra solo chiamate di emergenza per non essere disturbato.

In questo esempio le informazioni di contesto che caratterizzano la situazione dell'utente sono informazioni spazio-temporali legate all'utente ed informazioni derivate dall'agenda che ci permettono di capire il motivo per cui l'utente si trova in una determinata situazione e di conseguenza influenzare il comportamento del sistema.

2.1.2 *Categorie di contesto*

Nell'articolo [12], Ryan, suggerisce come categorie di contesto il luogo, l'ambiente, l'identità e il tempo; mentre Schilit nell'articolo [13] elenca importanti aspetti di contesto come: dove ti trovi?, chi sei e con chi? e che risorse ci sono vicine.

Al fine di comprendere facilmente quali siano le componenti che determinano una situazione, emerge che il contesto, per una data situazione, può essere specificato rispondendo alle seguenti domane: Chi? Cosa? Dove? Quando? Perché? Come?. "Chi" indica il soggetto, l'agente che fa l'azione, "Cosa" rappresenta l'oggetto, qualcosa

che sostiene l'azione, "Dove" e "Quando" forniscono informazioni spazio-temporali sull'azione considerata, "Perché" definisce le intenzioni, l'obbiettivo del soggetto, ed infine "Come" specifica la procedura necessaria per realizzare l'azione.

Certe categorie ti contesto, però, risultano più importanti di altre. Queste sono il *luogo*, l'*identità*, l'*attività* e il *tempo*. Esse identificano le principali categorie di contesto per determinare la situazione di una particolare entità, ed oltre a rispondere alle domande chi, dove, quando e cosa, fungo da indici per altre sorgenti di informazioni contestuali per individuare informazioni contestuali secondarie [1]. Esempio, data l'identità di una persona si possono acquisire molte altre informazioni correlate come numero di telefono, indirizzo, email, compleanno etc...

La differenza tra la lista di categorie di contesto definita sopra e quella proposta da Ryan nell'articolo [12] sta nell'uso di attività invece di ambiente. Questo perché ambiente è un sinonimo di contesto e non aggiunge informazioni utili a determinare il contesto.

2.1.3 *Context-Aware Computing*

Anche per context-aware computing sono state proposte diverse definizioni negli anni. Utilizzare delle informazioni di contesto nelle applicazioni software significa context-aware computing. Tali applicazioni si adeguano al contesto rilevato cambiando il loro comportamento senza l'intervento esplicito dell'utente, permettendo così di aggiungere funzionalità, aumentare l'usabilità e l'efficacia [6].

Come descritto nell'articolo [1], la precedente definizione è leggermente imprecisa in quanto tende ad escludere alcune categorie di applicazioni context-aware. Ad esempio, consideriamo una applicazione che semplicemente mostra a video il contesto dell'utente all'utente, tale applicazione non cambia il suo comportamento ma è context-aware. Per questo viene proposta un'altra definizione più generale per cercare di includere tutte le categorie di applicazioni context-aware: *"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task."*

2.1.4 *Classificazione di caratteristiche per applicazioni context-aware*

É possibile classificare le caratteristiche delle applicazioni context-aware secondo le funzionalità e il loro scopo. Secondo quanto descritto da Bill N. Schilit nell'articolo [13], è possibile definire una tassonomia di classi di applicazioni context-aware secondo il prodotto di due dimensioni ortogonali: se l'attività è recuperare informazioni o eseguire comandi, e se l'attività è eseguita manualmente o automaticamente.

	manual	automatic
information	proximate selection contextual information	automatic contextual reconfiguration
command	contextual commands	context-triggered actions

Secondo Bill N. Schilit si definiscono le seguenti classi di applicazioni context-aware:

Proximate selection - É una tecnica di interazione con l'utente dove le informazioni più vicine al contesto corrente dell'utente sono messe in evidenza in modo tale da facilitarne la scelta per l'utente.

Automatic contextual reconfiguration - Sono sistemi in grado di aggiungere nuovi componenti, rimuovere vecchi componenti o alterare le connessioni tra i componenti in base al contesto.

Contextual information and commands - Spesso le azioni possono essere predette in base alla situazione. Effettuare interrogazioni su informazioni contestuali può produrre risultati differenti in base al contesto, in maniera analoga il contesto può parametrizzare comandi contestuali, ad esempio il comando di stampa deve utilizzare la stampante più vicina.

Context-triggered actions - Sono applicazioni definite da regole IF-THEN utilizzate per specificare come il sistema deve adattarsi al contesto. L'informazioni riguardo al contesto è la condizione di trigger di un comando. Questa categoria è molto simile a quella precedente: la differenza sta nel fatto che le azioni sono invocate automaticamente sulla base di regole predefinite.

Diversamente, ma con concetti molto simili Pascoe nell'articolo [8] definisce una tassonomia di caratteristiche di context-awareness, diversamente da quanto visto prima dove si definivano classi di applicazioni context-aware. In realtà le caratteristiche di context-awareness mappano bene le classi di applicazioni context-aware di Schilit.

Contextual sensing - É l'abilità di rilevare informazioni contestuali e presentarle all'utente. Questa classe è molto simile a *proximate selection* eccetto che, l'utente non deve necessariamente selezionare un'informazione per ottenere più dettagli.

Contextual adaptation - É l'abilità di eseguire o modificare servizi automaticamente in base al contesto corrente. Concetto del tutto equivalente a *context-triggered actions* di Schilit.

Contextual resource discovery - Permette di localizzare e sfruttare risorse e servizi che sono rilevanti al contesto dell'utente. Concetto equivalente a *automatic contextual reconfiguration*.

Contextual augmentation - É l'abilità di associare informazioni contestuali ai dati dell'utente per far si che l'utente potrà vedere i dati del contesto associato.

Sia Pascoe che Schilit hanno presentato un lista di abilità per sfruttare risorse rilevanti al contesto dell'utente, abilità per eseguire comandi automaticamente in base al contesto dell'utente e abilità per visualizzare informazioni rilevanti all'utente. Pascoe aggiunse un dettaglio in più nella visualizzazione di informazioni rilevanti all'utente includendo anche la visualizzazione del contesto stesso all'utente ed inoltre aggiunse una tassonomia in più la rispetto a Schilit: *contextual augmentation* ovvero l'abilità di aggiungere informazioni contestuali ai dati dell'utente. In fine però, la tassonomia di Pascoe non supportava la presentazione di comandi rilevanti per il contesto dell'utente, quello che nella tassonomia di Schilit è definito *contextual commands*.

Dalle definizioni precedenti, combinando le idee e tenendo conto delle tre principali differenze, nell'articolo [1] vengono definite tre categorie che riassumo quelle viste in precedenza:

- *presentation of information and services to a user*;
- *automatic execution of a service*;
- *tagging of context to information for later retrieval*.

Presentation of information and services to a user è una combinazione delle categorie *proximate selection* e *contextual commands* di Schilit e la nozione di *presenting context* di Pascoe.

Automatic execution of a service equivale alla stessa categoria *context-triggered actions* di Schilit e *contextual adaptation* di Pascoe.

Tagging of context to information for later retrieval è la stessa categoria di *contextual augmentation* descritta da Pascoe.

3

MODELLO DI PROGRAMMAZIONE

3.1 MODELLO DI PROGRAMMAZIONE

Come sviluppare un'applicazione context-aware è uno dei temi principali della ricerca sul context-aware computing. I punti non ancora chiari che sollevano tante interrogazioni sono: l'acquisizione del contesto, la rappresentazione del contesto, l'interpretazione del contesto e l'astrazione del contesto, così come anche modelli di programmazione e sviluppo. Per questo lo sviluppo di applicazioni context-aware rimane sempre molto difficoltoso.

Spesso, la scelta delle informazioni di contesto utilizzate nelle applicazioni è guidata dai meccanismi hardware/software a disposizione per acquisire il contesto. Tale approccio bottom-up ostacola la flessibilità e l'estensibilità delle applicazioni portando i dettagli e le carenze dei dispositivi al livello logico dell'applicazione.

Molto spesso nelle applicazioni, il legame tra contesto e comportamento è fisso rendendo difficoltoso prevedere situazioni in cui: un nuovo contesto può essere introdotto, il legame tra comportamento e contesto può cambiare oppure possono essere aggiunti nuovi comportamenti.

La mancanza di un modello di programmazione porta, in genere, a sviluppare applicazioni contenenti l'intera parte di acquisizione del contesto, interpretazione del contesto, definizione di regole ed esecuzione del comportamento rendendo lo sviluppo complicato e dispendioso.

3.1.1 *Modello di programmazione a livelli*

Per facilitare il processo di sviluppo di applicazioni context-aware, nell'articolo [4] viene proposto un modello di programmazione basato sul principio dell'ingegneria del software della "separation of concerns". Come mostrato in figura Figura 1 nella pagina seguente, il modello è composto da tre livelli: application layer, context conversion layer e device layer.

L'*application layer* si preoccupa della logica applicativa dell'applicazione. Esso è costituito dalle specifiche del contesto, dalle specifiche di comportamento a dalle specifiche che definiscono il legame tra il comportamento e il contesto.

Il *device layer* si occupa dell'acquisizione dei dati di contesto grezzi ed è costituito da un insieme di sensori.

Infine, il *context conversion layer* si preoccupa di convertire i dati grezzi forniti dal device layer in specifiche di contesto per l'application layer.

Sicuramente il modello sopra descritto, individua un buon punto di partenza per progettare applicazioni context-aware secondo i principi dell'ingegneria del software, ma come descritto in precedenza il

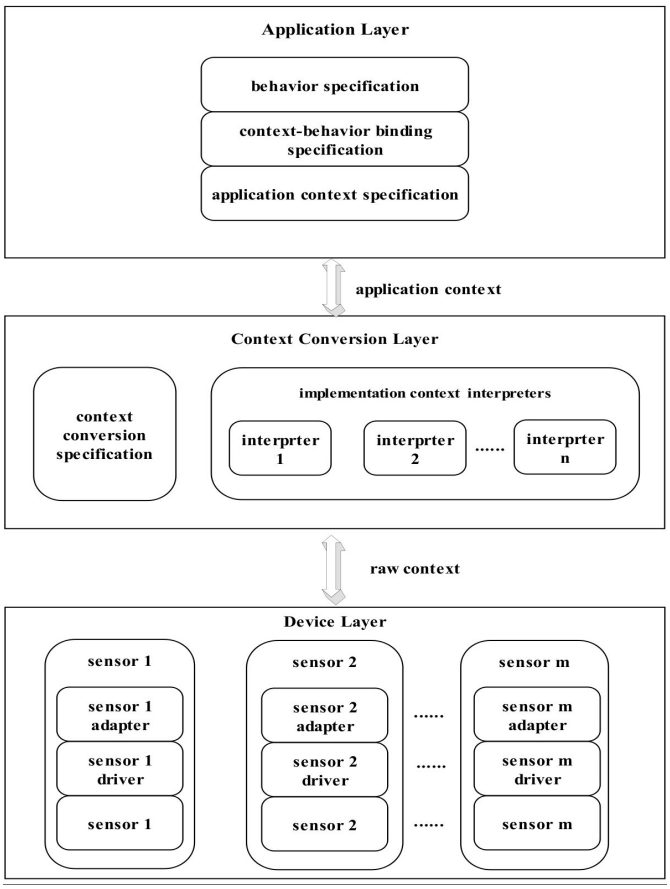

Figura 1: Modello di programmazione per applicazioni context-aware [4]

contesto è qualsiasi informazione che caratterizza la situazione di una entità. Ad esempio si prenda come riferimento la vita quotidiana, in cui un individuo vive in un particolare contesto/ambiente che può essere caratterizzato dalla conoscenza dell'individuo, dall'azione stessa che l'individuo sta svolgendo e dall'interazione dell'individuo con oggetti o altri suoi simili.

Si intuisce la necessità di modellare un ambiente (contesto) in cui ci siano entità attive che cooperano, compiono azioni, utilizzano ed osservano gli oggetti che gli stanno attorno ed inoltre si intuisce la necessità di opportuni modelli di coordinazione. Perciò si rendono necessarie astrazioni di prima classe per definire entità attive, oggetti osservabili e ambienti.

3.1.2 *Modello ad Agenti*

Un agente è un'entità autonoma in grado di osservare ed effettuare azioni sull'ambiente che lo circonda.

Per come esso è definito, si nota uno stretto legame con le caratteristiche richieste da un'applicazione context-aware.

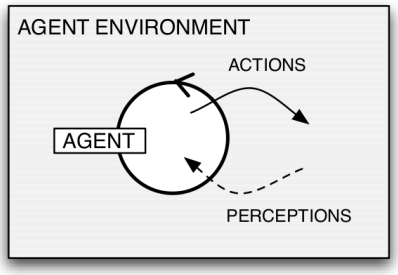

Figura 2: Rappresentazione astratta di un agente che percepisce ed esegue azioni nell'ambiente in cui è situato. [10]

Agenti Intelligenti

Di particolare interesse risultano gli Agenti Intelligenti. Un agente intelligente è descritto in termini di stati e attitudini mentali, contiene una rappresentazione simbolica del mondo ed è in grado di prendere decisioni per raggiungere un qualche obbiettivo ragionando sui propri stati mentali. Un agente intelligente è non deterministico; in ogni istante potrebbe avere più azioni eseguibili ed in ogni istante potrebbe avere obbiettivi multipli da raggiungere; le migliori azioni che l'agente dovrà svolgere per raggiungere i propri obbiettivi dipendono dal dallo stato dell'ambiente in cui si trova (contesto).

Gli agenti con stati mentali governano il loro comportamento sulla base di stati interni (simulano gli stati cognitivi); gli stati interni si suddividono in: *Epistemic States* che rappresentano la conoscenza del mondo (percepts, beliefs) e *Motivational States* che rappresentano gli obbiettivi (goals, desires).

Il processo di aggiornamento delle informazioni del mondo è definito *Epistemic Reasoning*, mentre il processo della scelta dell'azione da svolgere in base agli attuali stati mentali si chiama *Pratical Reasoning* e consiste di due attività di base: *Deliberation*, quando un agente prende la decisione in base a "cosa" desidera ottenere o fare; e *Means-Ends Reasoning*, quando un agente prende la decisione in base a "come" ottenere un certo stato.

Gli agenti BDI, sono un'architettura astratta proposta da A. S. Rao and M. P. Georgeff [9] che comprende strutture dinamiche e globali per rappresentare i *Beliefs*, i *Desires* e le *Intentions* (BDI) di un agente. I *Belief* sono insieme di informazioni riguardanti l'ambiente, il contesto; i *Desire* rappresentano i desideri, gli obiettivi che un agente vuole raggiungere; le *Intention* rappresentano i *Desires* per i quali l'agente ha pianificato delle azioni da eseguire.

I *Plans*, piani, sono il mezzo che l'agente utilizza per cambiare il mondo in modo tale da raggiungere i propri *Desires*. Un piano è implementato secondo strutture procedurali; è definito da un "body", descritto dall'insieme delle attività da eseguire per il successo dell'esecuzione del piano; è definito da una condizione secondo la quale il piano può essere scelto per una certa condizione ed è definito da una condizione di contesto ovvero la situazione che deve essere presente per l'esecuzione del piano.

Si può intuire che il paradigma ad agenti sia utile per la risoluzione dei tipi di problemi tipici delle applicazioni context-aware dove in generale viene richiesta una certa dinamicità del comportamento dell'applicazione in base al contesto in cui ci si trova.

Prendendo in considerazione il modello di programmazione proposto nell'articolo [4], gli agenti sono utili per esprimere la logica applicativa dell'applicazione e quindi definire l'*application layer*. In particolare, facendo riferimento agli agenti BDI abbiamo dei concetti utili per rappresentare il contesto in termini di *Belief* e il comportamento del sistema attraverso dei *Plans*. Ciò che ci occorre ora, è un'infrastruttura per la gestione e l'acquisizione del contesto da parte degli agenti.

3.1.3 *Modello A&A*

Come individuato nel precedente modello di programmazione a livelli, proposto nell'articolo [4], ed in base ai principi dell'ingegneria del software, per una buona progettazione di un sistema context-aware sarebbe bene separare la logica applicativa dalla logica che definisce l'acquisizione del contesto e la conversione del contesto in informazioni utilizzabili dal livello applicativo.

Nel modello Agenti e Artefatti (A&A) [11], le azioni e le percezioni dell'agente sono legate all'utilizzo di un artefatto. Mentre il concetto di agente è strettamente legato alla nozione di comportamento pro-attivo, all'astrazione di artefatto è associata alla nozione di utilizzo ovvero gli artefatti sono le entità utilizzate dagli agenti per lavorare.

Un artefatto rappresenta perciò una risorsa o un tool che un agente può dinamicamente creare, usare, condividere e manipolare. L'artefatto è un'entità passiva, progetta per incapsulare certi tipi di funzionalità strutturate in termini di operazioni, la quale esecuzione può essere innescata dagli agenti attraverso un'interfaccia d'uso. Differentemente dalla nozione di interfaccia d'uso in OO, quando un agente innesca l'esecuzione di una operazione questa viene eseguita in maniera asincrona al controllo dell'agente.

Oltre ad offrire funzionalità agli agenti, un artefatto è caratterizzato da *proprietà osservabili* e da *eventi osservabili* entrambi percepibili dagli agenti utilizzandolo ed osservandolo, Figura 3 nella pagina successiva. Una *proprietà osservabile* è un elemento utile per ispezionare lo stato dinamico di un artefatto senza interrogarlo mentre un *evento osservabile* è un evento generato dall'artefatto non legato necessariamente ad uno stato.

Infine, oltre agli agenti e agli artefatti, la nozione di *workspace* completa l'insieme delle astrazioni per il modello A&A. Un *workspace* è un contenitore logico di agenti e artefatti utile per definire la topologia dell'ambiente di lavoro.

Un workspace fornisce una nozione di località per gli agenti: un agente può partecipare a uno o più workspace e può utilizzare solo gli artefatti presenti nei workspace. Tali concetti possono essere utilizzati per definire, a livello astratto, un modello distribuito di una applicazioni: *a working environment*, Figura 4 nella pagina seguente, che corrisponde ad una applicazione o SMA. Un singolo workspace può

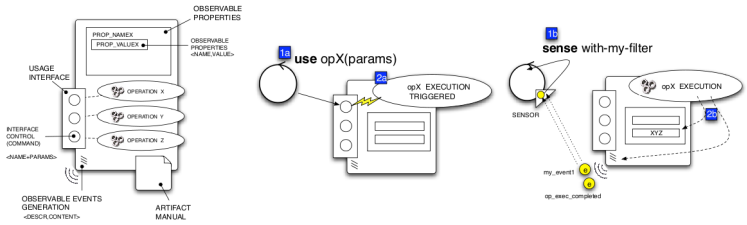

Figura 3: Rappresentazione astratta di un artefatto (sulla sinistra), e di un agente che utilizza un artefatto, innescando l'esecuzione di una operazione (al centro), e osservando un evento generato dall'artefatto (sulla destra). [10]

sia essere mappato su un singolo nodo della rete oppure può essere distribuito su più nodi della rete.

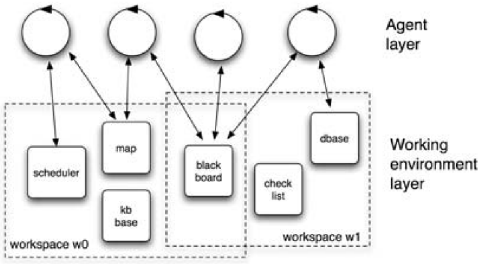

Figura 4: Rappresentazione astratta di un working environment con due workspace, con diversi tipi di artefatti all'interno. [11]

Come previsto dal modello A&A, gli artefatti sono il mezzo tramite il quale gli agenti acquisiscono informazioni ed effettuano azioni sull'ambiente, per questo motivo possono essere una buona astrazione per definire un'infrastruttura utilizzabile dagli agenti per la gestione e l'acquisizione del contesto.

Per di più, gli artefatti fungono anche da canale di comunicazione per la coordinazione di più agenti tramite il quale possono scambiarsi e condividere informazioni di contesto.

I workspace inoltre, sono strutture adatte per aggregare artefatti che o incapsulano risorse o reificano elementi che vogliamo rappresentare per: creare contesti per la coordinazione delle persone, fornire la conoscenza delle informazioni e supportare la scoperta di servizi e risorse [7].

3.1.4 *Esempio pratico*

Pensato per gli smartphone di ultima generazione, si vuole considerare un sistema di supporto all'utente per gestire gli appuntamenti. In base agli appuntamenti dell'utente il sistema deve fornire informazioni o attuare operazioni di aiuto all'utente stesso.

Il sistema deve essere in grado di:

1. Informare l'utente di eventuali cambiamenti di orario o luogo degli appuntamenti, in particolare si prende in considerazione come appuntamento la riunione;

2. Impostare un particolare profilo sullo smartphone in base alla situazione in cui si trova l'utente.

La prima cosa da fare è individuare gli elementi che caratterizzano il sistema.

Per quanto riguarda il primo requisito del nostro sistema, immaginando come l'uomo affronta questo tipo di problema nella vita reale si possono individuare i seguenti elementi: un'agenda, strumento utilizzato dalle persone per annotare i propri appuntamenti; una bacheca, ormai in quasi tutte le aziende/istituzioni è presente una bacheca utilizzata per comunicare col personale in maniera indiretta, si pensi ad esempio all'università dove nella bacheca vengono affissi continuamente avvisi riguardanti esami, riunioni, eventi, etc..., la bacheca perciò rappresenta lo strumento che l'uomo utilizza per captare informazioni che possono riguardare i propri appuntamenti.

Nel secondo requisito occorre individuare cosa caratterizza la situazione dell'utente e quali siano gli strumenti utili all'utente per impostare il profilo del proprio smartphone.

La situazione dell'utente può essere caratterizzata da un appuntamento; legate alla nozione di appuntamento ci saranno sicuramente una nozione di luogo e di tempo. Occorrono perciò strumenti per tener traccia del tempo e della posizione geografica dell'utente in modo tale da ricollegarsi ad un appuntamento il quale caratterizzerà la situazione dell'utente.

Una volta individuati tutti questi elementi del sistema, occorre mappare ognuno di essi sulle astrazioni del modello che si intende utilizzare, nel nostro caso il modello A&A.

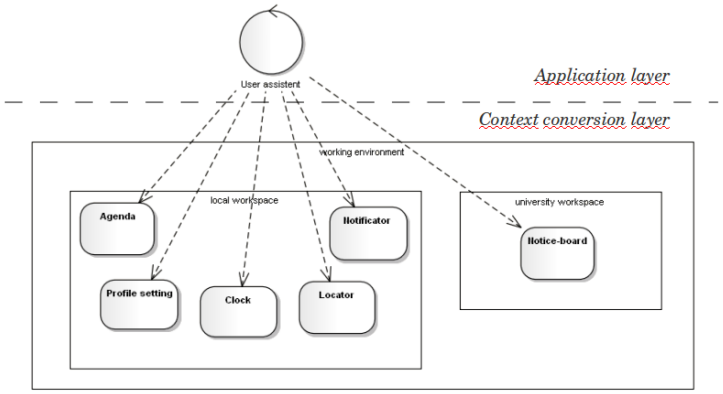

Figura 5: Rappresentazione astratta del working environment del nostro sistema

Si deve cercare di individuare quali siano le entità mappabili come agenti e quali come artefatti. Nel nostro sistema si può individuare un'entità attiva, chiamiamola Assistent Agent, che deve svolgere le attività che avrebbe dovuto fare l'utente; il suo ruolo sarà quello di

avvisare l'utente di eventuali cambiamenti nei propri appuntamenti e di impostare un particolare profilo dello smartphone a seconda della situazione; mentre si possono individuare i seguenti artefatti utilizzabili dall'Assistent Agent per svolgere il proprio lavoro: agenda, notice-board, locator, clock, profile setting e notificator, Figura 5 nella pagina precedente.

4

MODELLARE IL CONTESTO E RAGIONARE SUL CONTESTO

4.1 MODELLARE IL CONTESTO E RAGIONARE SUL CONTESTO

Alla base della context-awareness è necessario un modello formale per rappresentare il contesto in maniera tale che possa essere interpretato dalle macchine ed in base al quale sia possibile effettuare ragionamenti. Soprattutto nei sistemi distribuiti è particolarmente importante permettere lo scambio e la condivisione di informazioni di contesto tra differenti entità computazionali per renderle interoperabili.

4.1.1 *Modellazione del contesto*

La modellazione del contesto prevede di specificare tutte le entità e le relazioni tra esse le quali sono necessarie per descrivere il contesto completo, ad esempio informazioni su località, tempo, l'utente e le sue attività correnti o pianificate.

Un particolare tipo di problema che sorge nei sistemi distribuiti eterogenei legato alla modellazione del contesto è l'utilizzo di schemi proprietari per rappresentare il contesto ostacolando così l'interoperabilità di differenti entità computazionali. L'utilizzo di ontologie comuni per descrivere il contesto risolverebbe questo problema.

4.1.2 *Ragionare sul contesto*

Ragionare sul contesto significa dedurre automaticamente fatti impliciti da informazioni contestuali esplicite.

Esso può essere usato per ottenere informazioni di più alto livello, ad esempio il sensore di movimento rileva una presenza in cucina e il sensore del fornello segnala che è accesso, si può dedurre che una persona in cucina stia cucinando.

Il ragionamento sul contesto può essere utilizzato anche per verificare e risolvere inconsistenze sui dati rilevati da diversi sensori. Ad esempio quando un sensore segnala che il PDA dell'utente è situato nel suo uffico mentre la sua auto è a casa, non si può essere certi che l'utente sia in uffico. Ulteriori informazioni sono necessarie per risolvere l'inconsistenza, ad esempio quando la key card dell'utente è utilizzata per aprire la porta dell'ufficio si può essere certi che l'utente sia in ufficio.

4.1.3 *Ontologie*

Il termine "ontologia" ha origine dalla filosofia e si riferisce ad una teoria sulla natura dell'esistenza.

In computer science una ontologia si definisce come un sistema formalmente definito di concetti e relazioni tra questi concetti.

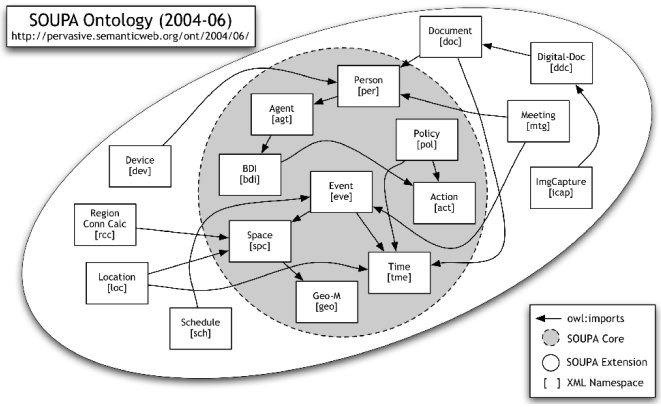

Figura 6: SOUPA è costituito da due insiemi di ontologie: SOUPA Core e SOUPA Extension. [3]

Secondo M. Gruninger and J. Lee [5] ci sono tre principali aree di applicazioni per le ontologie:

- comunicazione e condivisione di conoscenza: l'ontologia è utilizzata come vocabolario comune per differenti agenti.
- Ragionamento logico/ inferenza logica: una ontologia può essere utilizzata per dedurre conoscenza implicita da conoscenza esplicita applicando regole.
- Riuso della conoscenza: ontologie comuni possono essere riutilizzate quando di costruiscono ontologie per domini specifici.

Gli elementi tipici di una ontologia sono:

- Concetti e attributi;
- Tassonomie per categorizzare concetti attraverso generalizzazioni e specializzazioni;
- Relazioni tra concetti;
- Assiomi per definire dichiarazioni sempre vere;
- Individui (o fatti) sono esempi di concetti e le loro relazioni.

Ci sono diversi linguaggi che permettono di definire delle ontologie, esempio Ontolingua, LOOM e OWL Web Ontology Language.

CONON, COBRA-ONT e SOUPA sono ontologie basate su OWL pensate per essere utilizzate negli ambienti di pervise computing per permettere di modellare e ragionare sul contesto.

In particolare SOUPA - "Standard Ontology for Ubiquitous and Pervasive Applications", [3] è costituito da un due insiemi distinti di ontologie ma in relazione tra loro: SOUPA Core e SOUPA Extension, Figura 6.

L'insieme delle ontologie in SOUPA Core consiste in un vocabolario per esprime concetti associati a persone, agenti, belief-desire-intention

(BDI), azioni, politiche, tempo, spazio ed eventi; mentre SOUPA Extension dimostra come sia possibile definire nuove ontologie estendendo quelle di SOUPA Core definendo un'estensione del vocabolario per specificare particolari domini applicativi: Meeting & Schedule, Document & Digital Document, Image Capture, Region Connection Calculus e Location.

5 CONCLUSIONI

5.1 CONCLUSIONI

In questa relazione si presenta come l'approccio basato sul modello di programmazione A&A si utile in tutto il processo di sviluppo di sistemi context-aware a partire dall'analisi, poi nella progettazione ed infine nello sviluppo.

Dopo aver descritto i concetti fondamentali della context-awareness nel primo capitolo, nel secondo capito si descrive come il paradigma ad agenti e artefatti si presti bene nella progettazione di sistemi context-aware secondo i principi dell'ingegneria del software, esso infatti fornisce l'astrazione dell'agente per definire la logica applicativa del sistema mentre attraverso gli artefatti permette di definire un livello per la gestione e l'acquisizione del contesto.

Sempre nel secondo capitolo, si presenta inoltre un esempio pratico che permette di capire come attraverso il modello A&A sia facile ed intuitivo identificare gli elementi utili nella realizzazione del sistema e come essi siano riconducibili agli elementi che nella vita quotidiana utilizziamo per risolvere problemi.

Infine, nell'ultimo capitolo si introduce come il concetto di ontologia sia utile al fine di realizzare sistemi aperti interoperabili tra loro.

A questo punto, dopo aver ben motivato l'utilizzo del paradigma A&A, in lavori futuri si potrebbero svolgere esperimenti pratici ed implementativi utilizzando differenti tecnologie e confrontarne i pregi ed i difetti che il modello ad A&A può introdurre.

[1] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307, London, UK, 1999. Springer-Verlag.

[2] Mary Bazire and Patrick Brézillon. Understanding context before using it. In *Modeling and Using Context*, pages 29–40. 2005.

[3] Harry Chen, Filip Perich, Tim Finin, and Anupam Joshi. Soupa: Standard ontology for ubiquitous and pervasive applications. In *In International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 258–267, 2004.

[4] Weichang Du and Lei Wang. Context-aware application programming for mobile devices. In *C3S2E '08: Proceedings of the 2008 C3S2E conference*, pages 215–227, New York, NY, USA, 2008. ACM.

[5] Michael Grüninger and Jintae Lee. Ontology applications and design - introduction. *Commun. ACM*, 45(2):39–41, 2002.

[6] Stefano Mizzaro, Elena Nazzi, and Luca Vassena. Retrieval of context-aware applications on mobile devices: how to evaluate? In *IIiX '08: Proceedings of the second international symposium on Information interaction in context*, pages 65–71, New York, NY, USA, 2008. ACM.

[7] Andrea Omicini, Alessandro Ricci, and Giuseppe Vizzari. Smart environments as agent workspaces. *Ubiquitous Computing and Communication Journal*, Special Issue - CPE, June 2008. Special Issue on Coordination in Pervasive Environments.

[8] Mr. Jason Pascoe. Adding generic contextual capabilities to wearable computers. In *ISWC '98: Proceedings of the 2nd IEEE International Symposium on Wearable Computers*, page 92, Washington, DC, USA, 1998. IEEE Computer Society.

[9] A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proc. of the Third International Conference (KR'92)*, pages 439–449. Kaufmann, San Mateo, CA, 1992.

[10] Alessandro Ricci and Mirko Viroli. simpA: An agent-oriented approach for prototyping concurrent applications on top of Java. In Vasco Amaral, editor, *5th International Symposium on Principles and Practice of Programming in Java (PPPJ 2007)*, volume 272 of *ACM International Conference Proceeding*, pages 185–194, Lisboa, Portugal, 5–7 September 2007. ACM.

[11] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. A general purpose programming model & technology for developing working environments in MAS. In Mehdi Dastani, Amal El Fallah Seghrouchni, Alessandro Ricci, and Michael Winikoff, editors, *5th International Workshop "Programming Multi-Agent Systems" (PROMAS 2007)*, pages 54–69, AAMAS 2007, Honolulu, Hawaii, USA, 15 May 2007.

[12] N. S. Ryan, J. Pascoe, and D. R. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer Applications in Archaeology 1997*, British Archaeological Reports, Oxford, October 1998. Tempus Reparatum.

[13] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *WMCSA '94: Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*, pages 85–90, Washington, DC, USA, 1994. IEEE Computer Society.