

UNIVERSITÀ DEGLI STUDI DI BOLOGNA

FACOLTÀ DI INGEGNERIA

Corso di Laurea Specialistica in Ingegneria Informatica

**ANALISI E PROGETTO DI UN SISTEMA MULTI-AGENTE
PER L'INTERAZIONE AVANZATA DOCENTE/STUDENTE:
ORGANIZZAZIONE E COORDINAZIONE**

**Tesi di laurea in:
LINGUAGGI E MODELLI COMPUTAZIONALI L-S**

Tesi di laurea di
Leonardo Nicoli

Relatore
Prof. Enrico Denti

Correlatori
Prof. Andrea Omicini
Dott. Ing. Alessandro Ricci

Parole chiave:
**COORDINAZIONE, AGENTI, CENTRO DI TUPLE, TuCSoN,
SODA**

Sessione AUTUNNALE (2)

Anno Accademico 2003-2004

*Ringrazio di vero cuore Ambra
e tutti coloro che mi sono stati vicino.*

INTRODUZIONE

Nell'ambito della progettazione di complessi sistemi software, il paradigma ad agenti nasce allo scopo di far fronte a molte delle problematiche che emergono in un contesto fortemente dinamico e distribuito, come può essere la rete Internet.

Attualmente, infatti, il paradigma di programmazione più diffuso è quello ad oggetti, che consente di affrontare agevolmente un problema scomponendolo in sottoproblemi più semplici. Purtroppo, nonostante la sua grande potenza espressiva, il modello object-oriented induce un tipo di architettura in cui le diverse entità in gioco, pur separate, generalmente necessitano dell'aiuto di altri oggetti per far fronte alle proprie responsabilità. In altre parole, gli oggetti di un sistema sono caratterizzati da varie dipendenze reciproche, senza le quali il sistema non potrebbe funzionare: il funzionamento deriva infatti da una continua e reciproca invocazione di metodi tra gli oggetti stessi. Inoltre, gli oggetti sono tipicamente passivi e richiedono un meccanismo di attivazione che scateni l'inizio di ogni elaborazione.

È facile notare che un'architettura come questa viene messa in crisi dalle nuove esigenze dei più evoluti sistemi software, che tipicamente sono (i) distribuiti in un ambiente dinamico ed imprevedibile e (ii) fortemente eterogenei, in quanto integrano sottosistemi già esistenti e non standardizzati.

Per far fronte a queste nuove esigenze è nato il modello orientato agli agenti, consolidato da molti recenti studi sulla teoria degli agenti, che coinvolge diverse discipline informatiche, tra le quali l'intelligenza artificiale, l'ingegneria del software, i linguaggi, le

reti di calcolatori e i sistemi distribuiti.

In sintesi, un agente è definito come un'entità autonoma, intelligente, proattiva e sostanzialmente indipendente dalle altre, caratterizzata da una limitata conoscenza del mondo che la circonda e da una capacità altrettanto limitata di agire su di esso. Quindi, in base alle proprie conoscenze e credenze, un agente è libero di ragionare, decidere ed agire di sua iniziativa, allo scopo di conseguire i propri obbiettivi.

Tipicamente, anche se autonomo, un agente non è autosufficiente: in generale, per portare a termine la sua attività necessita di alcune risorse, che possono essere anche remote. Per questo un agente può anche essere mobile, cioè dotato della capacità di spostarsi da un nodo all'altro di una rete e riprendere la propria esecuzione nella nuova destinazione: in questo modo l'agente può decidere in qualsiasi momento di avvicinarsi alle risorse di cui ha bisogno, per potervi accedere più facilmente.

In accordo a questo modello, un sistema complesso può essere modellato come un insieme di agenti che collaborano per conseguire un obbiettivo comune. In quest'ottica il concetto di coordinazione degli agenti assume una grande importanza e suggerisce di considerare lo spazio di interazione come un'entità di prima classe, una sorta di "colla" che riunisca l'insieme degli agenti in una vera e propria *società*.

A questo scopo sono stati proposti alcuni modelli ed architetture di coordinazione avanzate, che attribuiscono al mezzo di coordinazione degli agenti un'importanza significativa. Questi modelli hanno portato allo sviluppo di infrastrutture che offrono agli agenti un efficace supporto per la loro interazione. Una di queste infrastrutture di coordinazione, chiamata TuCSoN [ref 3], è stata realizzata da un gruppo di ricercatori dell'università di Bologna, e consente una comunicazione efficiente ed evoluta tra agenti mobili via Internet.

Un sistema multi-agente costruito su un'opportuna infrastruttura di coordinazione rappresenta quindi una vera e propria società di

agenti, non solo un'aggregazione di componenti separate. Gli agenti di una società esibiscono un comportamento intelligente non solo individualmente, ma anche rispettando alcune regole sociali che conferiscono al sistema complessivo un'intelligenza collettiva.

Un sistema multi-agente di questo tipo è dunque profondamente diverso da un tradizionale sistema object-oriented, perciò richiede nuove metodologie di progetto orientate agli agenti, che si basino sulle nuove astrazioni introdotte dal modello e permettano al progettista di trattare il sistema multi-agente come tale.

L'obiettivo di questa tesi è realizzare un'applicazione software che permetta una complessa interazione tra docenti e studenti attraverso la rete Internet. Tale applicazione è dunque caratterizzata da evidenti requisiti di interattività e di integrazione di diverse tecnologie: per questo motivo si ritiene opportuno realizzarla basandosi su un modello ad agenti. A tale scopo sarà quindi necessario integrare una infrastruttura di coordinazione di agenti a più tradizionali tecnologie web, e porre particolare attenzione agli aspetti legati alla coordinazione delle diverse attività.

Questa tesi si articola dunque in tre parti. La prima parte è prettamente teorica: analizza il significato e l'importanza del concetto di coordinazione di agenti, descrive i principali modelli che sono stati proposti per rappresentarla, e illustra le principali caratteristiche di una infrastruttura di coordinazione costruita su di essi; la seconda parte illustra la metodologia agent-oriented adottata; infine, la terza parte descrive il progetto del sistema in esame, dall'analisi allo sviluppo, evidenziando con particolare attenzione gli aspetti legati alla coordinazione degli agenti nella gestione dei workflow.

CAPITOLO 1

LA COORDINAZIONE NEI SISTEMI MULTI-AGENTE

1.1. L'IMPORTANZA DELLA COORDINAZIONE

Il paradigma ad Agenti sta diventando sempre più un utile strumento per il progetto di sistemi software avanzati, grazie alla sua potenza espressiva, alla sua capacità di rendere semplice la riduzione di un problema complesso in sottoproblemi più semplici e soprattutto grazie al fatto che gli agenti lavorano come *loci* di controllo indipendenti. Questo consente infatti di progettare le applicazioni astraendo dal controllo, caratteristica molto positiva quando si ha a che fare con ambienti estremamente dinamici.

Tuttavia non è sufficiente scomporre un problema in un insieme di attività e progettare altrettanti agenti indipendenti che le svolgano: nella quasi totalità dei casi queste attività non sono indipendenti, e gli agenti che dovranno svolgerle, in generale, necessiteranno di interagire tra di loro.

Un agente è caratterizzato da tre aspetti fondamentali: è pensabile come un componente software *Autonomo* che *Interagisce* con il suo ambiente per portare a termine la sua *Attività*. Grazie all'autonomia degli agenti, progettare un sistema multi-agente significa in primo luogo individuare le molteplici attività da compiere per raggiungere gli obiettivi desiderati del sistema stesso, quindi assegnare ciascuna di queste attività alla responsabilità di un agente.

Questo approccio però si concentra solo sugli aspetti *intra-agente*,

come la loro struttura interna, la loro visione del mondo, le loro credenze e le loro intenzioni: in questo modo, prima vengono progettati i singoli agenti in accordo all'attività che devono svolgere, ed in seguito vengono semplicemente fatti interagire per realizzare l'insieme multi-agente, senza modellare esplicitamente il comportamento globale del sistema. Questo significa caricare gli agenti della responsabilità di coordinarsi tra loro, in accordo a certi specifici protocolli di interazione che vengono incorporati negli agenti stessi. In quest'ottica, un sistema multi-agente non è che la somma delle sue parti, e l'interazione degli agenti è solo un modo di aggregarli in modo che lavorino assieme. Il problema su cui ci si concentra è quindi il consentire una comunicazione: questo fa sì che esista uno spazio di interazione, ma non aiuta in alcun modo a gestirlo in modo efficiente.

Per gestire meglio la complessità di uno spazio di interazione di agenti, anziché sforzarsi di introdurre a posteriori un meccanismo di interazione, complicando così il progetto dei singoli agenti, sembra più conveniente, dal punto di vista ingegneristico, considerare il problema della coordinazione già in fase di analisi, astraendo dai meccanismi con cui essa possa essere realizzata. Questo consente di evidenziare il fatto che gli agenti hanno un comportamento collettivo: essi costituiscono una vera e propria *società*, dotata di precise regole ed obbiettivi.

Gestire l'interazione significa quindi specificare e rendere effettivi obbiettivi, vincoli e proprietà desiderate che non sono peculiari di un dato agente, ma di un insieme di agenti; significa poter adattare o modificare dinamicamente questa "colla sociale" che unisce un gruppo di agenti, senza dover limitare l'autonomia dei medesimi.

Occorre dunque esaminare anche il mondo in cui gli agenti vivono e socializzano, capire che tipo di individui lo abitano e quali leggi lo governano: le regole sociali non sono affatto una mera limitazione al comportamento degli individui, ma un'importante fonte di intelligenza.

In questo modo è possibile scindere il problema in due dimensioni ortogonali: *Computazione* e *Coordinazione*. La prima si concentra sugli aspetti *intra-agente*, e descrive gli agenti in base all'attività che devono svolgere e alle loro responsabilità; la seconda invece si focalizza sugli aspetti *inter-agente*, cioè sulle attività che richiedono la cooperazione di diversi agenti, e su quali informazioni devono essere scambiate e in che modo. Con questa distinzione lo spazio di interazione diventa un'entità di prima classe, e per essere formalizzato necessita di opportuni modelli e linguaggi di coordinazione.

Questo tipo di approccio suggerisce anche una separazione tra la percezione individuale della coordinazione, cioè dal punto di vista degli agenti, e la coordinazione globale del sistema, permettendo di modellare lo spazio di interazione indipendentemente dalle entità interagenti. Questo tipo di coordinazione è chiamata *oggettiva* proprio perché prescinde dalla visione soggettiva degli agenti coordinati. È detta anche *disaccoppiata* perché indipendente dalla computazione delle entità coordinate.

La coordinazione oggettiva, dunque, concerne problemi inter-agente, cioè la descrizione di come è organizzato l'ambiente del sistema multi-agente e la gestione delle interazioni degli agenti, che avvengono sia tra di loro sia con il loro ambiente. I modelli di coordinazione oggettiva sono fortemente basati sul concetto di interazione mediata: gli agenti sono forniti di specifiche astrazioni, che permettono loro di effettuare le loro azioni (tipicamente comunicazioni) e mediano le interazioni che derivano dalle dipendenze insite nell'insieme degli agenti. Questo ruolo di astrazione tipicamente viene assunto dal generico mezzo di coordinazione: si tratta di un'entità distinta dagli agenti, che regola la loro interazione applicando leggi di coordinazione che rappresentano i vincoli e le regole sociali del sistema.

1.1.1. Progettare lo spazio di coordinazione di un sistema multi-agente

Il concetto della coordinazione oggettiva cambia anche il modo di progettare un sistema multi-agente. In una prima fase, il problema deve essere analizzato e compreso, allo scopo di identificare sia attività individuali, che possono essere svolte da singoli agenti, sia attività collettive, dette Task Sociali, che sono il risultato della collaborazione di un gruppo di agenti. A questo punto, le attività individuali possono guidare il progetto e l'implementazione dei singoli agenti, mentre le attività sociali devono guidare il progetto e l'implementazione dello spazio di interazione, che non è solamente un mezzo di comunicazione dotato di relativi meccanismi di utilizzo, ma che comprende sia i protocolli che le regole di interazione.

A dire il vero, in un modello ad agenti puro, la gestione dell'interazione tra gli agenti potrebbe essere vista come un'attività particolare: in quanto tale potrebbe essere realizzata da particolari agenti fornitori di servizi di mediazione, o *middle-agents*. Tuttavia questa soluzione non sembra essere particolarmente desiderabile, perché un buon mezzo di coordinazione dovrebbe avere certe caratteristiche che un agente tipicamente non possiede. In particolare, un mezzo di coordinazione dovrebbe avere quattro proprietà fondamentali:

- *controllabilità*: il suo comportamento deve poter essere ispezionato sia da esseri umani che da agenti artificiali, quindi le sue specifiche dovrebbero essere espresse in modo dichiarativo, per semplificare la comprensione della semantica delle attività di coordinazione;
- *efficienza/specificità*: deve essere specializzato nella gestione delle interazioni per massimizzare le prestazioni dell'intero sistema, e possibilmente supportare concorrenza, sicurezza, disponibilità ed affidabilità;
- *predicibilità*: il suo comportamento deve riflettere le leggi di

coordinazione desiderate, non deve essere imprevedibile. Gli effetti delle leggi di coordinazione sullo stato del mezzo devono essere definite con precisione mediante un'opportuna semantica.

- *flessibilità*: il suo comportamento dovrebbe poter essere forgiato anche dinamicamente.

Un agente generico non può avere queste caratteristiche, poiché è definito come un'entità autonoma, proattiva e situata: nessuno può programmarla o controllarla, quindi è necessario che la coordinazione degli agenti sia a carico di un'entità apposita, analizzata e progettata parallelamente.

Per formalizzare questa attività di analisi è nata la Teoria della Coordinazione [ref 5]. La teoria della coordinazione comprende diversi formalismi che permettono di modellare e gestire lo spazio di interazione degli agenti: linguaggi, modelli e architetture di coordinazione. I Linguaggi di coordinazione consentono di esprimere il comportamento osservabile degli agenti e di progettare i loro protocolli di interazione, i Modelli di coordinazione consentono di costruire e regolare lo spazio di interazione, e le Architetture di coordinazione forniscono dei pattern per l'organizzazione dell'insieme degli agenti.

1.2. MODELLI E LINGUAGGI DI COORDINAZIONE

Risolvere un problema di coordinazione significa concentrarsi sul meccanismo di comunicazione di un gruppo di agenti in termini di **Protocolli** di interazione, ossia di quali informazioni devono scambiarsi per poter portare a termine la loro attività individuale, e di **Regole** di interazione. A questo scopo è possibile definire un Modello di coordinazione, che consiste in un framework formale in cui l'interazione di agenti software possa essere espressa.

Un modello di coordinazione comprende tre elementi:

- le entità interagenti (*coordinables*), che nel contesto di un sistema multi-agente corrispondono agli agenti stessi;
- il mezzo di coordinazione (*coordination media*), cioè l'astrazione che consente l'interazione degli agenti;
- le leggi di coordinazione (*coordination laws*), che definiscono il comportamento del mezzo in risposta agli eventi di interazione. Tali regole possono essere definite in un linguaggio di coordinazione (*coordination language*), che consiste in un insieme di primitive di interazione con una precisa semantica.

I modelli di coordinazione possono essere classificati in di tre diverse tipologie: *control-driven*, *data-driven* o *ibridi*, che uniscono i vantaggi delle due tipologie precedenti.

1.2.1. Modelli di coordinazione control-driven: comunicazione diretta

Nei modelli control-driven gli agenti comunicano attraverso delle porte di ingresso ed uscita: è il caso, ad esempio, della comunicazione diretta, o peer-to-peer, che è la soluzione più immediata al problema. Gli agenti interagiscono tramite modelli client/server tipici dei sistemi object-based, spesso comunicando a basso livello, sfruttando direttamente meccanismi conosciuti e disponibili come le socket.

Un modello come questo tuttavia non è molto indicato ad un sistema multi-agente distribuito: è di certo molto potente e flessibile, ma vincola molto gli agenti, che devono conoscere chi sono e dove sono gli interlocutori, nonché quando possono o non possono comunicare. Infatti, affinché la comunicazione sia possibile, il client deve saper raggiungere il server e sia il client che il server devono essere attivi nello stesso momento. Questo ostacola l'autonomia e la mobilità degli agenti e si scontra con la sostanziale imprevedibilità dell'ambiente di esecuzione.

Oltre a questo, la mobilità degli agenti riduce fortemente la scalabilità di un sistema di questo tipo: infatti, se la posizione di un

agente può cambiare dinamicamente, gli agenti che vorranno comunicare con lui dovranno poterlo rintracciare altrettanto dinamicamente. Se gli spostamenti avvengono in una rete di dimensioni ridotte, allora si può pensare di memorizzare tutti i movimenti di ogni singolo agente per rintracciarlo, ma questo può diventare troppo costoso al crescere delle dimensioni.

Un ulteriore problema è la gestione della sicurezza. Spesso una comunicazione di questo tipo non prevede alcun controllo da parte di un'entità esterna, e questo rende complesso rinforzare le politiche di sicurezza che si vogliono adottare.

Per superare questi problemi sono stati proposti modelli di coordinazione data-driven, in cui la comunicazione avviene tramite la produzione e il consumo di informazioni su uno spazio di interazione condiviso, che nei modelli ibridi diventa programmabile.

1.2.2. Modelli di coordinazione data-driven: Linda

Linda [ref 5], è il primo modello di coordinazione data-driven proposto, in cui gli agenti interagiscono scambiandosi strutture dati attraverso uno spazio di comunicazione che si comporta come una memoria condivisa.

In questo modello lo spazio di comunicazione è realizzato da uno *spazio di tuple*, un insieme strutturato di relazioni, intese come contenitori di attributi e valori. Su uno spazio di tuple si possono dunque depositare o estrarre informazioni di alto livello senza causare interferenze.

Gli agenti interagiscono scambiandosi tuple logiche, che sono insiemi ordinati di informazioni eventualmente eterogenee. Ad esempio, una tupla può essere rappresentata come un termine logico in stile Prolog: un'entità dotata di un nome e di una lista ordinata di termini, variabili o costanti.

In questo modo gli agenti possono comunicare, sincronizzarsi e cooperare semplicemente leggendo, prelevando e depositando tuple in modo associativo, tramite il meccanismo del pattern match-

ing. Ciò significa che quando un agente vuole prelevare una tupla deve fornire allo spazio di tuple un template, cioè una sorta di tupla avente uno o più attributi variabili, e lo spazio di tuple dovrà quindi restituire una tupla che corrisponda al template, che dunque rappresenta l'insieme dei parametri della richiesta.

In pratica, come è facile intuire, una tupla corrisponde ad un template se:

1. il numero di attributi della tupla è uguale a quello del template;
2. ogni attributo della tupla che corrisponde ad un attributo non variabile del template ha lo stesso valore dell'attributo del template.

Se queste condizioni sono verificate, si dice che la tupla fa matching con il template, e ad ogni attributo variabile del template viene assegnato il valore del corrispondente attributo della tupla.

Il linguaggio di coordinazione associato è dunque assolutamente generale e minimale: lo spazio di tuple fornisce infatti solo tre primitive fondamentali:

- **in**(TT), che consente di leggere una tupla T che faccia matching con il template TT. Se nello spazio sono presenti più tuple idonee ne viene restituita una in modo non deterministico. La tupla fornita viene automaticamente rimossa dallo spazio. Se invece non è presente nessuna tupla che faccia matching con il template, l'agente invocante viene sospeso in attesa.
- **rd**(TT), che ha la stessa semantica della in(), con la differenza che la tupla restituita non viene rimossa dallo spazio.
- **out**(T), che consente di inserire la tupla T nello spazio di tuple, con il vincolo che non possono esistere due tuple identiche nello spazio. Se al momento dell'invocazione c'è già un agente sospeso in attesa e la tupla T fa matching col suo template, tale agente viene risvegliato, e gli viene restituita direttamente la tupla T. Se invece ci sono due o più agenti sospesi in attesa, ne viene risvegliato uno in modo non deterministico, purchè, ovviamente, la tupla T faccia matching con il suo template. È evidente che per questo motivo il meccanismo non garantisce

fairness, ma rende più semplice l'implementazione.

Successive estensioni di questo modello prevedono anche delle varianti non bloccanti di `in()` e `rd()`, rispettivamente **`inp(TT)`** e **`rdp(TT)`**, che forniscono subito un risultato di fallimento se non sono presenti tuple idonee, e l'agente invocante può quindi proseguire la sua esecuzione senza sospendersi in attesa.

Un modello basato su tuple come questo garantisce una netta separazione tra computazione e coordinazione, permettendo di realizzare gli agenti in una architettura più pulita, dato che non è più necessario che gli agenti conoscano tutti i loro interlocutori. Inoltre rende possibile un disaccoppiamento degli agenti sia spaziale che temporale: non è più necessario che due interlocutori siano presenti ed attivi contemporaneamente, poiché le tuple vengono memorizzate nello spazio, quindi il loro consumo può essere differito. In questo caso, si dice che la comunicazione è generativa, nel senso che i dati relativi alla comunicazione sopravvivono anche dopo la fine dell'atto di comunicazione.

Queste vantaggiose caratteristiche rendono un modello come questo particolarmente adatto a realizzare una coordinazione di agenti *mobili*: infatti le operazioni di `in()` e `out()` possono tranquillamente essere effettuate da remoto, e non è necessario l'oneroso tener traccia degli spostamenti di ogni agente, che è invece richiesto da un modello a scambio di messaggi.

1.2.3. Possibili implementazioni del modello

Questo modello, molto promettente, è stato quindi studiato da molti ricercatori in diversi ambiti. In particolare, dato che consente la coordinazione di agenti che possono essere anche remoti, ha attirato l'attenzione di chi si occupa di Reti di Calcolatori e di Sistemi Distribuiti. Infatti, nel tentativo di implementare uno spazio di tuple in un contesto distribuito, emergono alcuni aspetti interessanti, che comportano la necessità di risolvere alcuni problemi. Avere un unico spazio di tuple, ad esempio, rischia di diventare

un pericoloso collo di bottiglia per gli agenti che vi accedono. Per fortuna questo problema può essere risolto dalla scienza delle Reti di Calcolatori, che vede nella replicazione una soluzione consolidata, funzionale e concettualmente semplice per risolvere qualsiasi problema di scalabilità come questo.

Una possibile implementazione consiste quindi in un insieme distribuito di nodi, la cui unione costituisca l'intero spazio di tuple. Questa soluzione rende però complicato garantire il rispetto dei vincoli di funzionamento del modello, come ad esempio l'unicità di una tupla, o il risveglio di un agente sospeso in attesa durante un'operazione di in. Per questo motivo l'insieme di nodi che realizza lo spazio di tuple dovrebbe avere una organizzazione precisa e razionale, poiché ogni operazione di inserimento o prelievo di tuple dovrebbe coinvolgere tutti i nodi dell'insieme. Ad esempio, si potrebbe organizzare lo spazio di tuple in un insieme di nodi ordinati in un anello logico: in questo modo ogni agente può rivolgersi al nodo più "vicino" e cominciare da questo a scandire l'anello alla ricerca di tuple o di richieste pendenti, a seconda che l'operazione richiesta sia una `in()/rd()` o una `out()`. Grazie al non determinismo del modello, la scansione dell'anello può essere interrotta non appena si rintraccia un nodo che contenga quello che si sta cercando: viceversa, se il modello fosse stato deterministico, avrebbe sì garantito *fairness*, ma a prezzo di dover scandire in ogni caso l'intero anello.

Un'altra soluzione prevede un'organizzazione a mesh di nodi: essi vengono disposti in una matrice, consentendo la replicazione delle tuple lungo una direzione. Questa soluzione tenta di ridurre i tempi di risposta limitando il numero di nodi da scandire durante un'operazione, ma è evidente che richiede maggiore memoria, complica la gestione del sistema, e non risolve i problemi della soluzione precedente, ma ne introduce di nuovi.

In alternativa, questo modello potrebbe essere implementato non come un unico spazio di tuple distribuito su più nodi, ma come diversi piccoli spazi di tuple distribuiti su diversi nodi. Una solu-

zione di questo tipo obbliga a specificare lo spazio su cui si vuole effettuare un'operazione, ma consente di avere diverse società di agenti, che possono comunque scambiarsi informazioni tra di loro semplicemente rivolgendosi al nodo opportuno. Questa è la soluzione adottata dall'infrastruttura TuCSon, come illustrato in seguito, nel paragrafo 1.3.

1.2.4. Limiti del modello

Un modello data-driven come quelli appena descritti ha indubbiamente molti vantaggi: in primo luogo la coordinazione è *information-driven*: la sincronizzazione, cooperazione e competizione degli agenti si basa sulle informazioni disponibili nello spazio di tuple condiviso, dove la produzione e consumo di tali informazioni avviene tramite un accesso associativo.

Questo rende i protocolli di interazione semplici ed espressivi, ma presenta anche alcuni svantaggi.

In primo luogo limita la flessibilità: sia la coordinazione tra gli agenti sia l'accesso a risorse sono realizzati dai meccanismi di accesso forniti dallo spazio di dati, definiti una volta per tutte e non modificabili. Di conseguenza, una politica di coordinazione non direttamente supportata dal modello deve essere a carico degli agenti, aumentandone la complessità e riducendone l'autonomia.

In altre parole, rappresentare un'informazione tramite un insieme di tuple rende impossibile separare il modo in cui è rappresentata dal modo in cui viene percepita dagli agenti.

Per fare un esempio, consideriamo il problema dei filosofi mangianti. Se rappresentiamo la situazione corrente con una serie di tuple che indicano la disponibilità di ogni singola posata separatamente (*chop(i)* significa che l'*i*-esima posata è disponibile) si rischia il deadlock, dato che non è possibile consumare due tuple contemporaneamente. Se invece rappresentiamo con una tupla la disponibilità di un paio di posate adiacenti (*chops(2,3)*, ad esempio) evitiamo il deadlock, poiché un filosofo (agente) può acquisi-

re o entrambe le posate di cui ha bisogno o nessuna delle due, ma d'altronde non è possibile garantire la coerenza dell'insieme di tuple dello spazio. Dovremmo infatti garantire che quando venga acquisita la tupla chops(2,3), le tuple chops(1,2) e chops(3,4) non siano più disponibili.

In pratica, il problema dei modelli data-driven sta nel fatto che il comportamento del mezzo di coordinazione è fisso, immutabile. Il comportamento di uno spazio di tuple, dato dalle sue transizioni di stato in risposta agli eventi di comunicazione, è fissato una volta per tutte dal modello, e non può essere adattato in base alle specifiche esigenze dell'applicazione. Può infatti succedere che i meccanismi standard messi a disposizione dallo spazio di tuple non siano sufficienti per supportare direttamente le politiche di coordinazione richieste: di conseguenza, tale onere dovrà essere a carico degli agenti stessi, perdendo così i vantaggi dell'adozione di un modello come questo, poiché impedisce agli agenti di astrarre dai dettagli di coordinazione.

1.2.5. Modelli di coordinazione ibridi: il Centro di Tuple

Per superare i limiti dei modelli data-driven sono successivamente nati i modelli di coordinazione *ibridi*, che uniscono ai vantaggi di un modello data-driven la potenza di un modello control-driven. L'idea alla base di questi modelli è la separazione tra la rappresentazione e la percezione dell'informazione. Questo consente di costruire i protocolli di interazione sull'informazione che gli agenti percepiscono, indipendentemente da come sia rappresentata in tuple, e quindi scaricare tutte le regole di coordinazione sul mezzo di coordinazione stesso.

Tornando all'esempio dei filosofi mangianti, la soluzione consiste nel continuare a rappresentare le singole posate separatamente, ma permettendo agli agenti di percepirle a coppie: in questo modo i filosofi potranno acquisire due posate con una singola operazione

di $\text{in}(\text{chops}(i,j))$, e sarà il mezzo di coordinazione a rimuovere dallo spazio le due tuple $\text{chop}(i)$ e $\text{chop}(j)$ in modo atomico.

In questi modelli, dunque, lo spazio di interazione condiviso dal gruppo di agenti diventa un'entità attiva, capace di associare agli eventi di comunicazione opportune reazioni che vengono scatenate ogni volta che si verifica l'evento stesso. In questo modo lo spazio di tuple evolve, diventando quello che viene definito un Centro di Tuple [ref 10]. L'interfaccia verso gli agenti non cambia, ma può cambiare il comportamento del centro di tuple a fronte di un evento di comunicazione, a seconda di come venga programmato il centro di tuple stesso.

Formalmente un centro di tuple è uno spazio di tuple con una specifica di comportamento, espressa in termini di reazioni ad eventi di comunicazione. Una reazione è definita come un insieme di attività computazionali associate ad un particolare evento.

Una reazione può sia vedere e modificare lo stato del centro di tuple, aggiungendo o rimuovendo tuple, sia accedere a tutte le informazioni legate all'evento di comunicazione verificatosi, come l'agente che l'ha scatenato, l'operazione richiesta, la tupla coinvolta e così via.

Ogni evento di comunicazione, dopo essere stato servito come in uno spazio di tuple normale, può scatenare più reazioni, che vengono eseguite localmente al centro di tuple prima di servire qualsiasi altro evento verificatosi. In tal modo gli agenti vedono i risultati di tutte le reazioni scatenate come una transizione atomica del centro di tuple, che può essere complessa quanto serve. Questo consente di mantenere separata la percezione degli agenti, che coincide con un normale spazio di tuple, dallo stato effettivo del centro di tuple, e di incapsulare così in quest'ultimo tutte le leggi di coordinazione.

Per poter programmare un centro di tuple è necessario che le reazioni vengano espresse in un opportuno linguaggio di specifica sufficientemente espressivo e potente, mediante il quale il centro di tuple possa essere programmato dinamicamente da opportuni

agenti di metalivello. Questo linguaggio di specifica deve poter non solo descrivere la computazione della reazione, ma anche associare la reazione ad un evento. Un linguaggio di specifica di comportamento molto interessante è ReSpecT [ref 9].

1.2.6. Programmare un centro di tuple: il linguaggio ReSpecT

Questo linguaggio per la specifica del comportamento di un centro di tuple si basa sulla logica del primo ordine. Una tupla è quindi un fatto *ground*, cioè un predicato con tutti i termini istanziati ad un valore costante, mentre qualsiasi clausola unitaria è un ammissibile template di tuple: in questo modo il meccanismo di matching è l'unificazione.

Un centro di tuple ReSpecT appare quindi come uno spazio di tuple logiche, accessibile tramite le normali operazioni di out, in, rd, inp e rdp. Dal punto di vista degli agenti, queste operazioni hanno una semantica che coincide esattamente con quella di un modello come Linda, ma in realtà, a livello applicativo, nascondono tutta una serie di sotto-operazioni che vengono svolte dal centro di tuple in risposta all'evento di invocazione della primitiva corrente: le reazioni, appunto.

Le reazioni in ReSpecT sono esse stesse definite in termini di tuple logiche. Il centro di tuple viene dunque programmato mediante l'inserimento e rimozione di particolari tuple logiche, dette tuple di specifica, che sono del tipo `reaction(Op, R)` e che associano all'evento di invocazione della primitiva di comunicazione Op la reazione R.

Una reazione è definita come una sequenza di obbiettivi (*reaction goals*), i quali possono accedere a tutte le proprietà dell'evento di comunicazione occorso e manipolare l'insieme di tuple disponibili correntemente nel centro di tuple. Anche queste modifiche allo spazio di tuple possono scatenare ulteriori reazioni.

Una reazione R, dunque, è descrivibile nella sintassi della logica del primo ordine come una lista di particolari predicati, i quali

hanno una semantica definita dal linguaggio ReSpecT. Alcuni di questi predicati permettono alla reazione l'accesso allo spazio di tuple: sono quindi simili alle primitive di comunicazione, e possono scatenare a loro volta ulteriori reazioni. Altri predicati servono invece per l'accesso alle proprietà dell'evento di comunicazione, sempre in modo associativo. Di seguito sono elencati in dettaglio i principali predicati ReSpecT.

- Accesso e modifica allo spazio di tuple:
 - `out_r(T)` ha la stessa semantica della `out`
 - `in_r(TT)` ha la stessa semantica della `inp`
 - `rd_r(TT)` ha la stessa semantica della `rdp`
 - `no_r(TT)` è duale alla `rd_r`: ha successo se non ci sono tuple adatte al template
- Accesso alle proprietà dell'evento di comunicazione:
 - `current_tuple(T)` ha successo se `T` fa matching con la tupla coinvolta nell'evento
 - `current_agent(A)` ha successo se `A` fa matching con l'agente che ha scatenato l'evento
 - `current_op(Op)` ha successo se `Op` fa matching con l'operazione corrente
 - `current_tc(N)` ha successo se `N` indica il centro di tuple dove avviene la reazione

Oltre a questi, per aumentare la potenza espressiva ed avere reazioni più dettagliate, vengono introdotti alcuni predicati che derivano da un'analisi più dettagliata delle operazioni possibili. In particolare, si nota come ogni operazione di accesso allo spazio di tuple, eccetto la `out`, avvenga in due fasi: la prima, detta fase *pre*, è una fase di query, cioè in cui si effettua una ricerca di tuple; la seconda, detta fase *post*, è invece la conclusione della primitiva, in cui viene restituito il risultato. Per poter sfruttare anche questa distinzione, ReSpecT introduce i seguenti predicati:

- `pre` ha successo nella fase *pre* di qualsiasi operazione

- `post` ha successo nella fase post di qualsiasi operazione
- `success` ha successo nella fase pre di qualsiasi operazione, e in quella post di ogni operazione che va a buon fine
- `failure` ha successo nella fase post di ogni operazione che fallisce

Durante l'esecuzione di una reazione, tutti i predicati di cui è composta la reazione stessa, ciascuno dei quali rappresenta un *reaction goal*, vengono valutati in sequenza riportando un risultato di successo o di fallimento. Non appena si verifica il fallimento di uno di essi l'intera reazione fallisce, mentre ha successo solo se tutti i suoi predicati terminano con successo, e quindi tutti i goal sono soddisfatti. La semantica di una reazione però deve essere transazionale: se ha successo, tutte le modifiche che apporta allo stato del centro di tuple devono essere salvate atomicamente, e se fallisce non deve apportarne nessuna. Inoltre ogni reazione scatenata da un evento deve essere eseguita prima del servizio di un evento successivo: è questo che consente di inserire un nuovo comportamento osservabile in un centro di tuple, in modo trasparente agli agenti.

Infine, è stato dimostrato che ReSpecT è Turing-equivalente, e quindi permette di incapsulare in un centro di tuple qualsiasi legge di coordinazione computabile.

1.3. INFRASTRUTTURE DI COORDINAZIONE: TUCSON

In seguito alle precedenti osservazioni, è nata una infrastruttura di coordinazione per agenti basata su Internet che viene chiamata TuCSoN [ref 3,16,17], nome che significa Tuple Centres Spread over Network. TuCSoN è realizzato in Java, e può quindi avvantaggiarsi della grande portabilità che caratterizza questa tecnologia, prestandosi a contesti fortemente distribuiti. Comprende uno spazio di coordinazione, che può essere visto sia come globale che

come collezione di spazi locali, e un mezzo di coordinazione, il cui comportamento può essere definito per supportare leggi di coordinazione globale.

Lo spazio di coordinazione di TuCSoN è realizzato da una molteplicità di centri di tuple sparsi su vari nodi Internet: ogni centro di tuple è associato ad un nodo, ed è individuato da un identificatore univoco locale, come indicato in figura.

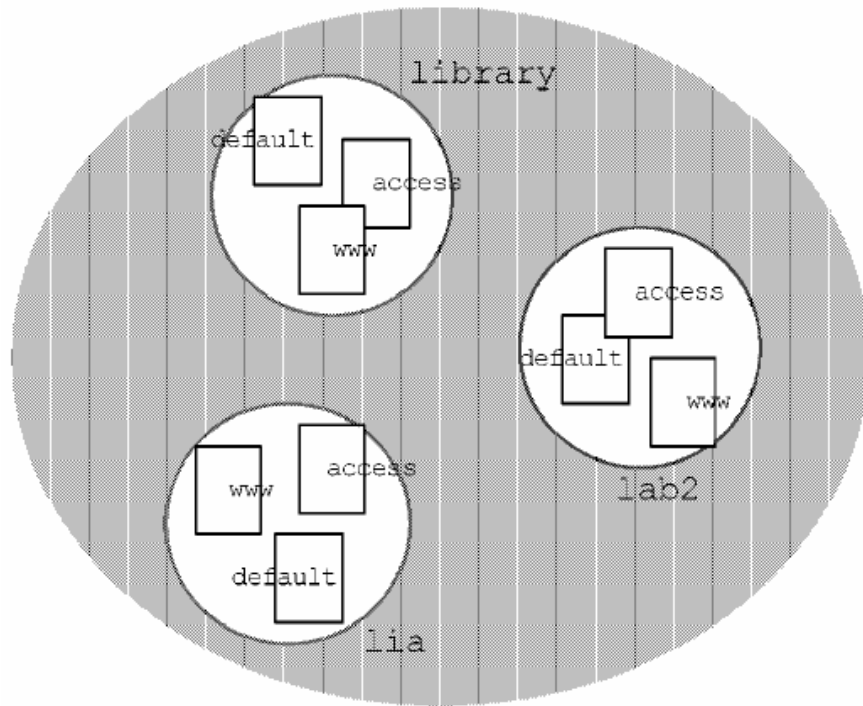


Fig. 1.1 – Nodi e Centri di Tuple in TuCSoN

Di conseguenza, ogni centro di tuple è visibile come un servizio Internet, e può essere acceduto sia tramite l'indirizzo Internet completo, nella forma $\langle nomeLocale \rangle @ \langle nodo \rangle$, sia tramite il suo nome locale, sufficiente all'interno del contesto del suo nodo. Questo rende possibile vedere lo spazio di coordinazione sia come unico globale, formato da un insieme di centri di tuple univocamente distinguibili, sia come collezione di spazi di interazione locali, poiché centri di tuple su nodi diversi possono avere lo stesso nome locale. Questo semplifica i protocolli di interazione, soprat-

tutto per gli agenti mobili: vediamo come.

1.3.1. Il linguaggio di coordinazione

I centri di tuple in TuCSoN offrono agli agenti le stesse primitive di comunicazione descritte per Linda, vale a dire *out*, *in*, *rd*, *inp* e *rdp*. Tuttavia, esistendo molti centri di tuple, per una corretta invocazione di una operazione occorre anche indicare il centro di tuple su cui si desidera effettuarla, specificando il suo nome assoluto oppure quello relativo: il formato di invocazione di una generica operazione TuCSoN sarà dunque:

$$TupleCentre?op(Tupla).$$

È interessante notare che il fatto di poter specificare sia il nome assoluto che il nome relativo di un centro di tuple permette di realizzare due tipi diversi di invocazione: invocazione di rete ed invocazione locale.

L'invocazione di rete, nella forma *nome@nodo?op(Tupla)*, consente agli agenti di raggiungere un particolare centro di tuple da qualunque parte della rete essi si trovino, identificando univocamente sempre solo quello. La forma *nome?op(Tupla)*, viceversa, denota l'invocazione locale, e consente all'agente di reperire l'implementazione locale al nodo corrente del centro di tuple. In questo modo l'agente può interagire con l'ambiente locale dell'host su cui si trova senza dover conoscerne la posizione assoluta: è facile intuire che questo è molto utile agli agenti mobili, i quali migrano all'interno della rete per eseguire l'elaborazione all'interno del contesto più adatto, cioè più vicino alle risorse di cui hanno bisogno. In questo modo essi possono operare allo stesso modo su qualunque nodo si spostino.

La disponibilità di entrambe le modalità di invocazione, dunque, rinforza la separazione tra i problemi relativi alla rete (come la migrazione di un agente) e quelli relativi alla computazione locale (come l'accesso alle risorse locali), riducendo la complessità di un

agente.

1.3.2. Il mezzo di coordinazione

Come ho già detto, il principale limite dei modelli Linda-like è il comportamento fisso dello spazio di tuple. TuCSoN supera il problema adottando come mezzo di comunicazione il Centro di Tuple, il cui comportamento è programmabile in un linguaggio come ReSpecT, ma non solo.

La grande flessibilità e potenza espressiva di TuCSoN, che lo rende idoneo a supportare sistemi multiagente molto complessi, deriva dal fatto che ogni nodo può contenere diversi centri di tuple, ciascuno dei quali può essere dedicato ad una parte distinta dell'applicazione. Questo permette di rendere il sistema estremamente modulare e flessibile, facilitando information hiding e sicurezza, ma porta anche un altro importante vantaggio: ogni singolo centro di tuple può essere programmato separatamente, indipendentemente dagli altri, in accordo alle sue specifiche leggi di coordinazione. È evidente che questo mette a disposizione dei progettisti una granularità molto più fine per l'implementazione delle politiche di coordinazione globali.

Ogni centro di tuple in TuCSoN adotta il modello di ReSpecT per la definizione dei linguaggi di comunicazione e di specifica di comportamento. Questa scelta è giustificata dal fatto che tale modello permette una comunicazione con l'espressività della logica del primo ordine, e consente di vedere l'astrazione di comunicazione come contenitore sia di messaggi che di teoria di interazione.

1.4. UN TIPICO PROBLEMA DI COORDINAZIONE: LA GESTIONE DEI WORKFLOW

I sistemi multiagente sono astrazioni molto efficaci quando si ha a che fare con ambienti complessi, aperti, dinamici ed imprevedibili. Un esempio di sistemi di questo tipo sono i cosiddetti *Virtual Enterprise* [ref 15], che sono aggregazioni temporanee di imprese autonome ed eterogenee connesse in rete, che servono a fornire la flessibilità e adattabilità necessaria a sostenere i continui cambiamenti che caratterizzano l'apertura degli scenari di business. Si tratta dunque di integrare attività diverse e distribuite, scenario in cui uno strumento come TuCSon può rivelarsi un aiuto particolarmente efficace.

Per costruire un VE è necessario affrontare due grandi problemi: anzitutto l'integrazione dei sistemi e infrastrutture esistenti, problema reso complicato dalla loro diversità di informazioni (sintassi, semantica ed ontologie), di tecnologie (software ed hardware) e di processi (natura delle entità coinvolte, linguaggi e protocolli di interazione); il secondo problema è la gestione del *workflow*, ossia la specifica, l'esecuzione ed il monitoraggio del processo di business del VE specifico.

La complessità di sistemi come questo deriva dal fatto che per raggiungere gli obiettivi del sistema globale è necessario gestire correttamente un grande numero di dipendenze tra le attività autonome ed eterogenee. Per fare un esempio, basti pensare a quanto è complicato far condividere e scambiare risorse ed informazioni tra entità eterogenee: è uno dei problemi di ingegneria del software attualmente più sentiti. È evidente che un sistema di questo tipo può essere facilmente modellato come un sistema multi-agente, nel quale gli agenti svolgono le loro attività autonomamente, ed è quindi altrettanto evidente che la modellazione di un VE può essere vista come un problema di coordinazione.

In questo contesto, il problema principale è la gestione dei workflow. Un workflow è un insieme di attività che devono essere eseguite in un dato ordine e che coinvolgono molte entità in un ambiente distribuito. Un sistema che lo gestisca dovrà fare in modo da minimizzare i tempi di esecuzione e massimizzare

l'efficienza dell'uso delle risorse, ma dovrà anche essere flessibile ed adattabile per sostenere i frequenti cambiamenti a cui il workflow può essere soggetto.

Tipicamente, questo problema viene affrontato separando entità coordinatrici, dette *workflow engines*, dalle entità partecipanti, o *workflow participants*. In questo caso, in accordo alla coordinazione oggettiva, è possibile incorporare nei workflow engines tutte le leggi di coordinazione, sia per abilitare l'interazione che per rendere effettive le politiche richieste. Così le regole del workflow possono essere poste al di fuori degli agenti, i quali possono mantenere la loro autonomia.

In pratica, le suddette osservazioni suggeriscono la necessità di un'infrastruttura adeguata, che fornisca servizi general purpose, come la coordinazione, e che sia configurabile in base alle specifiche necessità. Questa infrastruttura dovrà permettere sia l'integrazione delle risorse eterogenee fornite dai partecipanti, sia l'esecuzione del processo di business complessivo.

Per quanto riguarda la prima caratteristica, l'infrastruttura dovrà provvedere all'omogeneizzazione concettuale delle risorse, informazioni ed attività. In altre parole, dovrà agire come una colla, supportando interazioni consistenti tra processi e servizi condivisi, e minimizzando l'impatto sulla struttura dei partecipanti. Oltre a questo è opportuno che un'infrastruttura di questo genere supporti astrazioni e meccanismi per la sicurezza. Per quanto riguarda la seconda caratteristica, l'infrastruttura dovrà governare le dipendenze che caratterizzano l'esecuzione del processo di business, come assegnamento, prerequisiti e vincoli temporali delle attività. Per fare ciò, è necessario un sistema di gestione dei workflow, appunto.

In sostanza, un'infrastruttura per sistemi di gestione di workflow dovrà supportare:

- comunicazione senza protocolli punto a punto che richiedano accordi,
- partecipazione flessibile e scalabile, possibile da qualsiasi di-

- spositivo e senza necessità di connessioni dedicate o di informazioni a priori,
- operazioni non in linea, affinché i partecipanti possano essere mobili, indipendenti dalla locazione e non frequentemente connessi,
 - tracciabilità dello stato del processo di business

Per questo l'infrastruttura deve fornire una chiara separazione tra i partecipanti e i coordinatori, distaccando l'attività di coordinazione dalle attività da coordinare, e le regole del workflow devono poter essere descritte esplicitamente.

Un'infrastruttura come TuCSoN soddisfa perfettamente tutti i requisiti di un'infrastruttura per la gestione di workflow. Esso infatti consente una comunicazione fortemente disaccoppiata in un contesto distribuito, favorisce l'autonomia e la mobilità degli agenti partecipanti, e consente di incapsulare (e modificare dinamicamente) specifiche leggi di coordinazione, definibili chiaramente in un linguaggio dichiarativo molto espressivo. È dunque possibile mappare un workflow engine in un centro di tuple, che governa l'interazione dei partecipanti modellati come agenti.

CAPITOLO 2

METODOLOGIA DI PROGETTO AGENT-ORIENTED

2.1. L'IMPORTANZA DI UNA METODOLOGIA AGENT-ORIENTED

Il modello ad agenti fornisce utili astrazioni che consentono di trattare con maggiore efficacia alcuni problemi tipici dei sistemi software complessi, come la distribuzione, eterogeneità, e decentralizzazione del controllo. Per poter sfruttare queste astrazioni nel progetto di un sistema multi-agente è quindi necessaria una metodologia di progetto opportuna. A questo scopo sono state proposte alcune metodologie agent-oriented che invitano il progettista ad affrontare un problema ragionando in termini di agenti autonomi, piuttosto che in termini di attributi e responsabilità, come invece suggeriscono le metodologie object-oriented.

Le prime metodologie proposte, tuttavia, si limitavano a suggerire ai progettisti un modo di concepire gli agenti singolarmente, mediante opportuni linguaggi ad agenti, concentrandosi maggiormente sugli aspetti intra-agente: solo successivamente gli agenti così costruiti venivano arricchiti della capacità di comunicazione, mediante l'adozione di una qualche infrastruttura.

Diversamente, la metodologia SODA [ref 7], invita a considerare gli aspetti inter-agente già dalle prime fasi di analisi. Gli agenti non sono componenti software che prima vengono costruiti e poi combinati, ma sono entità consapevoli di vivere in un contesto sociale col quale devono interagire. Una società di agenti esibisce quindi un comportamento globale che non può essere ricondotto

alla mera somma dei comportamenti individuali: essa è un'entità di prima classe, caratterizzata da precise regole sociali, volte al conseguimento degli obbiettivi globali. Per questo motivo la metodologia SODA propone la separazione tra problemi sociali ed individuali sin dalle prime fasi dell'analisi.

Gli agenti sono entità individuali con abilità sociali, interagiscono con il loro ambiente per conseguire i propri obbiettivi, e il loro comportamento è spesso incomprensibile se considerato al di fuori della loro struttura sociale. Allo stesso modo, il comportamento globale della società non può essere generalmente espresso in termini del comportamento degli agenti che la compongono: sono le regole sociali a guidarne il comportamento.

2.2. LA METODOLOGIA SODA

SODA (*Society in Open and Distributed Agent spaces*) [ref 7] è una metodologia per l'analisi e il progetto di applicazioni basate su Internet come sistemi multi-agente. L'obiettivo di SODA è definire un framework concettuale e coerente ed una procedura di ingegnerizzazione del software comprensibile che serva per l'analisi e il progetto di agenti individuali, società di agenti, e ambienti.

SODA non si occupa di problemi intra-agente, ma si concentra invece su problemi inter-agente, società ed infrastrutture. Progettare un sistema multi-agente con SODA significa definire gli agenti in termini del loro comportamento osservabile e del loro ruolo in sistemi multi-agente, mentre i singoli agenti possono essere progettati nel modo più opportuno.

La fase di progetto di SODA è profondamente legata alla nozione di *modello di coordinazione*. In particolare, i modelli e i linguaggi di coordinazione forniscono le astrazioni ed i meccanismi richiesti per progettare le società di agenti: le regole sociali sono

progettate come leggi di coordinazione incorporate nel mezzo di coordinazione, e le infrastrutture sociali sono costruite sui sistemi di coordinazione.

SODA si articola in due fasi: una prima fase di Analisi cerca di fornire un modello completo e funzionale del sistema complessivo, ed una seconda fase di Progetto cerca di spiegare come tale modello possa essere realizzato.

2.2.1. La fase di Analisi

Questa prima fase ha l'obiettivo di fornire al progettista una descrizione formale o semi formale della struttura complessiva e dell'organizzazione dell'applicazione. In questa fase si studia l'ambiente in cui vivono gli agenti, che viene modellato in termini di risorse e servizi disponibili, e l'interazione degli agenti stessi, che viene modellata in termini di informazioni richieste e fornite dagli agenti e dalle risorse.

La fase di analisi di SODA si concentra quindi su tre differenti modelli: il modello dei *ruoli*, il modello delle *risorse*, e il modello di *interazione*.

Questi tre modelli, nonostante siano concettualmente distinti, sono ovviamente prettamente correlati e devono essere definiti in modo consistente.

2.2.1.1. Modello dei ruoli

Gli obiettivi dell'applicazione sono modellati in termini di attività, che dovranno essere associate a ruoli e gruppi.

Le attività sono espresse in termini delle responsabilità che coinvolgono, delle competenze che richiedono e delle risorse da cui dipendono, e sono classificate come individuali o sociali. Tipicamente le attività sociali sono quelle che richiedono molte diverse competenze, e l'accesso a diverse risorse, mentre quelle individuali richiedono competenze e risorse più limitate.

Ogni attività individuale è associata ad un ruolo, che di conse-

guenza è definito in termini delle responsabilità che porta. Analogamente le attività sociali sono assegnate a gruppi, che sono così definiti non solo in base ai ruoli dei partecipanti, ma anche in base alle responsabilità relative alla loro attività sociale.

2.2.1.2. Modello delle Risorse

L'ambiente del sistema multi-agente fornisce agli agenti alcuni servizi per permettere loro il corretto svolgimento delle loro attività. Ogni servizio è associato ad una risorsa astratta, che viene caratterizzata da una o più modalità di accesso, a seconda dei permessi di accesso presentati dagli agenti che la richiedono.

2.2.1.3. Modello di Interazione

L'interazione tra ruoli, gruppi e risorse è modellata in termini di regole e protocolli di interazione, espressi come l'insieme di informazioni richieste e fornite dai ruoli e dalle risorse.

Un protocollo di interazione associato ad un ruolo è definito in termini delle informazioni richieste e fornite dal ruolo per il compimento della sua attività individuale. Un protocollo di interazione associato ad una risorsa è definito in termini delle informazioni richieste per invocare il servizio fornito dalla risorsa stessa, e dalle informazioni restituite quando il servizio invocato viene portato a termine. Una regola di interazione è invece associata a un gruppo, e governa le interazioni tra i ruoli sociali e le risorse in modo che il gruppo possa perseguire le sue attività sociali.

Questi protocolli però non sono esplicitamente legati tra di loro: ciascuno di essi è infatti definito specificando solo le informazioni che richiede, non chi le deve fornire. È quindi necessario assicurarsi che per ogni informazione richiesta da ogni protocollo ci sia almeno un'entità nel sistema che abbia il compito di fornirla.

2.2.2. La fase di Progetto

Questa seconda fase ha il compito di esprimere i modelli astratti

risultanti dalla fase di analisi in termini di astrazioni che possano essere mappate uno a uno sugli attuali componenti del sistema sviluppato.

Anche questa fase si basa su tre modelli strettamente correlati: modello degli *Agenti*, della *Società*, e dell'*Ambiente*.

2.2.2.1. Modello degli Agenti

I ruoli individuali e sociali vengono mappati su classi di agenti. Una classe di agente è definita come un insieme di uno o più ruoli, sia individuali che sociali. Una classe Agente è quindi caratterizzata dalle attività, dall'insieme dei permessi e dai protocolli di interazione associati ai suoi ruoli. Le classi di agenti possono inoltre essere caratterizzate in termini di altre caratteristiche: la loro cardinalità (il numero di agenti della classe), la loro locazione (fissa per agenti statici o variabile per agenti mobili), e la loro sorgente (interna o esterna dal sistema).

Il progetto degli agenti come classi dovrebbe tener conto di tutte le specifiche che provengono dalla fase di analisi.

Il risultato di questa fase è il comportamento osservabile di ogni agente, espresso in termini di tutte le interazioni con l'ambiente che lo circonda. Tale comportamento è definito dai protocolli di interazione, delimitato dall'insieme dei permessi, e finalizzato al conseguimento dell'attività dell'agente.

2.2.2.2. Modello della Società

I gruppi sono mappati su società di agenti, e vengono progettati e organizzati attorno ad astrazioni di coordinazione. Una società di agenti è dunque caratterizzata dalle attività sociali, dall'insieme dei permessi, dai ruoli sociali partecipanti, e dalle regole di interazione associate al gruppo.

Il problema principale di questo modello è come progettare le regole di interazione in modo che le società conseguano le loro attività sociali: questo significa principalmente gestire le interazioni dei partecipanti, ed è quindi un problema di coordinazione. Per

questo motivo il primo punto del progetto della società degli agenti è la scelta del modello di coordinazione più idoneo alle necessità: la società viene poi progettata attorno ad un mezzo di coordinazione che incorpora le regole sociali del suo gruppo in termini di regole di coordinazione.

Il comportamento del mezzo di coordinazione preferito, unito al comportamento degli agenti che giocano un ruolo sociale ed interagiscono attraverso tale mezzo, fa sì che la società di agenti svolga interamente le sue attività sociali. Questo permette alle società di agenti di essere progettate come entità di prima classe.

2.2.2.3. Modello dell'Ambiente

Le risorse sono mappate in classi di infrastruttura. Una classe di infrastruttura è caratterizzata dai servizi, dai modi di accesso, dai permessi assegnati ai ruoli e gruppi, ed ai protocolli di interazione associati alle risorse. Le classi di infrastruttura possono inoltre essere caratterizzate in termini di altre proprietà: la loro cardinalità (numero dei componenti di infrastruttura che appartengono alla classe), la loro locazione, e il loro proprietario, che potrebbe anche non essere lo stesso del sistema di agenti, data l'assunzione del controllo decentralizzato.

Il risultato di questo modello è l'insieme dei servizi forniti da ogni componente dell'infrastruttura, con le sue interfacce, che risultano dai protocolli di interazione associati, mentre i servizi, come i singoli agenti, possono essere progettati singolarmente nel modo più opportuno.

CAPITOLO 3

ANALISI DEL SISTEMA IN ESAME

3.1. L'APPLICAZIONE

Come caso di studio, si vuole sviluppare come sistema multi-agente un'applicazione web-based per la coordinazione di attività più o meno articolate e complesse, che possono coinvolgere siti molteplici e servizi web. Sarà quindi necessario esplorare l'integrazione di infrastrutture di coordinazione per sistemi multi-agente (come per esempio TuCSoN) e tecnologie Web lato server. Come contesto applicativo si è scelto lo sviluppo del sito del corso di Fondamenti di Informatica L-B, che include la coordinazione di varie attività e l'interazione di persone in ruoli diversi.

In particolare, l'applicazione dovrà consentire agli studenti di partecipare attivamente al corso, sottoponendo al docente esercitazioni e progetti che dovranno essere valutati. Oltre a questo, è necessario prevedere la possibilità di sostenere esami via Internet, predisponendo appositamente un'area dedicata all'interno del sito. Inoltre, l'applicazione dovrà consentire agli studenti di scaricare il materiale didattico reso disponibile dal docente, leggere tutte le informazioni e le FAQ che vengono pubblicate, e partecipare ad un forum del corso.

Analogamente, il docente dovrà poter pubblicare materiale didattico ed informazioni varie, oltre che visionare e correggere tutte le esercitazioni ed i progetti ricevuti, per poter poi esprimere un giudizio in merito.

3.2. ANALISI DEI REQUISITI

Per formalizzare meglio le specifiche, che normalmente sono descritte in maniera piuttosto vaga, è opportuno svolgere una analisi preventiva ed individuare i possibili casi d'uso, per poter descrivere il funzionamento del sistema più dettagliatamente.

Dalle specifiche si possono già individuare gli attori principali del sistema, rappresentati in relazione gerarchica nel diagramma seguente.

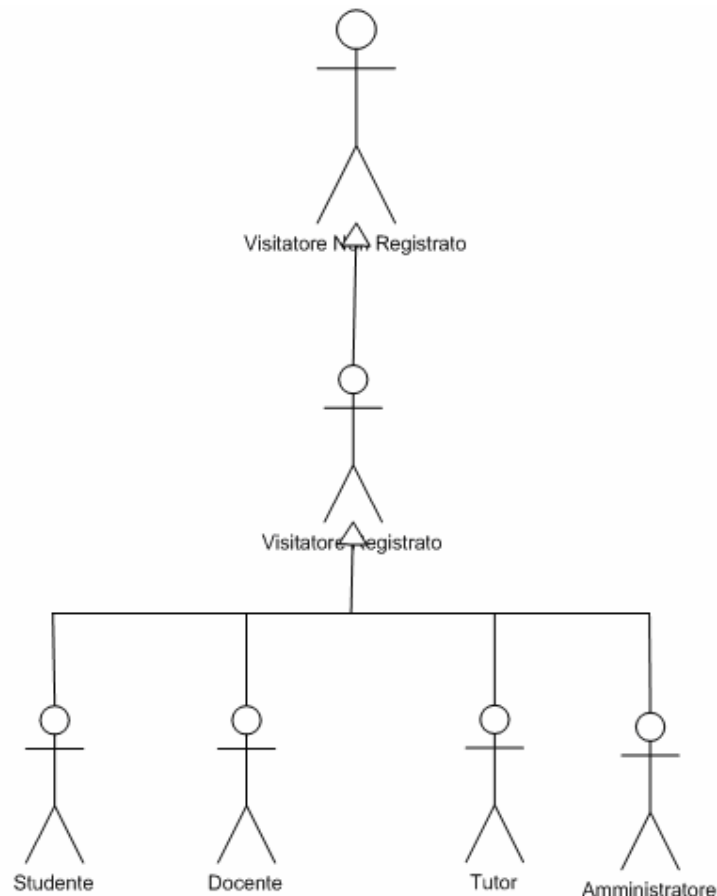


Fig. 3.1 – Attori principali

Alcune informazioni del sito, infatti, possono essere di dominio pubblico, e quindi accessibili da un generico utente Internet, che possiamo chiamare Visitatore Non Registrato.

Le informazioni riservate del sito sono invece accessibili solo da utenti che possano identificarsi e dimostrare di essere autorizzati all'accesso. Questo tipo di utente, chiamato Visitatore Registrato, può a sua volta distinguersi in diverse categorie a seconda dei permessi di accesso: rispettivamente, Studente, Docente, Tutor o Amministratore.

Vediamo dunque più in dettaglio quello che ciascun utente è autorizzato a fare.

- *Docente* : ha la visione più completa del sito, può:
 - inserire il materiale relativo alle lezioni
 - controllare che gli studenti inviano il materiale delle esercitazioni nella forma prestabilita e segnalare eventuali problemi.
 - controllare che gli studenti non abbiano copiato le esercitazioni
 - controllare quali studenti non inviano esercitazioni
 - testare le applicazioni ricevute
 - commentare le proposte di progetto inviate dagli studenti
 - scrivere le domande per l'esame
 - correggere l'esame
 - leggere il forum
 - inserire delle FAQ
 - tenere aggiornato il “diario delle lezioni”, una tabella resoconto che riporta data ed argomento delle lezioni svolte.

Obbiettivo: fornire tutto il materiale e le informazioni utili agli studenti ed ai visitatori non registrati

Permessi: accesso in lettura e scrittura a tutte le parti del sito.

- *Studente* : un visitatore che durante il login si identifica come studente del corso, può:

- scaricare materiale reso disponibile dal docente
- inviare esercitazioni in formato opportuno
- controllare lo stato di correzione delle esercitazioni, cioè verificare se siano già state corrette oppure no
- ove consentito, inviare versioni parziali del progetto che sta svolgendo, e chiederne una valutazione al docente
- inviare versione definitiva del progetto
- scaricare le domande durante l'esame
- salvare i risultati intermedi
- inviare la prova d'esame una volta terminata
- inviare messaggi nel forum

Permessi:

- lettura: accesso a tutto il sito, escluse le parti riguardanti gli altri studenti e le domande d'esame
- scrittura: messaggi del forum, risposte a domande dell'esame

Obbiettivi: trovare tutte le informazioni necessarie per poter superare l'esame

- *Visitatore non registrato* : un internauta che accede al sito senza identificarsi: può essere interessato alle informazioni generali del sito.

Ha un ruolo passivo di sola lettura di informazioni non protette.

Permessi: lettura delle informazioni pubbliche del sito del corso

Obbiettivi: trovare informazioni di suo interesse.

- *Tutor* : deve poter inserire materiale relativo alle esercitazioni e lezioni che svolge, partecipare al forum ed aiutare il docente nella correzione delle esercitazioni, dei progetti e degli esami.

Permessi:

- lettura di tutto il sito
- pubblicazione materiale relativo alle esercitazioni e lezioni svolte

- scrittura: fornire valutazione delle esercitazioni, dei progetti e degli esami

- *Amministratore* : entità referenziata che può controllare lo stato delle varie attività

3.3. CASI D'USO

Vediamo ora in dettaglio i diagrammi e i relativi scenari dei casi d'uso associati ad ogni utente

3.3.1. Casi d'uso dell'attore Docente

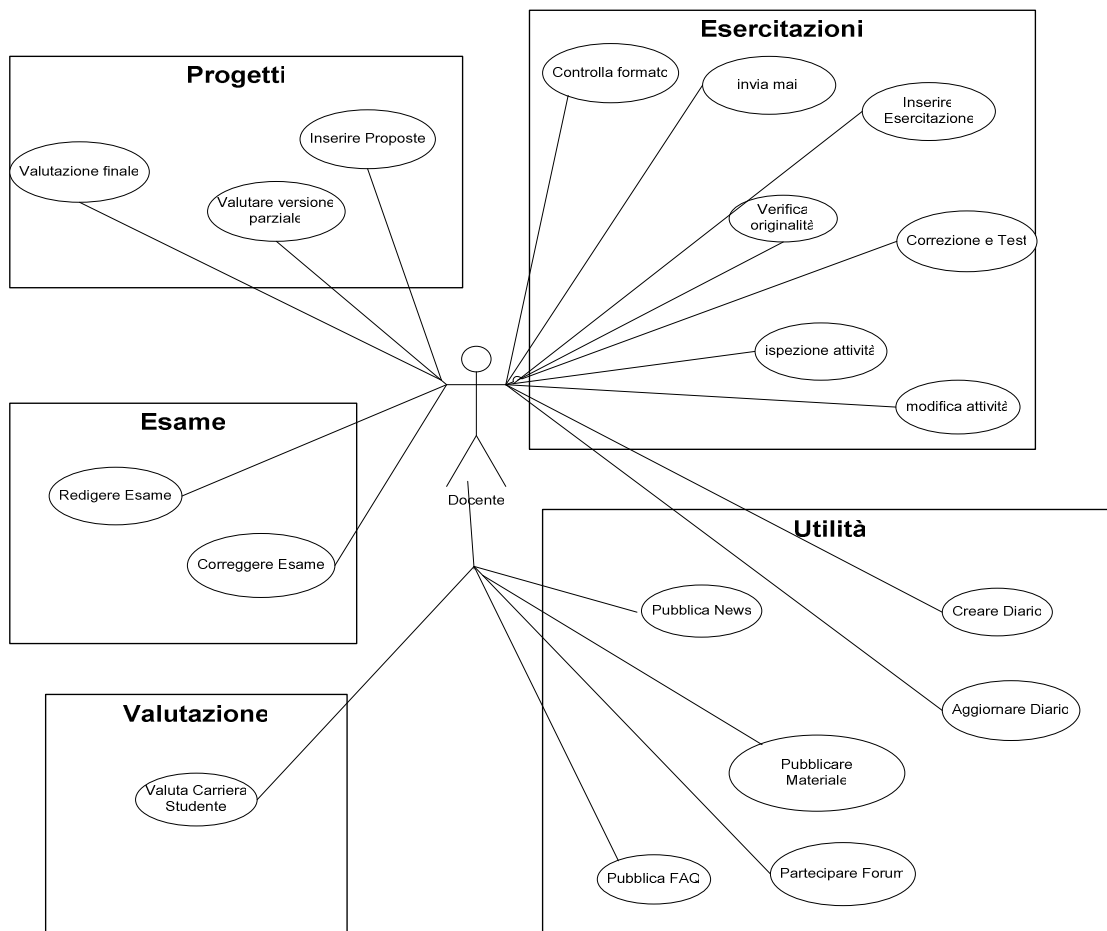


Fig. 3.2 – Casi d'uso Docente

3.3.1.1. Caso d'uso Login:

Attori : Visitatore registrato, Sistema

Obbiettivo : autenticare l'utente corrente

Frequenza : alta

Condizione di successo : l'utente riesce ad accedere

Condizione di fallimento : l'utente non riesce ad accedere al sistema

Scenario Principale

1. L'utente si reca nell'area di login per accedere al Sistema ed immette username e password
2. Il Sistema controlla i dati inseriti
3. Il Sistema assegna all'utente i privilegi di accesso legati ai suoi dati
4. Il Sistema presenta all'utente la siteview ad esso associata

Scenario di fallimento:

- 3.a il Sistema non riconosce l'utente e non gli concede di entrare

3.3.1.2. Caso d'uso Creare Diario:

Attori : Docente, Sistema

Obbiettivo : creare il diario delle lezioni

Frequenza : bassa

Condizione di successo : il Docente riesce a creare il diario

Condizione di fallimento : il Docente non riesce a creare il diario

Scenario Principale

1. Il Docente si reca nell'area relativa al diario delle lezioni.
2. Il Docente chiede al Sistema di creare il Diario per uno specifico corso
3. Il Sistema crea il diario
4. Il Sistema comunica al docente che la creazione è avvenuta con successo.

Scenario di fallimento:

- 4a. Il Sistema comunica al docente che si sono verificati problemi e non è stato possibile creare il diario

3.3.1.3. Caso d'uso Aggiorna Diario:

Attori : Docente, Sistema

Obbiettivo : aggiornare il diario delle lezioni

Frequenza : media

Condizione di successo : il Docente riesce ad aggiornare il diario

Condizione di fallimento : il Docente non riesce ad aggiornare il diario

Scenario Principale

1. Il Docente si reca nell'area relativa al diario delle lezioni.
2. Il Docente chiede al Sistema di aggiornare il Diario del corso scelto
3. Il Sistema mostra al Docente tutte le informazioni immesse sino ad ora e si predispone per l'aggiornamento
4. Il Docente aggiorna (o modifica) il Diario inserendo gli argomenti relativi alla lezione (o alle lezioni) svolta
5. Il Docente chiede al Sistema di Salvare i dati immessi
6. Il Sistema comunica al docente che l'operazione è avvenuta con successo.

Scenario di fallimento:

- 6a. Il Sistema comunica al Docente che si sono verificati problemi e non è stato possibile aggiornare il diario.

3.3.1.4. Caso d'uso Pubblica News:

Attori : Docente, Sistema

Obbiettivo : pubblicare nuove notizie di interesse

Frequenza : media

Condizione di successo : il Docente riesce a pubblicare le notizie

Condizione di fallimento : il Docente non riesce a pubblicare le notizie

Scenario Principale

1. Il Docente si reca nell'area relativa alle news
2. Il Docente chiede al Sistema di pubblicare nuove notizie per un corso scelto

3. Il Sistema mostra al Docente tutte le informazioni immesse sino ad ora e si predispone per il nuovo inserimento
4. Il Docente inserisce le nuove notizie
5. Il Docente chiede al Sistema di Salvare i dati immessi
6. Il Sistema comunica al docente che l'operazione è avvenuta con successo.

Scenario di fallimento:

6a. Il Sistema comunica al Docente che si sono verificati problemi e non è stato possibile inserire le nuove news.

3.3.1.5. Caso d'uso Pubblica FAQ:

Attori : Docente, Sistema

Obiettivo : pubblicare nuove FAQ

Frequenza : media

Condizione di successo : il Docente riesce a pubblicare le FAQ

Condizione di fallimento : il Docente non riesce a salvare le FAQ

Scenario Principale

1. Il Docente si reca nell'area relativa alle FAQ
2. il Docente sceglie il corso per cui vuole pubblicare le FAQ
3. Il Docente chiede al Sistema di pubblicare nuove FAQ
4. Il Sistema mostra al Docente tutte le informazioni immesse sino ad ora e si predispone per il nuovo inserimento
5. Il Docente inserisce le nuove FAQ
6. Il Docente chiede al Sistema di Salvare i dati immessi
7. Il Sistema comunica al docente che l'operazione è avvenuta con successo.

Scenario di fallimento:

7a. Il Sistema comunica al Docente che si sono verificati problemi e non è stato possibile inserire le nuove FAQ.

3.3.1.6. Caso d'uso Pubblicare Materiale:

Attori : Docente, Sistema

Obiettivo : pubblicare materiale

Frequenza : alta

Condizione di successo : il Docente riesce a pubblicare il materiale

Condizione di fallimento : il Docente non riesce a pubblicare il materiale

Scenario Principale

1. Il Docente si reca nell'area relativa al materiale didattico
2. Il Docente sceglie il corso per cui vuole pubblicare il materiale
3. Il Docente chiede al Sistema di pubblicare nuovo materiale didattico
4. Il Sistema si predispone per il nuovo inserimento: richiede il nome del materiale, l'abstract e il nome del file dove è contenuto il materiale
5. Il Docente inserisce il nuovo materiale
6. Il Docente chiede al Sistema di Salvare i dati immessi
7. Il Sistema comunica al docente che l'operazione è avvenuta con successo.

Scenario di fallimento:

- 7a. Il Sistema comunica al Docente che si sono verificati problemi e non è stato possibile inserire il nuovo materiale.

3.3.1.7. Caso d'uso Partecipare al Forum:

Attori : Docente/Tutor, Sistema

Obiettivo : partecipare al forum

Frequenza : media

Condizione di successo : il Docente riesce a leggere ed eventualmente pubblicare un messaggio

Condizione di fallimento : il Docente non riesce ad accedere

Scenario Principale

1. Il Docente si reca nell'area relativa al Forum
2. Il Docente sceglie il corso
3. Il Sistema mostra al Docente tutta la "conversazione" avvenuta sul forum
4. Il Docente decide di scrivere qualcosa sul forum (risposta ad una domanda, chiarimento ad un dubbio degli studenti.....)

5. Il Docente dopo aver letto le “conversazioni” non compie nessuna azione
6. Il Docente chiede al Sistema di Salvare i dati immessi
7. Il Sistema comunica al docente che l’operazione è avvenuta con successo.

Scenario di fallimento:

- 6a. Il Sistema comunica al Docente che si sono verificati problemi e non è stato possibile inserire le informazioni.

3.3.1.8. Caso d’uso Valuta Carriera Studente:

Attori : Docente, Sistema

Obbiettivo : assegnare un voto allo studente

Frequenza : media

Condizione di successo : il Docente riesce a valutare lo studente

Condizione di fallimento : il Docente non riesce ad accedere ai dati dello studente

Scenario Principale

1. Il Docente si reca nell’area della valutazione
2. Il Sistema richiede al Docente il nome e la matricola dello studente a cui è interessato
3. Il Docente inserisce i dati richiesti
4. Il Sistema mostra al Docente tutte le informazioni relative alla carriera dello studente nel corso: valutazione delle esercitazioni svolte, valutazione del progetto o risultato della prova al calcolatore
5. Il Docente assegna un voto allo studente in base alla sua carriera
6. Il Docente chiede al Sistema di salvare le informazioni
7. Il Sistema comunica al docente che l’operazione è avvenuta con successo.

Scenario di fallimento:

- 7a. Il Sistema comunica al Docente che si sono verificati problemi e non è stato possibile inserire le informazioni.

3.3.1.9. Caso d'uso Redigere Esame:

Attori : Docente, Sistema

Obbiettivo : scrivere le domande dell'esame

Frequenza : media

Condizione di successo : il Docente riesce ad inserire le domande dell'esame

Condizione di fallimento : il Docente non riesce a pubblicare le domande

Scenario Principale

1. Il Docente si reca nell'area relativa alle prove d'esame
2. Il Docente sceglie il corso per cui vuole inserire un nuovo esame
3. Il Docente chiede al Sistema di inserire le nuove domande dell'esame
4. Il Sistema si predispone per il nuovo inserimento
5. Il Docente inserisce le domande
6. Il Docente chiede al Sistema di Salvare i dati immessi
7. Il Sistema comunica al docente che l'operazione è avvenuta con successo.

Scenario di fallimento:

- 7a. Il Sistema comunica al Docente che si sono verificati problemi e non è stato possibile inserire le nuove domande.

3.3.1.10. Caso d'uso Correggere Esame:

Attori : Docente, Sistema

Obbiettivo : correggere l'esame svolto al calcolatore

Frequenza : media

Condizione di successo : il Docente corregge l'esame

Condizione di fallimento : il Docente non può correggere l'esame

Scenario Principale

1. Il Docente si reca nell'area relativa alle prove d'esame
2. Il Docente seleziona il corso a cui è interessato
3. Il Docente chiede al Sistema di visionare le risposte date dagli studenti alle domande

4. Il Sistema mostra al Docente le risposte date da ogni studente e propone un voto per ognuno di essi
5. Il Docente chiede al Sistema di mostrare le Statistiche dell'esame
6. Il Sistema elabora le statistiche e le mostra al Docente
7. Il Docente visto l'andamento dell'esame può confermare il voto proposto dal Sistema oppure può richiedere al Sistema di variare i criteri di Valutazione e ricalcolare un voto, oppure può decidere di inserire manualmente i voti rivalutando tutte le risposte date dagli studenti nel caso non sia soddisfatto del lavoro del Sistema
8. Il Docente chiede al Sistema di memorizzare il risultato dell'esame
9. Il Sistema comunica al docente che l'operazione è avvenuta con successo.

Scenario di fallimento:

- 9a. Il Sistema comunica al Docente che si sono verificati problemi e non è stato possibile inserire le nuove domande.

3.3.1.11. Caso d'uso Inserire Proposte:

Attori : Docente, Sistema

Obbiettivo : inserire proposte di progetti

Frequenza : bassa

Condizione di successo : il Docente riesce a inserire le proposte

Condizione di fallimento : il Docente non riesce ad accedere al sistema

Scenario Principale

1. Il Docente si reca nell'area relativa ai progetti
2. Il Docente sceglie il corso
3. Il Docente chiede al Sistema di pubblicare nuova proposta di progetto
4. Il Sistema si predispone per il nuovo inserimento
5. Il Docente inserisce la nuova proposta
6. Il Docente chiede al Sistema di Salvare i dati immessi

7. Il Sistema comunica al docente che l'operazione è avvenuta con successo.

Scenario di fallimento:

11a. Il Sistema comunica al Docente che si sono verificati problemi e non è stato possibile inserire la nuova proposta.

3.3.1.12. Caso d'uso Valutazione Versione Parziale:

Attori : Docente/Tutor, Sistema

Obbiettivo : valutare una versione parziale di un progetto

Frequenza : bassa

Condizione di successo : il Docente valuta la versione

Condizione di fallimento : il Docente non riesce ad accedere al sistema

Scenario Principale

1. Il Docente si reca nell'area relativa ai progetti
2. Il Sistema mostra al Docente le richieste di valutazioni parziali pendenti
3. Il Docente sceglie la richiesta
4. Il Sistema presenta il materiale relativo alla richiesta scelta
5. Il Docente visiona il materiale ed esprime un parere in merito al lavoro
6. Il Docente chiede al Sistema di memorizzare il commento
7. Il Sistema comunica al docente che l'operazione è avvenuta con successo (il Sistema invia anche una mail con il commento allo studente).

Scenario di fallimento:

7a. Il Sistema comunica al Docente che si sono verificati problemi e non è stato possibile memorizzare la valutazione.

3.3.1.13. Caso d'uso Valutazione Versione Finale:

Attori :Docente, Sistema

Obbiettivo : valutare una versione finale di un progetto

Frequenza : bassa

Condizione di successo : il Docente valuta la versione

Condizione di fallimento : il Docente non riesce ad accedere al sistema

Scenario Principale

1. Il Docente si reca nell'area relativa ai progetti
2. Il Sistema mostra al Docente le richieste di valutazioni finali pendenti
3. Il Docente sceglie la richiesta
4. Il Sistema presenta il materiale relativo alla richiesta scelta
5. Il Docente visiona il materiale ed esprime un giudizio in merito al lavoro
6. Il Docente chiede al Sistema di memorizzare il commento
7. Il Sistema comunica al docente che l'operazione è avvenuta con successo

Scenario di fallimento:

7a. Il Sistema comunica al Docente che si sono verificati problemi e non è stato possibile memorizzare la valutazione.

3.3.1.14. Caso d'uso Inserire Esercitazioni:

Attori : Docente/Tutor, Sistema

Obbiettivo : inserire una nuova esercitazione

Frequenza : media

Condizione di successo : il Docente riesce a inserire il testo della nuova esercitazione

Condizione di fallimento : il Docente non riesce ad accedere al sistema

Scenario Principale

1. Il Docente si reca nell'area relativa alle esercitazioni
2. Il Docente sceglie il corso
3. Il Docente chiede al sistema di inserire una nuova esercitazione
4. Il Sistema si predispone per il nuovo inserimento: predispone un'area dove inserire il testo e una dove viene indicata la data ultima di consegna
5. Il Docente inserisce la nuova esercitazione
6. Il Docente chiede al Sistema di Salvare i dati immessi

7. Il Sistema comunica al docente che l'operazione è avvenuta con successo.

Scenario di fallimento:

7a. Il Sistema comunica al Docente che si sono verificati problemi e non è stato possibile inserire la nuova esercitazione.

3.3.1.15. Caso d'uso Ispezione Attività:

Attori : Docente/Tutor, Sistema

Obbiettivo : ispezionare le attività relative alle esercitazioni

Frequenza : media

Condizione di successo : il Docente riesce ad ispezionare le attività

Condizione di fallimento : il Docente non riesce ad accedere al sistema

Scenario Principale

1. Il Docente si reca nell'area relativa alle esercitazioni
2. Il Docente chiede al Sistema di visionare le attività in corso
3. Il Sistema elabora i dati necessari e mostra al Docente lo stato di correzione delle esercitazioni
 - 3.1. Il Docente è soddisfatto dell'andamento dei lavori
 - 3.2. Il docente non è soddisfatto e chiede delle modifiche(scenario Modifica Attività)

3.3.1.16. Caso d'uso Modifica Attività:

Attori : Docente/Tutor, Sistema

Obbiettivo : modifica le attività relative alle esercitazioni

Frequenza : media

Condizione di successo : il Docente riesce a modifica le attività

Condizione di fallimento : il Docente non riesce ad accedere al sistema

Scenario Principale

1. Il Docente si reca nell'area relativa alle esercitazioni
2. Il Docente chiede al Sistema di visionare le attività in corso
3. Il Sistema elabora i dati necessari e mostra al Docente lo stato

- di correzione delle esercitazioni
4. Il Docente chiede al Sistema di apportare qualche modifica alle attività
 5. Il Sistema si predispone a modificare le attività
 6. Il Docente apporta le modifiche
 7. Il Docente chiede al Sistema di Salvare i dati immessi
 8. Il Sistema comunica al docente che l'operazione è avvenuta con successo.

Scenario di fallimento:

8a. Il Sistema comunica al Docente che si sono verificati problemi e non è stato possibile inserire la nuova esercitazione.

3.3.1.17. Caso d'uso Controlla Formato:

Attori : Docente/Tutor, Sistema

Obiettivo : controllare che il materiale inviato dallo studente rispetti il formato stabilito dal docente

Frequenza : alta

Condizione di successo : il Docente controlla il formato

Condizione di fallimento : il Docente non riesce ad accedere al sistema

Scenario Principale

1. Il Docente si reca nell'area relativa alle esercitazioni
2. Il Docente chiede al Sistema di avviare la correzione delle nuove esercitazioni consegnate
3. Il Sistema elabora i dati necessari e mostra al Docente la lista delle nuove esercitazioni
4. Il Docente seleziona una esercitazione
5. Il Sistema mostra al Docente l'elenco del materiale inviato dallo studente
6. Il Docente verifica che il materiale rispetti il formato prestabilito, se questo si verifica si comunica al Sistema che l'esercitazione può passare allo stadio successivo di correzione, altrimenti si comunica che questa fase è "fallita"
7. Il Docente chiede al Sistema di memorizzare i dati immessi

8. Il Sistema comunica al docente che l'operazione è avvenuta con successo.

Scenario di fallimento:

8a. Il Sistema comunica al Docente che si sono verificati problemi e non è stato possibile salvare i risultati.

3.3.1.18. Caso d'uso Invia Mail:

Attori : Sistema

Obbiettivo : notificare allo studente che la sua esercitazione presenta problemi

Frequenza : alta

Condizione di successo : il Sistema riesce ad inviare la mail

Condizione di fallimento : il Sistema non riesce ad inviare la mail

Scenario Principale

1. Quando il Docente ha dichiarato una fase di correzione “fallita”, il Sistema ricerca la mail relativa allo studente
2. Il Sistema prepara la mail da inviare
3. Il Sistema invia la mail

Scenario di fallimento:

- 3.a il Sistema non riesce ad inviare la mail

3.3.1.19. Caso d'uso Verifica Originalità:

Attori : Docente/Tutor, Sistema

Obbiettivo : controllare che gli studenti non abbiano copiato

Frequenza : alta

Condizione di successo : il Docente controlla eventuali copiatore

Condizione di fallimento : il Docente non riesce ad accedere al sistema

Scenario Principale

1. Il Docente si reca nell'area relativa alle esercitazioni
2. Il Docente chiede al Sistema mostrare le esercitazioni che hanno superato con successo la fase 1 di correzione
3. Il Sistema elabora i dati necessari e mostra al Docente la lista delle esercitazioni

4. Il Docente seleziona una esercitazione
5. Il Sistema mostra al Docente l'elenco del materiale inviato dallo studente
6. Il Docente verifica che non siano presenti copie tra le esercitazioni inviate, se non ne individua comunica al Sistema che l'esercitazione può passare allo stadio successivo di correzione, altrimenti si comunica che questa fase è "fallita"
7. Il Docente chiede al Sistema memorizzare i dati immessi
8. Il Sistema comunica al docente che l'operazione è avvenuta con successo.

Scenario di fallimento:

8a. Il Sistema comunica al Docente che si sono verificati problemi e non è stato possibile inserire la nuova esercitazione.

3.3.1.20. Caso d'uso Correzione e Test:

Attori : Docente/Tutor, Sistema

Obiettivo : terminare la correzione dell'esercitazione

Frequenza : alta

Condizione di successo : il Docente termina la correzione

Condizione di fallimento : il Docente non riesce ad accedere al sistema

Scenario Principale

1. Il Docente si reca nell'area relativa alle esercitazioni
2. Il Docente chiede al Sistema di mostrare l'elenco delle esercitazioni per cui la fase 2 di correzione è terminata con successo
3. Il Sistema elabora i dati necessari e mostra al Docente la lista delle esercitazioni
4. Il Docente seleziona una esercitazione
5. Il Sistema mostra al Docente l'elenco del materiale inviato dallo studente
6. Il Docente corregge il materiale inviato (controlla che il codice funzioni, legge la relazione) ed esegue le opportune operazioni di test, se il test ha successo si comunica al Sistema che l'esercitazione ha terminato la fase di correzione e si assegna

- un giudizio, altrimenti si comunica che questa fase è “fallita”
7. Il Docente chiede al Sistema di memorizzare i dati immessi
 8. Il Sistema comunica al docente che l’operazione è avvenuta con successo.

Scenario di fallimento:

- 8a. Il Sistema comunica al Docente che si sono verificati problemi e non è stato possibile inserire la nuova esercitazione.

3.3.2. Casi d’uso dell’attore Studente

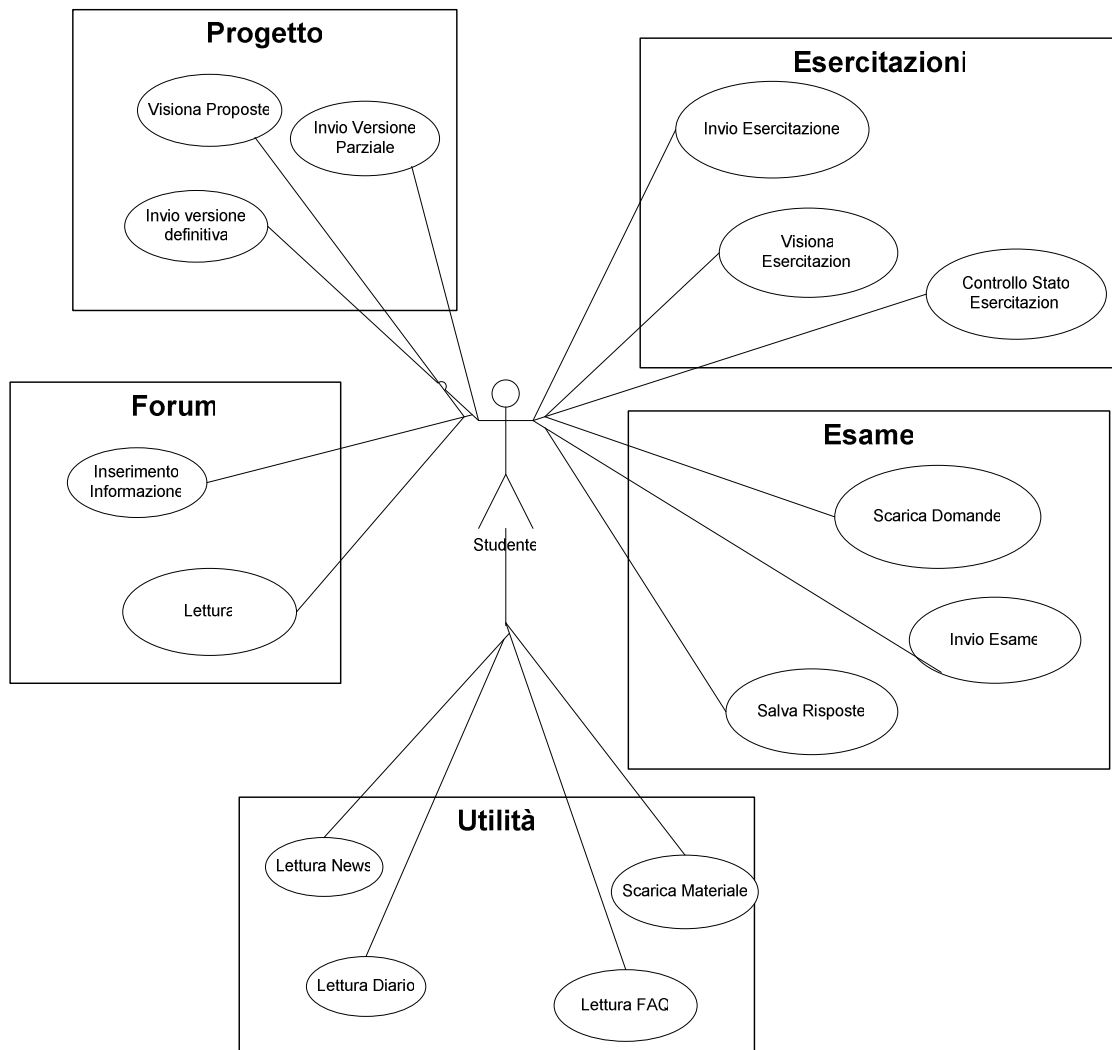


Fig. 3.3 – Casi d’uso Studente

3.3.2.1. Caso d'uso Lettura Diario

Attori : Studente, Sistema

Obbiettivo : leggere il diario delle lezioni

Frequenza : media

Condizione di successo : lo studente legge il diario

Condizione di fallimento : lo studente non riesce ad accedere al sistema

Scenario Principale

1. Lo Studente sceglie il corso dall'elenco dei corsi a cui risulta iscritto
2. Lo Studente si reca nell'area relativa al diario delle lezioni.
3. Lo Studente legge il diario delle lezioni

3.3.2.2. Caso d'uso Lettura News

Attori : Studente, Sistema

Obbiettivo : leggere le news

Frequenza : media

Condizione di successo : lo studente legge le news

Condizione di fallimento : lo studente non riesce ad accedere al sistema

Scenario Principale

1. Lo Studente sceglie il corso dall'elenco dei corsi a cui risulta iscritto
2. Lo Studente si reca nell'area relativa alle news.
3. Lo Studente legge le news disponibili

3.3.2.3. Caso d'uso Lettura FAQ

Attori : Studente, Sistema

Obbiettivo : leggere le FAQ

Frequenza : media

Condizione di successo : lo studente legge le FAQ

Condizione di fallimento : lo studente non riesce ad accedere al sistema

Scenario Principale

1. Lo Studente sceglie il corso dall'elenco dei corsi a cui risulta iscritto
2. Lo Studente si reca nell'area relativa alle FAQ.
3. Lo Studente legge le FAQ disponibili

3.3.2.4. Caso d'uso Scarica Materiale

Attori : Studente, Sistema

Obbiettivo : Scaricare il materiale

Frequenza : alta

Condizione di successo : lo studente scarica il materiale

Condizione di fallimento : lo studente non riesce ad accedere al sistema

Scenario Principale

1. Lo Studente sceglie il corso dall'elenco dei corsi a cui risulta iscritto
2. Lo Studente si reca nell'area relativa al materiale didattico.
3. Lo Studente sceglie il materiale che intende scaricare

3.3.2.5. Caso d'uso Scarica Domande

Attori : Studente, Sistema

Obbiettivo : Scaricare le domande dell'esame

Frequenza : bassa

Condizione di successo : lo studente scarica le domande

Condizione di fallimento : lo studente non riesce ad accedere al sistema

Scenario Principale

1. Lo Studente sceglie il corso dall'elenco dei corsi a cui risulta iscritto
2. Lo Studente si reca nell'area relativa all'esame.
3. Lo Studente chiede al Sistema di scaricare le domande dell'esame
4. Il Sistema elabora il file delle domande e presenta le domande allo studente

Scenario di fallimento:

4.a il Sistema segnala che si è verificato un problema e non può mostrare allo Studente le domande

3.3.2.6. Caso d'uso Salva Risposte

Attori : Studente, Sistema

Obbiettivo : Salvare le risposte alle domande dell'esame

Frequenza : bassa

Condizione di successo : lo studente salva le risposte

Condizione di fallimento : lo studente non riesce a salvare le risposte

Precondizioni: Lo Studente sta svolgendo la prova d'esame ed ha già scaricato le domande.

Scenario Principale

1. Lo Studente scorre le domande che compongono l'esame e risponde ad esse
2. Lo Studente chiede al Sistema di salvare le risposte date sino a quel momento
3. Il Sistema comunica allo Studente che il salvataggio è stato compiuto con successo

Scenario di fallimento:

3.a il Sistema segnala che si è verificato un problema e non può salvare le risposte

3.3.2.7. Caso d'uso Invio Esame

Attori : Studente, Sistema

Obbiettivo : invio dell'esame

Frequenza : bassa

Condizione di successo : lo studente invia l'esame

Condizione di fallimento : lo studente non riesce a salvare le domande

Precondizioni: Lo Studente sta svolgendo la prova d'esame ed ha già terminato la sua sessione

Scenario Principale

1. Lo Studente chiede al Sistema di salvare in modo definitivo le risposte
2. Il Sistema comunica allo Studente che il salvataggio è stato compiuto con successo

Scenario di fallimento:

2.a il Sistema segnala che si è verificato un problema e non può salvare le risposte

3.3.2.8. Caso d'uso Lettura Forum

Attori : Studente, Sistema

Obbiettivo : leggere il forum

Frequenza : media

Condizione di successo : lo studente legge il forum

Condizione di fallimento : lo studente non riesce ad accedere al sistema

Scenario Principale

1. Lo Studente sceglie il corso dall'elenco dei corsi a cui risulta iscritto
2. Lo Studente si reca nell'area relativa al Forum.
3. Lo Studente legge le informazioni

3.3.2.9. Caso d'uso Inserisci Informazioni

Attori : Studente, Sistema

Obbiettivo : inserire le informazioni desiderate nel Forum

Frequenza : media

Condizione di successo : lo studente inserisce le informazioni

Condizione di fallimento : lo studente non riesce ad accedere al sistema

Scenario Principale

1. Lo Studente sceglie il corso dall'elenco dei corsi a cui risulta iscritto
2. Lo Studente si reca nell'area relativa al Forum.
3. Lo Studente chiede al sistema di poter inserire informazioni
4. Il Sistema si predispone all'inserimento

5. Lo Studente inserisce l'informazione
6. Il Sistema salva i dati immessi
7. Il Sistema comunica che l'inserimento ha avuto successo

Scenario di fallimento:

- 7.a il Sistema segnala che si è verificato un problema

3.3.2.10. Caso d'uso Visiona Proposte

Attori : Studente, Sistema

Obbiettivo : visionare proposte di Progetto

Frequenza : media

Condizione di successo : lo studente visiona le proposte

Condizione di fallimento : lo studente non riesce ad accedere al sistema

Scenario Principale

1. Lo Studente sceglie il corso dall'elenco dei corsi a cui risulta iscritto
2. Lo Studente si reca nell'area relativa ai progetti.
3. Lo Studente visiona i progetti disponibili

3.3.2.11. Caso d'uso Invio versione parziale

Attori : Studente, Sistema

Obbiettivo : inviare una versione parziale del progetto

Frequenza : media

Condizione di successo : lo studente invia la versione

Condizione di fallimento : lo studente non riesce ad accedere al sistema

Scenario Principale

1. Lo Studente sceglie il corso dall'elenco dei corsi a cui risulta iscritto
2. Lo Studente si reca nell'area relativa ai progetti.
3. Lo Studente chiede al sistema di poter inviare una versione parziale del progetto
4. Il Sistema si predispone all'inserimento della versione
5. Lo Studente inserisce il materiale

6. Il Sistema salva i dati immessi

7. Il Sistema comunica che l'inserimento ha avuto successo

Scenario di fallimento:

7.a il Sistema segnala che si è verificato un problema

3.3.2.12. Caso d'uso Invio versione finale

Attori : Studente, Sistema

Obbiettivo : inviare la versione finale del progetto

Frequenza : media

Condizione di successo : lo studente invia la versione

Condizione di fallimento : lo studente non riesce ad accedere al sistema

Scenario Principale

1. Lo Studente sceglie il corso dall'elenco dei corsi a cui risulta iscritto

2. Lo Studente si reca nell'area relativa ai progetti.

3. Lo Studente chiede al sistema di poter inviare la versione finale del progetto

4. Il Sistema si predispone all'inserimento della versione

5. Lo Studente inserisce il materiale

6. Il Sistema salva i dati immessi

7. Il Sistema comunica che l'inserimento ha avuto successo

Scenario di fallimento:

7.a il Sistema segnala che si è verificato un problema

3.3.2.13. Caso d'uso Visiona esercitazioni

Attori : Studente, Sistema

Obbiettivo : visionare l'elenco delle esercitazioni

Frequenza : media

Condizione di successo : lo studente visiona le esercitazioni

Condizione di fallimento : lo studente non riesce ad accedere al sistema

Scenario Principale

1. Lo Studente sceglie il corso dall'elenco dei corsi a cui risulta

iscritto

2. Lo Studente si reca nell'area relativa alle esercitazioni.
3. Lo Studente visiona l'elenco delle esercitazioni proposte

3.3.2.14. Caso d'uso Invio Esercitazione

Attori : Studente, Sistema

Obbiettivo : inviare una esercitazione

Frequenza : alta

Condizione di successo : lo studente invia l'esercitazione

Condizione di fallimento : lo studente non riesce ad accedere al sistema

Scenario Principale

1. Lo Studente sceglie il corso dall'elenco dei corsi a cui risulta iscritto
2. Lo Studente si reca nell'area relativa alle esercitazioni
3. Lo Studente chiede al sistema di poter inviare una esercitazione
4. Il Sistema si predispone all'inserimento dell'esercitazione
5. Lo Studente inserisce il materiale
6. Il Sistema salva i dati immessi
7. Il Sistema comunica che l'inserimento ha avuto successo

Scenario di fallimento:

- 7.a il Sistema segnala che si è verificato un problema

3.3.2.15. Caso d'uso Controlla Stato

Attori : Studente, Sistema

Obbiettivo : visionare lo stato di correzione delle esercitazioni

Frequenza : media

Condizione di successo : lo studente vede lo stato delle proprie esercitazioni

Condizione di fallimento : lo studente non riesce ad accedere al sistema

Scenario Principale

1. Lo Studente sceglie il corso dall'elenco dei corsi a cui risulta iscritto

2. Lo Studente si reca nell'area relativa alle esercitazioni.
3. Lo Studente chiede al Sistema di visionare lo stato delle proprie esercitazioni
4. Il Sistema elabora la richiesta e mostra allo Studente l'elenco delle esercitazioni che ha consegnato e lo stato di correzione a cui sono giunte.

3.3.3. Casi d'uso dell'attore Tutor

Molti dei casi d'uso dell'attore Tutor sono già stati illustrati nella sezione relativa al Docente, per cui non vengono riportati.

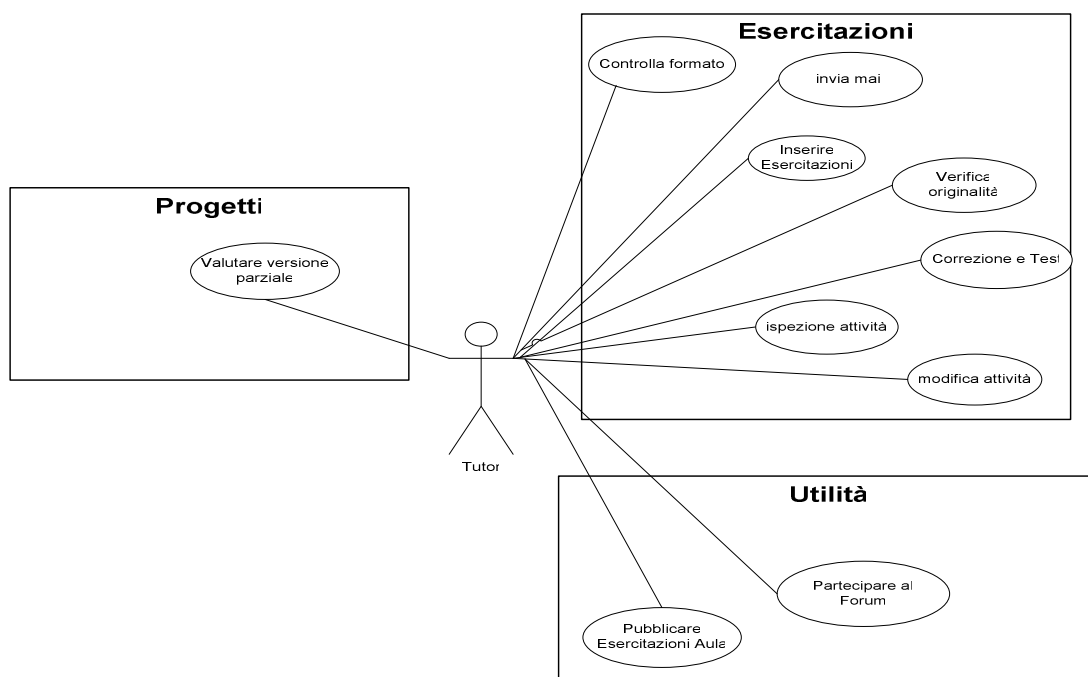


Fig. 3.4 – Casi d'uso Tutor

3.3.3.1. Caso d'uso Pubblicare esercitazione aula

Attori : Tutor, Sistema

Obiettivo : pubblicare materiale relativo alle esercitazioni svolte in aula

Frequenza : media

Condizione di successo : il Tutor riesce a pubblicare il materiale

Condizione di fallimento : il Tutor non riesce ad accedere al sistema

Scenario Principale

1. Il Tutor si reca nell'area relativa al materiale didattico
2. Il Tutor chiede al Sistema di pubblicare nuovo materiale didattico relativo ad un determinato corso
3. Il Sistema si predispone per il nuovo inserimento: richiede il nome del materiale, l'abstract e il nome del file dove è contenuto il materiale
4. Il Tutor inserisce il nuovo materiale
5. Il Tutor chiede al Sistema di Salvare i dati immessi
6. Il Sistema comunica al Tutor che l'operazione è avvenuta con successo.

Scenario di fallimento:

- 6a. Il Sistema comunica al Tutor che si sono verificati problemi e non è stato possibile inserire il nuovo materiale.

3.3.4. Casi d'uso dell'attore Amministratore

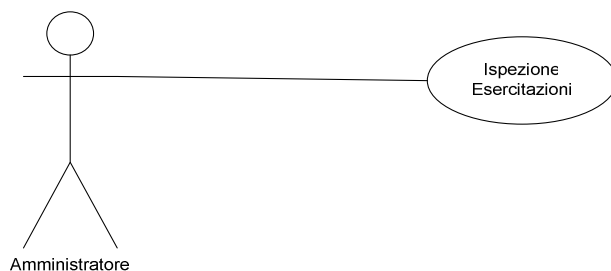


Fig. 3.5 – Casi d'uso Amministratore

3.3.4.1. Caso d'uso Ispezione Esercitazione

Attori : Amministratore, Sistema

Obbiettivo : controllare lo stato di avanzamento della correzione

delle esercitazioni

Frequenza : bassa

Condizione di successo : l'amministratore riesce a controllare lo stato di correzione

Condizione di fallimento : l'amministratore non riesce ad accedere al sistema

Scenario Principale

1. L'Amministratore esegue il login
2. Il Sistema presenta all'Amministratore l'area relativa alle esercitazioni
3. L'Amministratore può controllare lo stato di avanzamento nelle correzioni delle esercitazioni che sono state consegnate dagli studenti.

3.3.5. Workflow Individuati

Per quanto riguarda l'interazione degli utenti, è possibile individuare alcuni workflow che devono essere seguiti.

3.3.5.1. Esercitazioni:

1. Lo studente consegna l'esercitazione e riceve notifica di consegna
2. Viene controllato il formato dell'esercitazione, cioè che i file consegnati siano nel numero giusto, con l'estensione giusta e organizzati nel modo prestabilito
 - 2.1. se non compatibile alle specifiche invio mail allo studente e fine
 - 2.2. altrimenti continua
3. Viene controllato che non ci siano copie da altri studenti
 - 3.1. se sono presenti fermare processo e contattare tutti gli studenti coinvolti e il docente
4. Viene eseguito il test del corretto funzionamento. Viene gene-

rato un report che specifica l'esito del test

5. Tutto il materiale viene sottomesso alla valutazione del docente/tutor
6. Viene memorizzata la valutazione all'esercitazione
 - 6.1. se è negativa si agisce in base alle politiche stabilite: ad esempio si richiede allo studente il rinvio di una nuova versione

3.3.5.2. Progetti:

1. Lo studente sceglie una tematica di suo gradimento: la scelta può essere libera o limitata ad una lista predefinita
2. Durante lo svolgimento del progetto lo studente può inviare versioni parziali, se consentito
 - 2.1. In questo caso, viene notificato al docente la presenza di una nuova versione
 - 2.2. Il docente/tutor valuta il lavoro ed esprime un commento
3. Lo studente invia la versione definitiva del progetto
4. Il docente/tutor visiona la versione definitiva
5. Il docente/tutor esprime una valutazione del progetto

3.3.5.3. Esami:

1. Il docente prepara le domande
2. Lo studente, dopo essersi autenticato, scarica le domande dal sito
3. Lo studente risponde alle domande
4. Lo studente richiede salvataggio parziale delle risposte
5. Lo studente consegna l'esame
6. Lo studente riceve la valutazione

CAPITOLO 4

APPLICAZIONE DELLA METODOLOGIA SODA NEL PROGETTO DEL SISTEMA

Definiti i requisiti, i casi d'uso e i workflow dell'applicazione che si vuole sviluppare, si può procedere al progetto del sistema multi-agente in accordo alla metodologia SODA descritta nel capitolo 2.

Per prima cosa, quindi, si procede ad una fase di analisi dettagliata per individuare i task e le risorse, poi si passa al progetto del sistema.

4.1. FASE DI ANALISI

Analizzando le specifiche è possibile individuare molteplici attività necessarie al corretto funzionamento dell'applicazione. Queste attività riguardano tutti gli attori del sistema, ciascuno dei quali è interessato alla ricerca di determinate informazioni ed ha il compito di fornirne certe altre. Trascurando per semplicità le attività riguardanti l'Amministratore, cioè la gestione delle utenze, possiamo notare come i tre workflow principali possano individuare altrettante società di agenti relativamente indipendenti tra di loro: i relativi task potranno quindi essere assegnati a tre gruppi distinti. Vediamoli più in dettaglio.

4.1.1. Modello dei ruoli

La seguente tabella elenca i vari task individuati, fornendo per ciascuno di essi una sintetica descrizione.

<u>Task</u>	<u>Ruolo</u>	<u>Cosa deve fare</u>
Ricerca Informazioni	Visitatore generico	Andare alla ricerca di informazioni nel sito
Pubblica Materiale	Autore Materiale	Pubblicare materiale relativo al corso
Scarica Materiale	Lettore Materiale	Scaricare materiale di interesse
News	Redattore news	Inserisce notizie ed avvisi
Leggere news	Lettore news	Leggere le news
Registrazione	Nuovo Iscritto	Registrare un nuovo utente come studente che frequenta il corso
Login	Utente	Autenticarsi
Pubblica Esercitazioni	Redattore Esercitazioni (Gruppo Esercitazione)	Pubblicare il testo della nuova esercitazione con indicazione della data di consegna
Scarica Esercitazione	Visiona Esercitazione (Gruppo Esercitazione)	Scaricare il testo della esercitazione che deve essere svolta
Consegna Esercitazione	Studente Sviluppatore (Gruppo Esercitazione)	Depositare nel “repository” il materiale che lo studente invia

Controllo Stato Esercitazione	Studente Controllore (Gruppo Esercitazione)	Controlla lo stato di avanzamento delle esercitazioni che ha inviato
Controllo Formato	Controllore formato (Gruppo Esercitazione)	Controlla che il materiale inviato dallo studente rispetti le specifiche di formato imposte dal docente
Controllo Copiature	Controllore originalità (Gruppo Esercitazione)	Controlla che il materiale inviato dallo studente non sia stato copiato da altri studenti
Test Funzionamento	Tester (Gruppo Esercitazione)	Controlla che l'esercitazione consegnata dallo studente funzioni correttamente
Ispezione/modifica attività	Ispettore (Gruppo Esercitazione)	Controlla lo stato di avanzamento delle attività relative alle esercitazioni eventualmente facendo modifiche
Inserisce Proposta progetto	Redattore Progetti (Gruppo Progetto)	Inserisce il testo di proposte di progetto tra cui gli studenti potranno eventualmente scegliere
Scarica progetti	Visiona Progetti (Gruppo Progetto)	Scaricare l'eventuale elenco dei progetti proposti dal docente
Consegna Versione Parziale Progetto	Progettista (Gruppo Progetto)	Inviare materiale relativo al progetto che sta svolgendo

Valutazione Parziale	Revisore (Gruppo Progetto)	Fornire un commento sulla versione parziale consegnata
Consegna Finale Progetto	Progettista (Gruppo Progetto)	Invia codice e relazione del progetto
Creazione Prova d'esame	Esaminatore (Gruppo Esame)	Composizione delle domande teoriche e messa in linea
Salvataggio risposte intermedie	Esaminando (Gruppo Esame)	Richiesta di salvataggio delle risposte intermedie
Scarica domande esame	Visiona Domande (Gruppo Esame)	Richiesta di scaricare le domande dell'esame
Consegna Finale Esame	Esaminando (Gruppo Esame)	Consegna delle risposte definitive
Valutazione Esame	Valutatore (Gruppo Esame)	Valutazione dell'esame teorico consegnato dallo studente
Partecipazione al forum	Partecipante	Invia commenti anonimi sul corso ed interagisce con altri studenti
Scrivere il Diario	Redattore Diario	Aggiorna il diario di ciò che viene fatto durante le lezioni
Leggere il Diario	Lettore Diario	Lettura del diario delle lezioni
Scrivere FAQ	Autore FAQ	Pubblicare FAQ di interesse per gli studenti
Leggere FAQ	Lettore FAQ	Lettura delle FAQ
Verifica "Carriera" Studente	Verificatore	Controlla l'andamento didattico dello studente nel corso per formularne giudizi e valutazione finale

Tabella 4.1 – ruoli individuali e sociali

Questa tabella fornisce una panoramica esaustiva dei task individuati, ciascuno dei quali viene associato ad un determinato ruolo che dovrà essere assunto da un agente. Si può notare che la trattazione appare alquanto dettagliata, e che i ruoli siano stati elencati ad un livello di granularità molto fine. Tuttavia non è necessario costruire un agente per ogni task, perciò distinguere molti ruoli può aiutare nelle fasi successive, evitando che due ruoli solo apparentemente simili possano non essere distinti.

Per quanta riguarda i gruppi, la tabella 4.1 riporta solo i task individuali svolti dai ruoli appartenenti al gruppo, mentre i loro task sociali sono descritti nella tabella seguente.

<u>Task</u>	<u>Gruppo</u>	<u>Cosa deve fare</u>
Gestione e- esercitazioni	Esercitazione	Dopo la consegna dell'esercitazione devono partire le verifiche di formato e originalità, e se positive si prosegue nella correzione e nel test, tenendo traccia dello stato di avanzamento della correzione. Se qualche verifica non va a buon fine si prendono i provvedimenti del caso.
Gestione Progetti	Progetto	All'arrivo di ogni nuova versione di un progetto viene inoltrata notifica al docente, il quale può visionarla e valutarla. Occorre tener traccia della valutazione
Gestione e- same	Esame	Si deve prevedere la memorizzazione delle domande e delle risposte corrette, il salvataggio parziale delle risposte date dallo studente, la memorizzazione delle risposte definitive dello studente, la correzione automatica dell'elaborato

Tabella 4.2 – tabella dei task sociali

4.1.2. Modello delle risorse

I ruoli individuati nel precedente modello necessitano di alcuni servizi per portare a termine il loro task. Tali servizi devono essere forniti da alcune risorse accessibili agli agenti, coerentemente ai permessi che gli agenti stessi esibiscono. È possibile individuare diversi servizi, che possono essere raggruppati in base alla loro tipologia.

Lo schema sottostante descrive le quattro risorse individuate, specificando per ciascuna di esse quali servizi fornisce e come gli agenti possono accedervi.

Risorsa: Gestore Login

Servizi: Gestione autenticazione:

- Registrazione
- Login
- Ricerca Studente

Metodo di accesso:

Per registrazione: fornire dati dell'utente; restituisce successo/insuccesso. Per Interrogazione: inviare username e password; restituisce successo/insuccesso con specifica dei permessi concessi all'utente autenticato

Risorsa: Gestore Dati

Servizi: Gestione dei dati:

- Esercitazioni
- Progetti
- Domande esame
- Esami svolti dagli studenti
- Materiale didattico
- FAQ
- Forum

Metodo di accesso:

Load(dati); restituisce i dati richiesti.

Store(dati); restituisce successo/insuccesso.

Queste operazioni sono soggette al controllo dei permessi concessi all'utente.

Risorsa: Gestore Controllo

Servizi: Controllo di stato:

- Esercitazione
- Progetto

Metodo di accesso:

ControllaStato (esercitazione/progetto); restituisce lo stato di avanzamento dell'esercitazione richiesto.

ModificaStato (esercitazione/progetto), aggiorna lo stato a fronte di un evento.

Queste operazioni sono soggette al controllo dei permessi e dell'identità dell'utente, infatti lo studente può controllare lo stato solo delle sue esercitazioni, mentre docente/tutor/amministratore possono controllare lo stato di tutte, inoltre lo studente non può accedere alla modifica dello stato.

Risorsa: Gestore Comunicazioni

Servizi: Invio e-mail

Metodo di accesso:

InviaE-mail(studente, messaggio); permette di inviare una informazione ad uno specifico studente: questa informazione riguarda commenti sul progetto, problemi verificatisi nella correzione delle esercitazioni.

4.1.3. Modello di interazione

Individuati i ruoli e le risorse, è possibile descrivere i protocolli e le regole di interazione che li legano.

La seguente tabella descrive, per ogni ruolo individuato, le informazioni che richiede e quelle che deve fornire per interagire correttamente con gli altri ruoli.

<u>Ruolo</u>	<u>Informazioni richieste</u>	<u>Informazioni fornite</u>
Visitatore generico		
Autore Materiale	URL	Materiale
Lettore Materiale	URL	
Redattore news		Testo della news
Lettore News	News	
Nuovo Iscritto	Dati studente	Nuovo account
Utente	account	Identificativo utente
Redattore Esercitazioni (Gruppo Esercitazione)		Testo Esercitazione
Visiona Esercitazione (Gruppo Esercitazione)		Testo Esercitazione
Studente sviluppatore (Gruppo Esercitazione)	Identificativo utente	Esercitazione
Studente Controllore (Gruppo Esercitazione)	Identificativo utente, Esercitazione	
Controllore formato (Gruppo Esercitazione)	Esercitazione	Aggiornamento stato esercitazione
Controllore originalità (Gruppo Esercitazione)	Esercitazione	Aggiornamento stato esercitazione
Tester (Gruppo Esercitazione)	Esercitazione	Aggiornamento stato esercitazione
Ispettore (Gruppo Esercitazione)	Esercitazione	Aggiornamento stato esercitazione
Valutatore Esercitazioni (Gruppo Esercitazione)	Esercitazione	Valutazione esercitazione
Redattore Progetti (Gruppo Progetto)		Testo progetto

Visiona Progetti (Gruppo Progetto)		Testo progetti
Progettista (Gruppo Progetto)	Identificativo utente	Progetto
Revisore (Gruppo Progetto)	Progetto	Valutazione progetto
Progettista (Gruppo Progetto)	Identificativo utente	Progetto
Esaminatore (Gruppo Esame)		Testo Esame
Visiona Domande (Gruppo Esame)		Domande dell'esame
Esaminando (Gruppo Esame)	Identificativo utente, risposte	Backup Esame
Esaminando (Gruppo Esame)	Identificativo utente, risposte	Esame
Valutatore (Gruppo Esame)	Esame	Valutazione Esame
Partecipante		Messaggi forum
Redattore Diario		Aggiornamento Diario
Lettore Diario		Diario delle lezioni
Autore FAQ		Aggiorna le FAQ
Lettore FAQ		Elenco delle FAQ
Verificatore	Studente	Situazione studente

Tabella 4.3 – tabella protocolli di interazione associati ai ruoli

La tabella della pagina seguente mostra invece i protocolli di accesso di ogni risorsa individuata, che vengono descritti in base alle informazioni che devono essere fornite alla risorsa per poter usufruire del servizio che rende disponibile.

<u>Risorsa</u>	<u>Protocollo</u>
Gestore Login	• Registra(dati utente) • Login(username, password) • Ricerca(dati)
Gestore dati	• Fornisci(tipo dato, permessi accesso) • Memorizza(tipo dato, permessi accesso)
Gestore controllo	• ControllaStato(tipo dato, permessi accesso) • ModificaStato(tipo dato, permessi accesso)
Gestore Comunicazioni	• InviaE-mail(studente, messaggio)

Tabella 4.4 – tabella protocolli di interazione associati alle risorse

Per quanto riguarda le regole di interazione, si può notare che esse derivano principalmente dai workflow individuati durante l'analisi dei requisiti. Alla luce dei ruoli individuati e descritti in questa fase, le regole di interazione possono quindi essere riscritte in modo più chiaro come segue.

Gruppo: Esercitazione

Descrizione:

- il Redattore Esercitazioni pubblica il testo dell'esercitazione
- lo Studente Visionatore delle esercitazioni scarica il testo dell'esercitazione
- lo Studente Sviluppatore consegna la sua Esercitazione
- il Controllore Formato controlla il formato dell'esercitazione consegnata
- il Controllore Originalità controlla che non ci siano copie
- lo Studente Controllore e l'Ispettore possono sempre controllare lo stato di correzione dell'esercitazione
- il Tester verifica il funzionamento dell'esercitazione
- il Valutatore delle esercitazioni esprime quindi un giudizio sull'esercitazione dello studente

Gruppo: Progetto

Descrizione:

- il Redattore delle proposte di progetto pubblica il testo di un progetto
- lo Studente Visionatore dei progetti cerca tra le proposte di progetto
- il Progettista consegna il suo progetto
- il Revisore lo valuta ed esprime un commento

Gruppo: Esame

Descrizione:

- l'Esaminatore pubblica il testo dell'esame
- lo Studente Visionatore di domande scarica il testo dell'esame quando è consentito
- l'Esaminando salva le risposte parziali
- l'Esaminando consegna il suo Esame
- il Valutatore scarica l'Esame e lo corregge
- il Valutatore esprime un giudizio sull'Esame

4.2. FASE DI PROGETTO

A questo punto, individuati i ruoli, le risorse e i protocolli che costituiscono il sistema multi-agente, è possibile passare alla seconda fase della metodologia SODA, e studiare come queste astrazioni possano essere mappate in un sistema concretamente realizzabile. A tale scopo è necessario mappare i ruoli e i relativi task su classi di agenti, i task sociali in società di agenti e le risorse in classi di infrastruttura.

4.2.1. Modello degli Agenti

Data la molteplicità dei ruoli individuati, è possibile prevedere diverse classi di agenti, che vengono descritte negli schemi seguenti. Per ogni classe di agente vengono specificati i ruoli che assume e le risorse a cui accede, con i relativi permessi. Vengono descritti anche i protocolli di interazione relativi, espressi in termini di tuple logiche.

Agente: Visitatore Generico

Ruoli: Visitatore generico

Cardinalità: Molti

Risorse: Gestore Dati

Permessi: lettura informazioni non riservate

Protocolli: -

Agente: Autore Materiale

Ruoli: Autore Materiale

Cardinalità: Pochi

Risorse: Gestore Dati

Permessi: Salvataggio del proprio materiale

Protocolli: (out) newMat(titolo,url)

Agente: Lettore Materiale

Ruoli: Lettore Materiale

Cardinalità: Molti

Risorse: Gestore Dati

Permessi: Download materiale

Protocolli: (out) reqMat(url)

Agente: Redattore news

Ruoli: Redattore news

Cardinalità: 1

Risorse: Gestore Dati

Permessi: Scrittura news

Protocolli: -

Agente: **Lettore News**

Ruoli: Lettore News

Cardinalità: Molti

Risorse: Gestore Dati

Permessi: Lettura news

Protocolli: -

Agente: **Nuovo Iscritto**

Ruoli: Nuovo Iscritto

Cardinalità: Molti

Risorse: Gestore Login

Permessi: Scrittura account

Protocolli: (out) registra(dati); (in) account(dati,Id,Pwd)

Agente: **Utente**

Ruoli: Utente

Cardinalità: Molti

Risorse: Gestore Login

Permessi: login

Protocolli: (out) login(username, password);

(in) login_response (Utente, Permessi)

Agente: **Redattore Esercitazioni**

Ruoli: Redattore Esercitazioni

Cardinalità: 1

Risorse: Gestore Dati

Permessi: Scrittura testi delle esercitazioni

Protocolli: (out) newEs(nomeEs, testoEs)

Agente: **Visiona Esercitazione**

Ruoli: Visiona Esercitazione

Cardinalità: Molti

Risorse: Gestore Dati

Permessi: Scarica testo esercitazione

Protocolli: (rd) newEs(nomeEs, testoEs)

Agente: **Studente Sviluppatore**

Ruoli: Studente Sviluppatore

Cardinalità: Molti

Risorse: Gestore Dati

Permessi: Salvataggio della propria esercitazione

Protocolli: (in) newEs(codice, Testo); (out) esercitazione(codice, studente)

Agente: **Studente Controllore**

Ruoli: Studente Controllore

Cardinalità: Molti

Risorse: Gestore Dati

Permessi: Lettura stato delle proprie esercitazioni

Protocolli: (in) statoEs(codice, studente, Stato)

Agente: **Controllore formato**

Ruoli: Controllore formato

Cardinalità: 1

Risorse: Gestore Dati

Permessi: Lettura esercitazioni consegnate dagli studenti

Protocolli: (in) esercitazione(codice, Studente);
(out) okform(codice, Studente)

Agente: **Controllore originalità**

Ruoli: Controllore originalità

Cardinalità: 1

Risorse: Gestore Dati

Permessi: Lettura esercitazioni consegnate dagli studenti

Protocolli: (in) esercitazione(codice, Studente);

(out) no_copie(codice, Studente)

Agente: **Tester**

Ruoli: Tester

Cardinalità: 1

Risorse: Gestore Dati

Permessi: Lettura esercitazioni consegnate dagli studenti

Protocolli: (in) esercitazione(codice, Studente);

(out) oktest(codice, Studente)

Agente: **Valutatore Esercitazioni**

Ruoli: Valutatore esercitazioni

Cardinalità: 1

Risorse: Gestore Dati

Permessi: Lettura esercitazioni consegnate dagli studenti

Protocolli: (in) esercitazione(codice, Studente);

(out) valutazione_eser(codice, Studente, voto)

Agente: **Ispettore**

Ruoli: Ispettore

Cardinalità: 1

Risorse: Gestore Dati

Permessi: Lettura e scrittura esercitazioni

Protocolli: statoEs(codice, studente, Stato)

Agente: **Redattore Progetti**

Ruoli: Redattore Progetti

Cardinalità: 1

Risorse: Gestore Dati

Permessi: Scrittura testo progetto

Protocolli: (out) nuovo_progetto(nomeProg, testoProg)

Agente: **Visiona Progetti**

Ruoli: Visiona Progetti

Cardinalità: Molti

Risorse: Gestore Dati

Permessi: Lettura dei progetti proposti

Protocolli: (rd) nuovo_progetto(nomeProg, testoProg)

Agente: **Progettista**

Ruoli: Progettista

Cardinalità: Molti

Risorse: Gestore Dati

Permessi: Salvataggio materiale progetto

Protocolli: (out) progetto(studente, descrizione)

Agente: **Revisore**

Ruoli: Revisore

Cardinalità: 1

Risorse: Gestore Dati

Permessi: Lettura materiale progetti

Protocolli: (in) progetto(Studente, Descrizione);
(out) valutazione_progetto(Studente, voto)

Agente: **Esaminatore**

Ruoli: Esaminatore

Cardinalità: 1

Risorse: Gestore Dati

Permessi: Lettura e scrittura esami

Protocolli: (out) esame(testo, durata)

Agente: **Visiona Domande**

Ruoli: Visiona Domande

Cardinalità: Molti

Risorse: Gestore Dati

Permessi: Lettura domande d'esame

Protocolli: (rd) esame(Testo, Durata)

Agente: **Esaminando**

Ruoli: Esaminando

Cardinalità: Molti

Risorse: Gestore Dati

Permessi: Lettura delle domande e scrittura delle risposte

Protocolli: (in) esame(Testo, Durata);

(out) risposte(studente, risposte)

Agente: **Valutatore**

Ruoli: Valutatore

Cardinalità: 1

Risorse: Gestore Dati

Permessi: Lettura risposte

Protocolli: (in) risposte(Studente, Risposte);

(out) valutazione_esame(Studente, voto)

Agente: **Partecipante**

Ruoli: Partecipante

Cardinalità: Molti

Risorse: Gestore Dati

Permessi: Lettura e scrittura forum

Protocolli: -

Agente: **Redattore Diario**

Ruoli: Redattore Diario

Cardinalità: 1

Risorse: Gestore Dati

Permessi: Lettura e scrittura diario

Protocolli: -

Agente: **Lettore Diario**

Ruoli: Lettore Diario

Cardinalità: Molti

Risorse: Gestore Dati

Permessi: Lettura del diario

Protocolli: -

Agente: **Autore FAQ**

Ruoli: Autore FAQ

Cardinalità: 1

Risorse: Gestore Dati

Permessi: Lettura e scrittura delle FAQ

Protocolli: -

Agente: **Lettore FAQ**

Ruoli: Lettore FAQ

Cardinalità: Molti

Risorse: Gestore Dati

Permessi: Lettura delle FAQ

Protocolli: -

Agente: **Verificatore**

Ruoli: Verificatore

Cardinalità: 1

Risorse: Gestore Dati

Permessi: Lettura esercitazioni, esami e progetti

Protocolli:

(in) valutazione_eser(codice, Studente, Voto),

 valutazione_progetto(Studente, Voto),

 valutazione_esame(Studente, Voto);

(out) voto_finale(Studente, voto)

4.2.2. Modello della Società

I gruppi e i relativi task sociali individuano tre distinte società di agenti, le cui regole sociali sono tali da garantire il rispetto dei workflow relativi. Vediamo dunque come le regole di interazione

delle società individuate possano essere espresse in termini di tuple logiche.

Società: Esercitazione

Gruppo: Esercitazione

Ruoli partecipanti: Redattore Esercitazioni; Visiona Esercitazioni; Studente Sviluppatore; Studente Controllore; Controllore Formato; Controllore Originalità; Tester; Ispettore; Valutatore Esercitazioni

Regole di interazione:

- Dopo che il Redattore pubblica una esercitazione, gli studenti possono reperirne il testo: [newEs(nomeEs,testo)]
- Lo sviluppatore chiede l'autorizzazione alla consegna [consegna(codice, studente)]: non è infatti consentito consegnare due volte la stessa esercitazione, né consegnare oltre una certa data fissata dal docente [abilitazione_consegna(codice, studente, R)]
- Lo sviluppatore consegna l'esercitazione [esercitazione(codice, studente)], e lo stato dell'esercitazione è "consegnata" [statoEs (codice, studente, 'consegnata')]
- Se il formato è corretto il Controllore Formato lo dichiara [okform(codice, studente)], e allora viene aggiornato lo stato dell'esercitazione [statoEs(codice, studente, 'verificato formato')]
- Se non è copiata il Controllore Originalità lo dichiara [no_copie(codice, studente)], e allora viene aggiornato lo stato dell'esercitazione [statoEs(codice, studente, 'in test')]
- Se il test è positivo il Tester lo dichiara [oktest(codice, studente)], e viene aggiornato lo stato dell'esercitazione [statoEs (codice, studente, 'corretta')]

Società: Progetto

Gruppo: Progetto

Ruoli partecipanti: Redattore Progetti; Visiona Progetti; Progetti-

sta; Revisore

Regole di interazione:

- Dopo che il Redattore Progetti pubblica proposte di progetto, gli studenti possono visionarle
[nuovo_progetto(nomeProg, testoProg)]
- Il Progettista consegna una o più versioni del suo progetto
[progetto(studente, descrizione)]
- Il Revisore legge e valuta il progetto
[valutazione_progetto(studente, voto)]

Società: Esame

Gruppo: Esame

Ruoli partecipanti: Esaminando; Visiona Domande; Esaminatore; Valutatore

Regole di interazione:

- L'Esaminatore pubblica le domande d'esame
[esame(Testo, Durata)]
- L'Esaminando legge le domande e salva le risposte date
[risposte(studente, risposte)]
- Il Valutatore legge le risposte definitive dello studente ed esprime un giudizio [valutazione_esame(Studente, voto)]

4.2.3. Modello dell'Ambiente

Le risorse possono essere mappate nelle classi di infrastruttura descritte di seguito.

Per ogni classe di infrastruttura vengono elencati i servizi che offre, i permessi di accesso ed i protocolli, sempre espressi in termini di tuple logiche.

Risorsa: **Gestore Login**

Servizi: Registrazione, Controllo accesso

Cardinalità: 1

Permessi:

Docente, Tutor, Amministratore: accesso completo

Studente: accesso limitato ad autenticazione e registrazione

Protocolli:

- Autenticazione:
 - (in) login(Uname, Password);
 - (out) login_response(utente, permessi)
- Registrazione: (in) registra(dati);
- Ricerca: (in)cerca(Matricola); (out) ris(dati)

Risorsa: **Gestore Dati**

Servizi: Lettura e Scrittura Dati

Cardinalità: 1

Permessi:

- Docente, Tutor, Amministratore: accesso completo
- Studente: accesso limitato ai suoi dati

Protocolli:

- (in) load(Dato, Utente), (out) dati(dato)
- (in) store(Dato, Utente)

Risorsa: **Gestore Comunicazioni**

Servizi: Invio e-mail

Cardinalità: 1

Permessi:

- Accesso consentito solo a docente/tutor

Protocolli:

- (in) mail(To, Obj, Body, Att)

4.3. ARCHITETTURA DEL SISTEMA

A questo punto, essendo definiti tutti i componenti principali del sistema multi-agente, è possibile descrivere l'architettura del si-

stema complessivo con l'aiuto di qualche diagramma UML, standard che continua ad essere un utile punto di riferimento per la successiva fase di sviluppo.

4.3.1. Diagramma delle classi

Il diagramma sottostante illustra schematicamente la struttura complessiva del sistema.

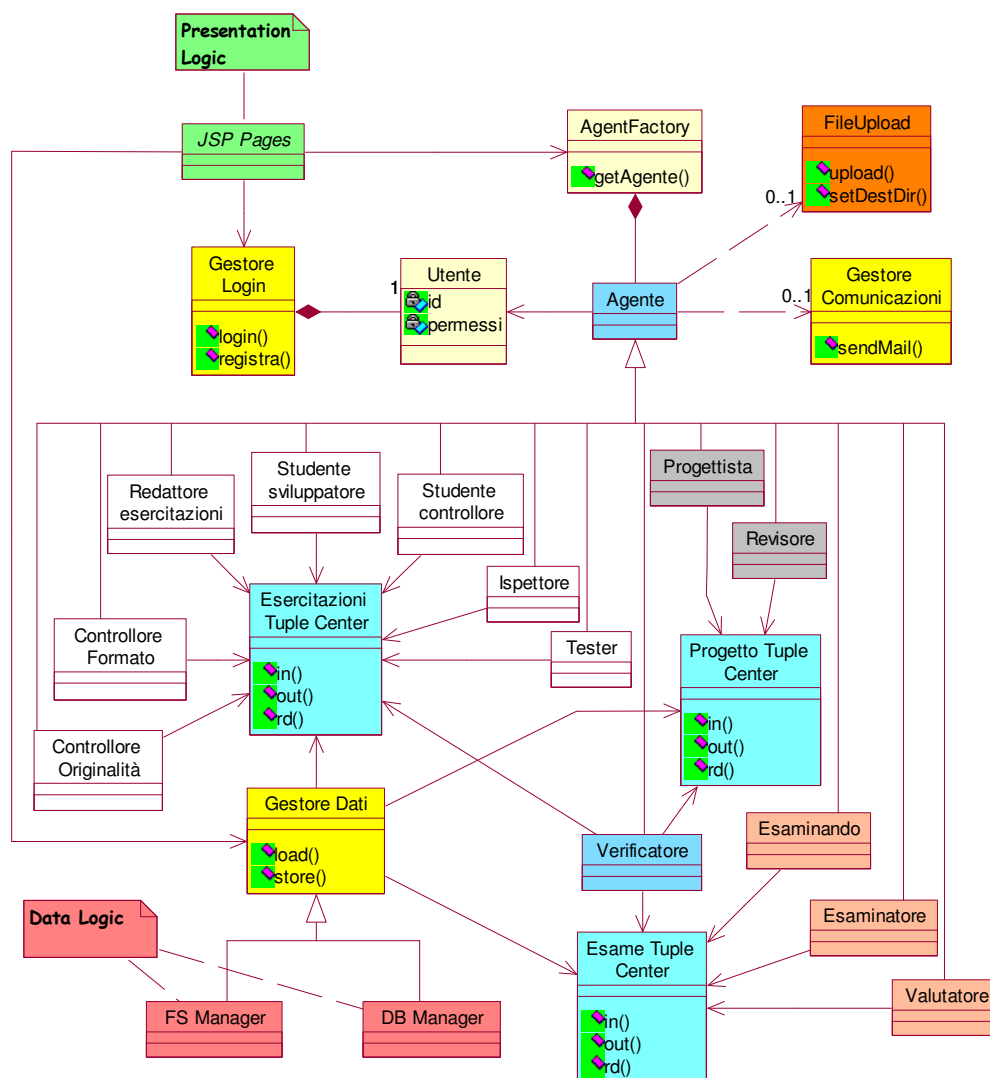


Fig. 4.1 – Architettura logica

Questa è l'architettura logica del Sistema in esame. Per comodità, sono state omesse tutte le classi di agenti individuali, cioè che non hanno necessità di coordinarsi con altri agenti; esse sono quindi in relazione con le sole classi di infrastruttura che forniscono i servizi a loro necessari.

La colorazione degli oggetti indicati è stata introdotta per evidenziare differenti tipologie di componenti: le classi di infrastruttura sono colorate in giallo, i centri di tuple in azzurro e gli agenti hanno un colore diverso a seconda del gruppo a cui appartengono; l'interfaccia utente (*presentation logic*) è colorata in verde, mentre i componenti *data logic* per l'accesso ai dati sono in rosso.

Si nota come gli agenti dei principali gruppi lavorano attorno a dei centri di tuple che rappresentano il meccanismo della loro interazione, ad eccezione dell'agente "verificatore" che per poter fornire un giudizio complessivo deve attingere informazioni da tutti i centri di tuple. Ogni centro di tuple, dunque, è dedicato ad un gruppo di agenti, perciò conterrà tutte e sole le informazioni necessarie allo svolgimento del loro task sociale.

Gli agenti vengono attivati da un front-end di componenti che rappresenta l'interfaccia grafica del sistema: dato che si tratta di un sistema web-based, è possibile prevedere che tale componente sia costituito da un insieme di pagine JSP. Tali agenti vengono attivati mediante una factory, perché l'attività di un agente potrebbe coinvolgere più pagine.

La risorsa `Gestore Login` deve provvedere all'identificazione degli utenti correnti, ed è quindi sua la responsabilità di istanziare le entità che li rappresentano con gli opportuni permessi. Per svolgere il suo compito, il `Gestore Login` avrà bisogno di leggere i profili degli utenti dalla base di dati, anche se per comodità nel diagramma tale associazione è stata sottintesa.

Il `Gestore Dati` rappresenta l'omonima classe di infrastruttura individuata, ed ha quindi la responsabilità di mediare la lettura e scrittura di dati su richiesta degli agenti. La sorgente dati può essere un database apposito, oppure un file system se il dato corren-

te è rappresentato da un file, come nel caso dei progetti o delle esercitazioni consegnate dagli studenti.

Alcuni agenti svolgono attività che richiedono l'invio di materiale da parte dello studente: l'upload di tale materiale viene gestito dal componente `FileUpload`.

`Esercitazioni Tuple Centre` è il centro di tuple che coordina l'attività di esercitazione: esso conterrà dunque le informazioni sullo stato di correzione di tutte le esercitazioni consegnate dagli studenti. Analogamente il `Progetto Tuple Centre` conterrà informazioni riguardanti i vari progetti, mentre `Esame Tuple Centre` conterrà quelle riguardanti le sessioni d'esame.

4.3.2. Diagrammi di sequenza

Per chiarire meglio il funzionamento del sistema si riportano alcuni diagrammi di sequenza.

Il primo diagramma (Figura 4.2) mostra l'elaborazione che deve essere compiuta quando uno studente consegna il materiale relativo ad una esercitazione svolta. Il secondo (Figura 4.3) rappresenta invece il controllo che il materiale dell'esercitazione consegnata dallo studente rispetti il formato stabilito dal docente. In entrambi i casi il controllo è affidato all'agente che viene attivato: esso deve controllare i permessi, svolgere le azioni necessarie a portare a compimento con successo il suo task, e infine registrare l'esito dell'operazione nel centro di tuple.

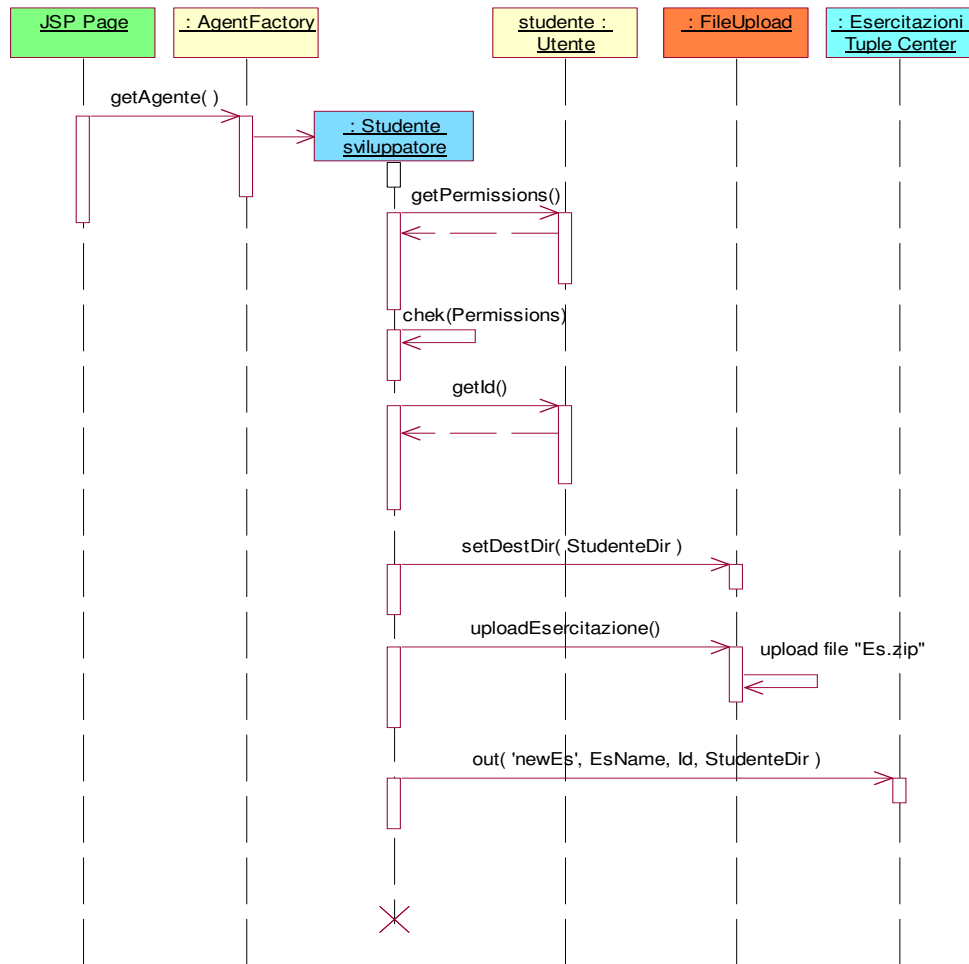


Fig. 4.2 – Diagramma di sequenza: “Consegna esercitazione”

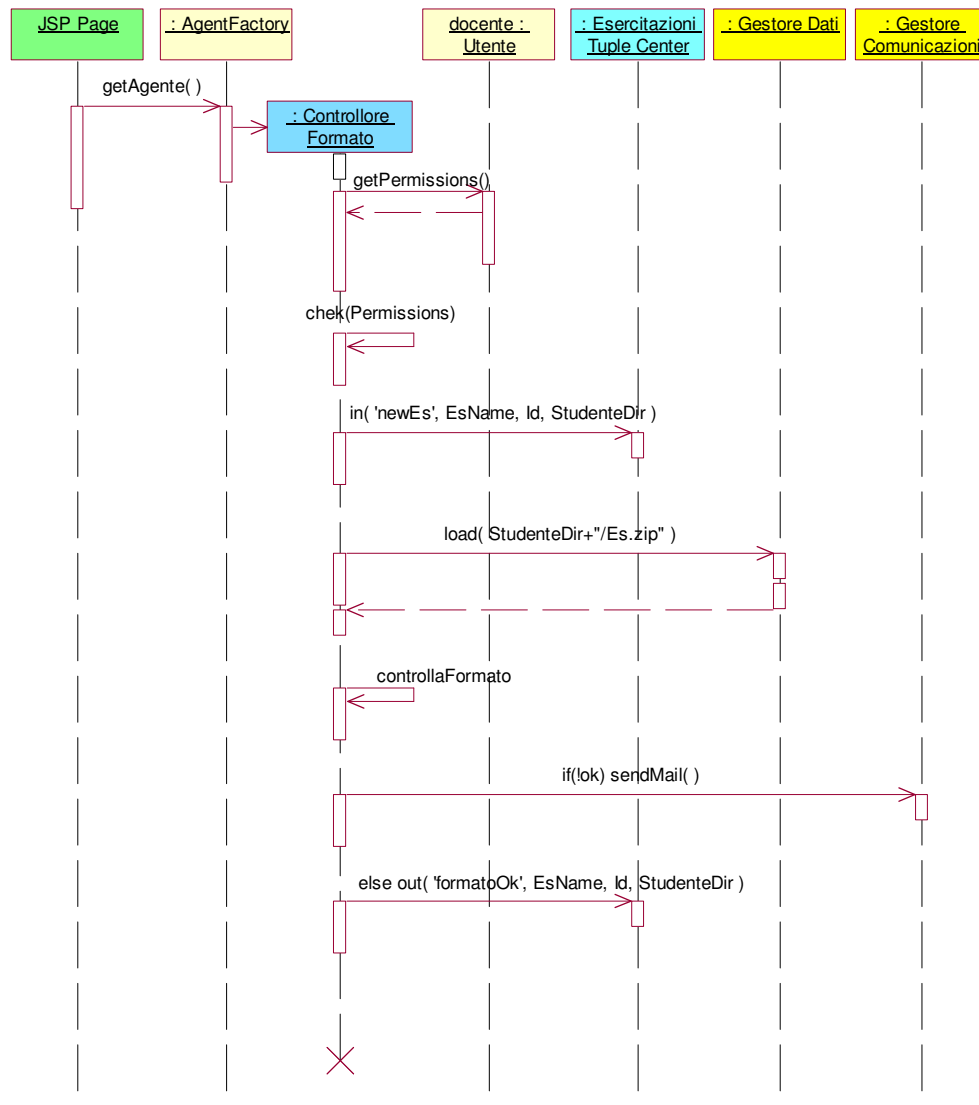


Fig. 4.3 – Diagramma di sequenza: “Controlla formato”

Il diagramma seguente (Figura 4.4) mostra la verifica del corretto funzionamento e la valutazione delle esercitazioni consegnate. È stata omessa la parte relativa alla verifica dei permessi dell'utente, dato che è la medesima dei due diagrammi precedenti.

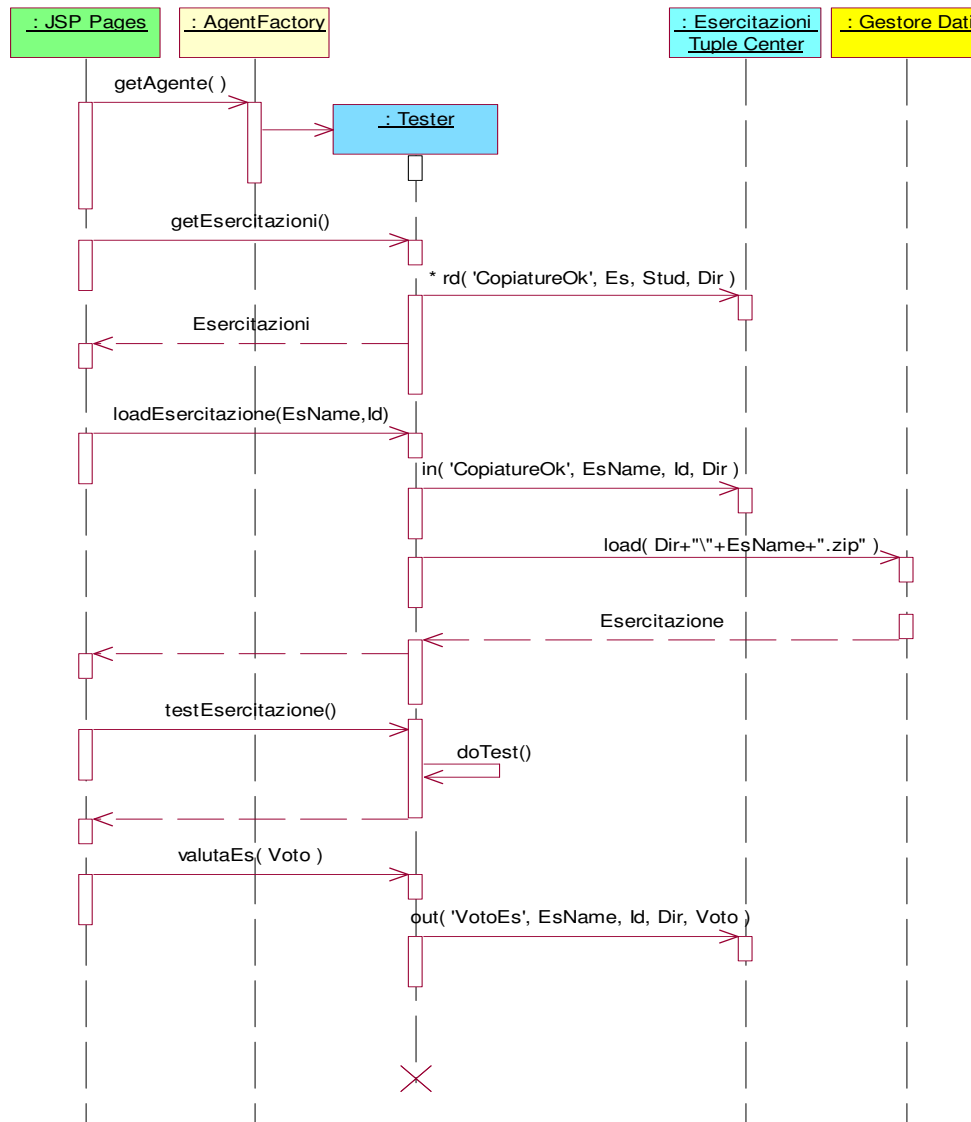


Fig. 4.4 – Diagramma di sequenza: “Correzione esercitazione”

L’ultimo diagramma (Figura 4.5) mostra le elaborazioni relative alla valutazione della carriera di uno studente nel corso, cioè l’assegnazione del voto finale in base delle prove svolte dal candidato. È stata di nuovo omessa la parte relativa alla verifica dei permessi dell’utente.

Si nota come, a differenza degli altri casi, per questa operazione sono necessarie informazioni riguardanti tutti i centri di tuple.

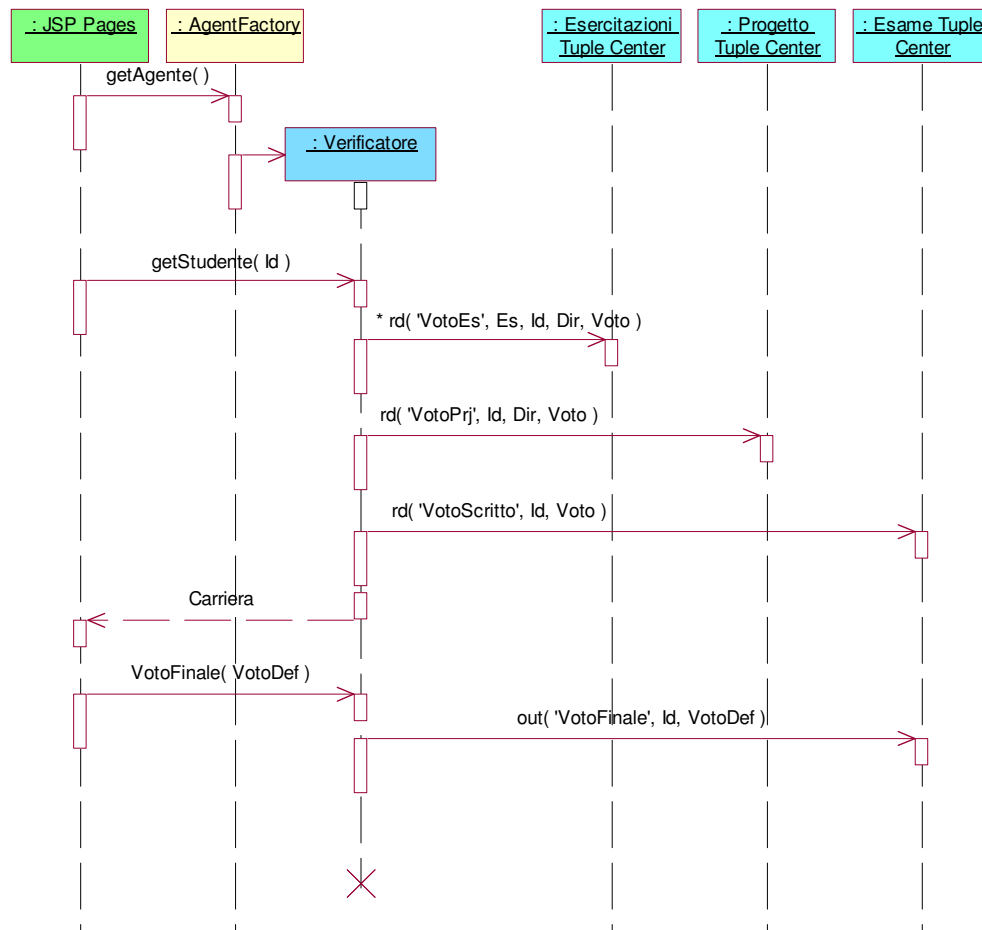


Fig. 4.5 – Diagramma di sequenza: “Valutazione carriera”

4.4. PROGETTO DELLA BASE DI DATI

Per concludere la fase di progetto è ora necessario modellare con un diagramma E-R la struttura relazionale dei dati che devono essere letti e scritti dal sistema complessivo.

Questa base di dati dovrà contenere, oltre alle informazioni relative alle entità in gioco, anche i dati relativi agli utenti.

I due diagrammi che seguono mostrano i legami concettuali che è

possibile individuare.

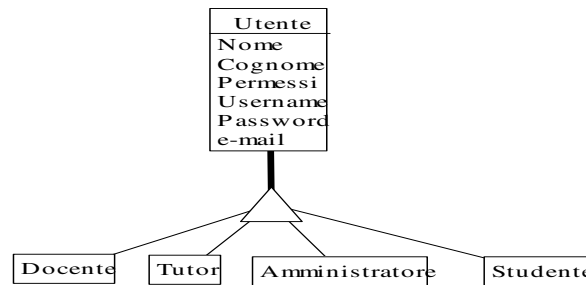


Fig. 4.6 – Diagramma E-R relativo alle categorie di utenti

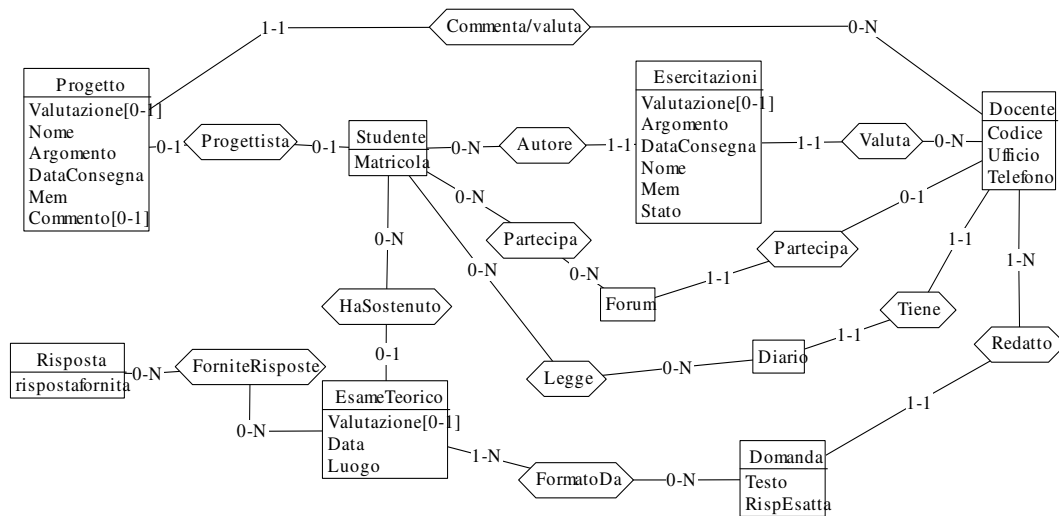


Fig. 4.7 – Diagramma E-R concettuale

CAPITOLO 5

IMPLEMENTAZIONE DEL SISTEMA

Conclusa la parte di progetto, è possibile passare allo sviluppo dell'applicazione, ed è quindi giunto il momento di effettuare le scelte implementative. Tali scelte riguardano principalmente tre aspetti: il linguaggio di programmazione, l'infrastruttura di coordinazione del sistema multi-agente e la tecnologia web.

Si è deciso di usare Java come linguaggio di programmazione principale, per poter sfruttare la sua portabilità e capacità di integrazione con diverse tecnologie. Per quanto riguarda l'interfaccia utente, si è deciso di usare la tecnologia JSP, e di sviluppare la parte web come applicazione per il server Apache Tomcat. Come infrastruttura di coordinazione degli agenti, infine, la scelta è TuCSoN, già descritto nel paragrafo 1.3. Per questo motivo, prima di esaminare nel dettaglio l'implementazione del sistema, è opportuno illustrare come sia possibile utilizzare l'infrastruttura di coordinazione adottata nel linguaggio Java.

5.1. USARE TUCSON

TuCSoN è disponibile in un file .jar che contiene tutti i package necessari. Per poterlo utilizzare è necessario installare almeno un nodo, operazione che si ottiene semplicemente mettendo in esecu-

zione la classe `Node` del package `alice.tucson.runtime`. Occorre quindi invocare da console il comando:

```
java -cp dir/tucson.jar alice.tucson.runtime.Node
```

dove `dir` è il nome del direttorio in cui si trova il file `tucson.jar`. Includendo il percorso al file `.jar` nella variabile d'ambiente `CLASSPATH` si può evitare di specificare la direttiva `-cp`.

L'invocazione di questo comando crea un contesto in cui è possibile creare diversi centri di tuple ed avviare l'esecuzione degli agenti, ed il servizio è accessibile anche da remoto attraverso la porta 20504.

TuCSOn mette a disposizione anche un IDE che permette agli utenti di interagire direttamente con l'infrastruttura. In particolare, lo strumento `CLIAgent` permette di accedere direttamente ai centri di tuple e di depositare e prelevare tuple mediante le operazioni di `out`, `in`, `rd`, `inp` e `rdp` già descritte. Lo strumento `Inspector` consente invece di monitorare lo stato, le richieste pendenti, le reazioni e le specifiche di un centro di tuple. Questi strumenti sono molto utili in fase di debug.

I servizi di TuCSOn, naturalmente, sono accessibili anche da programmi Java. Anche in questo caso l'accesso non è affatto complicato. Il file `.jar` contiene i package `alice.tucson.api` e `alice.logictuple`, che mettono già a disposizione le classi necessarie: è sufficiente includerli.

Un agente Java è una istanza della classe `Agent`. Questa classe contiene tutti gli attributi ed i metodi necessari al funzionamento di un agente generico: per realizzare un agente dell'applicazione è quindi sufficiente scrivere la classe dell'agente che si vuole implementare, la quale dovrà ereditare da `Agent` e definire il metodo `body()`, che costituisce il corpo dell'agente. Per attivare un agente basta invocarne il metodo `spawn()`, che richiama il `body()`.

La classe `TupleCentreId` consente di creare un centro di tuple e di assegnargli un nome identificativo tramite il quale possa esse-

re reperito dagli agenti. Su un centro di tuple gli agenti possono depositare o prelevare tuple logiche, che sono istanze della classe `LogicTuple`. Una tupla logica è costituita da un nome e da una lista di argomenti (`TupleArgument`), che possono essere delle costanti (`Value`) o delle variabili (`Var`).

La classe `Agent` dispone dei metodi che consentono all'agente di accedere ad un centro di tuple: rispettivamente `out()`, `in()`, `rd()`, `inp()` e `rdp()`. Tutti questi metodi hanno come argomento l'identificativo del centro di tuple su cui si vuole effettuare l'operazione e la tupla logica che si vuole depositare o prelevare. Il comportamento di un centro di tuple può essere programmato da un agente tramite i metodi `getSpec()` e `setSpec()`, che danno accesso alle reazioni del centro di tuple espresse in linguaggio `ReSpecT`.

5.2. IMPLEMENTAZIONE DEL SISTEMA

Come si può vedere dall'architettura logica, è possibile prevedere che i singoli agenti, realizzati in modo tale da eseguire i task di loro competenza, possano essere attivati da una serie di pagine JSP che consentano agli utenti di accedere al sistema come ad un tradizionale sito Internet. Lo sviluppo della struttura ipertestuale dell'applicazione esula tuttavia dagli obiettivi di questa tesi, che si concentra invece sugli aspetti legati alla coordinazione degli agenti che vengono attivati.

In quest'ottica, è possibile notare che il sistema è costituito da tre società di agenti che lavorano su altrettanti centri di tuple. Pertanto è necessario realizzare uno strumento che operi la configurazione del sistema, creando i centri di tuple richiesti, programmandoli opportunamente ed eventualmente inserendovi alcune tuple di inizializzazione che possono ritenersi necessarie. Inoltre, tale

strumento dovrà mettere in esecuzione tutte le classi di infrastruttura che forniscono i servizi agli agenti.

Questo strumento dovrà quindi essere messo in esecuzione prima degli agenti dell'applicazione: per questo motivo si ritiene opportuno implementarlo all'esterno dell'applicazione web, come un componente a parte. Questa separazione è possibile grazie alla metodologia di progetto adottata: mantenere distinte le dimensioni di computazione e coordinazione consente infatti di progettare lo spazio di interazione a prescindere dagli agenti stessi, e viceversa.

5.3. CONFIGURAZIONE DEL SISTEMA

La creazione e la programmazione di un centro di tuple sono responsabilità di un oggetto apposito. È chiamato `TcConfigurer`, ed è realizzato come un agente di meta-livello.

Le specifiche di un centro di tuple `TuCSon` devono essere espresse in linguaggio `ReSpecT` e devono essere tali da garantire il rispetto delle leggi di coordinazione emerse dall'analisi. Si può notare facilmente che queste specifiche rivestono un'importanza fondamentale: da esse dipende infatti il comportamento dell'intero sistema. Purtroppo, come spesso accade soprattutto per sistemi web-based, il comportamento desiderato dagli utenti del sistema non è costante, ma è soggetto a frequenti cambiamenti che derivano dall'emergere di nuovi requisiti e necessità. Per questo motivo, si ritiene estremamente utile mantenere le specifiche `ReSpecT` separate dal codice Java, collocandole in un file a parte.

Questa soluzione comporta tre grandi vantaggi.

- In primo luogo evita di costringere l'oggetto `TcConfigurer` a conoscere il codice `ReSpecT` relativo al suo centro di tuple: è sufficiente che conosca il nome del file in cui è salvato per poterlo caricare direttamente dal disco. In questo modo l'oggetto diventa un componente riusabile, ed è possibile utilizzarlo per

tutti i centri di tuple.

- In secondo luogo comporta una maggiore pulizia del codice, dato che consente di scrivere il ReSpecT come qualsiasi altro linguaggio di programmazione, e non come una anonima stringa Java. Questo consente di concentrarsi maggiormente su di esso e trattarlo più efficacemente, modificarlo, sostituirlo e produrne copie e backup senza toccare il codice Java.
- Infine consente di caricare il codice ReSpecT a run-time, rendendolo un codice interpretato, e non compilato. In questo modo è possibile modificare il comportamento di un centro di tuple senza dover fermare l'applicazione per ricompilarla, caratteristica molto importante per una applicazione web-based.

Il codice ReSpecT dei tre centri di tuple del sistema si trova quindi in tre file di estensione .rsp; durante la configurazione vengono istanziati tre oggetti `TcConfigurer`, a ciascuno dei quali viene fornito il nome di uno di tali file.

5.3.1. Centro di tuple Esercitazioni

Questo centro di tuple rappresenta il mezzo di coordinazione della società di agenti relativa alle esercitazioni, e dovrà quindi integrare tutte le leggi di coordinazione di quella società.

Come si evince dal modello della società descritto nel paragrafo 4.2.2, tali leggi riguardano soprattutto la gestione dello stato delle esercitazioni e l'ordinamento delle attività del workflow.

Oltre a questo è possibile individuare altre regole. Ogni esercitazione ha una scadenza, oltre la quale non è consentito consegnare; inoltre non è consentito agli studenti consegnare più volte la stessa esercitazione.

Riassumendo, il centro di tuple dovrà:

1. registrare il testo di ogni nuova esercitazione inserita;
2. concedere o negare il permesso di effettuare la consegna operando le opportune verifiche;

3. registrare la consegna di una esercitazione da parte di uno studente autorizzato;
4. gestire correttamente la sequenza di operazioni da svolgere su una esercitazione consegnata facendo rispettare il workflow;
5. aggiornare lo stato di correzione dell'esercitazione ad ogni evento che la riguarda;
6. registrare la valutazione di ogni esercitazione.

Si può notare come il centro di tuple non debba solamente reagire ad eventi, ma anche registrare alcune informazioni, come l'elenco delle esercitazioni proposte dal docente e l'elenco degli studenti che le consegnano. A questo scopo si ritiene necessario che il centro mantenga una tupla contenente l'elenco delle esercitazioni proposte, e per ciascuna di esse una tupla contenente l'elenco degli studenti che la consegnano; queste tuple verranno inserite da un apposito agente di inizializzazione ed aggiornate ad ogni evento. Più in dettaglio, la prima tupla avrà la forma seguente:

```
elenco_esercitazioni(Num, [es(ID, Txt, Data), ...])
```

dove Num è il numero di esercitazioni nella lista, e per ogni esercitazione della lista viene riportato il codice, il testo e la data ultima di consegna. L'elenco delle consegne sarà invece nella forma seguente: `consegne(ID, Num, [Matr, ...])` dove ID è il codice dell'esercitazione, Num è il numero di studenti che l'hanno consegnata, e Matr il numero di matricola dell'i-esimo studente.

Il codice ReSpecT con cui viene programmato questo centro di tuple è riportato a titolo di esempio nell'Appendice A.

5.3.2. Centro di tuple Progetti

Questo centro di tuple governa l'interazione degli agenti della società relativa ai progetti. Le relative leggi di coordinazione sono simili a quelle delle esercitazioni, ma non è necessario tener traccia dello stato di correzione di un progetto, dal momento che costituisce una scelta più libera e quindi non è possibile prevedere

una correzione automatica.

Questo centro di tuple dovrà quindi:

1. registrare le proposte di progetto inserite dal docente;
2. registrare le richieste di commento presentate dagli studenti;
3. concedere o negare l'autorizzazione alla consegna della versione definitiva verificando che lo studente non l'abbia già consegnata;
4. registrare la consegna definitiva di ogni studente;
5. registrare ogni valutazione espressa dal docente.

Anche in questo caso il centro di tuple necessita di mantenere alcune informazioni sotto forma di tupla, che dovranno essere aggiornate dalle reazioni agli eventi in modo opportuno. È quindi necessario mantenere la tupla `elenco_proposte(Num, [Txt, ...])` che contenga l'elenco delle proposte di progetto presentate dal docente, la tupla `elenco_consegne(N, [Matr, ...])` che contenga l'elenco degli studenti che hanno consegnato la versione definitiva, ed un contatore per le richieste di commento pendenti.

5.3.3. Centro di tuple Esami

Questo centro di tuple governa l'interazione degli agenti del gruppo relativo all'esame. A differenza degli altri due centri di tuple, le operazioni effettuabili sono soggette a vincoli temporali molto restrittivi.

Lo svolgimento dell'esame tuttavia esula dagli obbiettivi di questa tesi: pertanto in una realizzazione prototipale è sufficiente modellare l'esame come un'attività che gli studenti svolgono in locale, consegnando i risultati finali mediante upload di file, come nel caso delle esercitazioni e dei progetti.

Le responsabilità di questo centro di tuple sono pertanto:

1. registrare le domande e la durata dell'esame;
2. registrare l'orario di inizio dell'esame di ogni studente, che coincide con il download delle domande, e calcolare il termine

- di consegna sommandovi la durata della prova;
3. concedere o negare l'autorizzazione al salvataggio parziale delle risposte, verificando che lo studente non abbia già effettuato la consegna definitiva e che non sia scaduto il termine di consegna;
 4. concedere o negare l'autorizzazione alla consegna definitiva dell'esame, operando le medesime verifiche del punto 3;
 5. registrare la consegna finale dello studente verificando che non sia terminata oltre la scadenza;
 6. registrare la valutazione dell'esame di ogni studente;
 7. registrare il voto finale assegnato ad ogni studente in base alla sua carriera complessiva.

Per poter effettuare tutte le verifiche è sufficiente che l'Esaminatore depositi una tupla `esame(Url,tempo(Ore,Min))` indicante il file delle domande e la durata dell'esame. Ogni volta che un Esaminando esegue il download delle domande viene inserita la tupla `fine_esame(matricola,tempo(ore,min))` che registra il termine di consegna per quello studente.

Anche in questo caso, non è detto che tutti gli esaminandi riescano ad effettuare una consegna finale, pertanto viene memorizzato nella tupla `consegne(Num, [Matr,...])` l'elenco degli studenti che hanno consegnato correttamente l'esame.

5.4. CLASSI DI INFRASTRUTTURA

Come si può dedurre dall'analisi, i centri di tuple, pur opportunamente programmati, non esauriscono l'infrastruttura necessaria all'applicazione che si sta sviluppando: gli agenti del sistema necessitano di risorse che devono far parte del contesto in cui operano gli agenti. Tali risorse corrispondono alle classi di infrastruttura individuate in fase di progetto: Gestore Login e Gestore Dati.

5.4.1. Scelte implementative

La soluzione apparentemente più semplice per la realizzazione di questi oggetti consiste nel vederli come entità puramente passive che entrano nel raggio d'azione degli agenti. Tuttavia questa soluzione non è la più efficace, in quanto comporta un problema rilevante: se le risorse fossero oggetti passivi, infatti, gli agenti potrebbero accedervi simultaneamente, e questo potrebbe causare interferenze. Il problema potrebbe essere risolto prevedendo un meccanismo di sincronizzazione, ma questo significherebbe limitare l'autonomia degli agenti e caricarli della responsabilità di rispettare una legge di coordinazione.

Una soluzione migliore consiste nello scaricare la sincronizzazione necessaria sull'infrastruttura di coordinazione. Le risorse vengono quindi realizzate come agenti particolari, il cui compito è quello di restare in attesa di richieste di servizio e processarle una per volta. Tali agenti di infrastruttura operano quindi come demoni sempre attivi, in attesa di una tupla di comando che rappresenta il servizio richiesto. Se il servizio richiesto comporta un risultato, esso sarà restituito nuovamente sotto forma di tupla logica.

È facile notare che anche questi agenti possono essere attivati una volta per tutte prima dell'attivazione degli altri agenti del sistema. Questa soluzione comporta un ulteriore vantaggio: gli agenti non sono costretti a conoscere il fornitore del servizio, che potrebbe quindi essere anche replicato per aumentare la scalabilità del sistema. Inoltre, non è più necessario che le risorse vengano allocate in un'area di memoria accessibile agli agenti, ma potrebbero essere allocate anche in remoto senza bisogno di particolari artifici.

Occorre notare tuttavia che un agente può essere sospeso in attesa di un'unica tupla alla volta. Questo però non impedisce ad un agente di infrastruttura di attendere una qualsiasi richiesta di servizio di sua competenza: è sufficiente che la richiesta di servizio sia l'argomento di una tupla, e non la tupla stessa. In questo modo

l'agente può fare semplicemente una `in` di una tupla del tipo:

```
richiesta(Tipo_servizio, [Parametri_eventuali]).
```

Questo tuttavia comporta due piccoli problemi. Anzitutto rischia di vincolare eccessivamente i protocolli di accesso, che ovviamente devono essere noti agli agenti. In secondo luogo, le risorse necessarie al sistema sono diverse, e la tupla dovrebbe quindi poter distinguere univocamente il destinatario della richiesta, ad esempio facendo in modo che i diversi agenti di infrastruttura attendano tuple con nomi differenti. In questo modo però gli agenti che richiedono un servizio sono costretti a conoscere quali e quanti sono i gestori disponibili, cosa che può limitare la flessibilità del sistema complessivo.

In seguito a queste considerazioni si ritiene opportuno lasciare che gli agenti eseguano le richieste di servizio tramite gli stessi protocolli che emergono dal modello dell'ambiente prodotto in fase di progetto, e lasciare al mezzo di coordinazione il compito di tradurre le richieste nella tupla attesa dal fornitore del servizio.

Per questo motivo viene introdotto un quarto centro di tuple, chiamato per convenzione “servizi”, su cui gli agenti di infrastruttura rimangono in attesa di richieste di servizio. Tale centro di tuple verrà creato e programmato assieme agli altri tre, prima dell'attivazione degli agenti.

Un altro aspetto che è possibile notare è che le risorse di questa applicazione hanno una caratteristica in comune: la maggior parte dei servizi che offrono consiste nel fornire o memorizzare informazioni. Per questo motivo è opportuno realizzare una base di dati a cui possano accedere, prima di sviluppare i singoli gestori.

5.4.2. Implementazione della base di dati

La struttura concettuale della base di dati del sistema era già stata introdotta alla fine della fase di progetto, nel paragrafo 4.4, mentre in fase di implementazione si è poi deciso di mantenere tutte le

informazioni relative alle esercitazioni, ai progetti e agli esami nei corrispondenti centri di tuple. Questo poteva far sembrare inutile l'uso di un database, ma il sistema ha bisogno di gestire alcune entità che non riguardano direttamente le tre società di agenti, come gli account degli utenti e le altre informazioni di utilità previste dal sistema. Per consentire la gestione di questi dati si ritiene opportuno sviluppare un piccolo database.

Per non vincolare il sistema all'uso di una particolare tecnologia si è scelto di effettuare un accesso standardizzato, cioè via JDBC. In questo modo il sistema può essere collegato a qualsiasi sorgente dati ODBC, semplicemente caricando il famoso driver `sun.jdbc.odbc.JdbcOdbcDriver` della Sun.

Per la realizzazione prototipale è stato quindi creato un piccolo database con il programma Microsoft® Access. La sua struttura relazionale è indicata in figura 5.1.

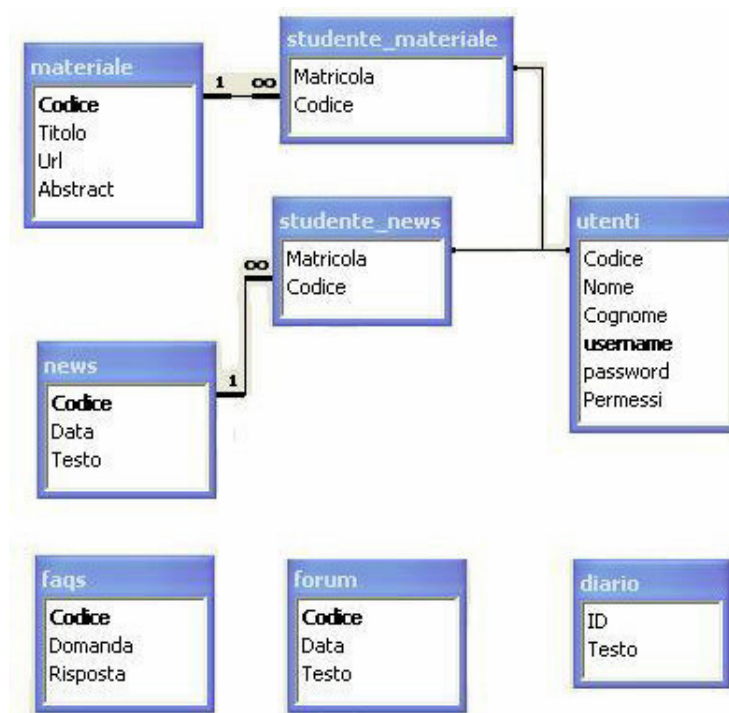


Fig. 5.1 – Diagramma E-R fisico

Si può notare che la tabella *utenti*, che raccoglie le informazioni anagrafiche, l'account e i permessi di accesso di un utente, non abbia come chiave primaria il Codice, cioè la matricola dello studente, bensì l'account stesso. È possibile infatti che un utente possa accedere al sistema con differenti permessi di accesso, e quindi due account differenti. È il caso, ad esempio, dello studente che sostiene l'esame: in quel caso non gli è consentito accedere al materiale didattico e ad altre informazioni che potrebbero facilitarlo, quindi è necessario che si identifichi con una diversa chiave d'accesso, in quanto accede con permessi differenti.

Le tabelle *studenti_materiale* e *studenti_news* permettono poi di memorizzare le associazioni tra gli studenti e il materiale che hanno scaricato e le news che hanno letto.

5.4.3. Gestore Login

Questo agente di servizio ha il compito di identificare un utente associandogli i permessi di accesso. Tali permessi sono individuati dalla categoria dell'utente, che può essere Docente, Studente, o Esaminando.

Quando riceve la tupla `login(Username, Password)` accede al database; se la query ha successo l'account è corretto e il gestore risponde con una tupla contenente tutti i dati dell'utente, altrimenti la tupla di risposta conterrà valori fittizi che rappresentano un errore, e l'accesso viene negato.

Per maggiore sicurezza, prima di concedere l'accesso ad uno studente il Gestore Login accede al centro di tuple degli esami, per verificare che lo studente non stia sostenendo l'esame. In questo caso non gli consente l'accesso.

5.4.4. Gestore Dati

Questa risorsa ha il compito di fornire agli agenti un accesso quasi completo alle diverse entità della base di dati. Per questo motivo, sia per dare alla risorsa una maggiore modularità sia per evitare di renderla un collo di bottiglia, si ritiene opportuno suddividerla in diversi agenti di infrastruttura, ciascuno dei quali avrà la responsabilità di accedere ai dati che gli competono. Sarà compito del centro di tuple smistare correttamente le richieste in arrivo.

L'accesso alla base di dati viene dunque fornito da tre agenti di infrastruttura: Gestore Materiale, Gestore News e Gestore Varie, rispettivamente.

Il Gestore Materiale ha il compito di:

- registrare la pubblicazione di nuovo materiale didattico,
- fornire l'elenco completo del materiale didattico pubblicato,
- verificare se è stato pubblicato del materiale che lo studente corrente non ha ancora scaricato,
- memorizzare il materiale che lo studente scarica.

Il Gestore News ha gli stessi compiti del Gestore Materiale, ma relativi alle news che possono essere pubblicate dal docente e lette dagli studenti.

Il Gestore Varie, infine, ha il compito di accedere alla base di dati per le informazioni relative alle FAQ, al Forum e al diario delle lezioni.

5.5. CONCLUSIONI

A mio parere, la metodologia adottata ha consentito di realizzare l'applicazione in maniera molto efficace. La separazione della dimensione della coordinazione ha infatti consentito di progettare e realizzare tutta l'infrastruttura come un componente a sé stante, e il progetto dei singoli agenti non può che essere facilitato, in quanto ogni agente potrà essere costruito concentrandosi unica-

mente sulla sua attività.

Inoltre, ritengo possibile apportare facilmente future modifiche all'applicazione, dal momento che la sua architettura è estremamente modulare e flessibile. Grazie all'adozione di un modello orientato agli agenti possono infatti essere introdotte nuove funzionalità semplicemente inserendo nuovi agenti, e si possono modificare le leggi di coordinazione anche dinamicamente, senza dover riavviare il sistema.

In base a queste osservazioni ritengo che questo modo di procedere possa essere adottato vantaggiosamente in quasi tutti i settori dell'informatica, ed azzardo l'ipotesi di un futuro non troppo remoto in cui complessi sistemi software orientati agli agenti forniscano alla gente servizi attualmente inimmaginabili.

APPENDICE A

In questa appendice riporto a titolo di esempio il codice ReSpecT con cui viene programmato il centro di tuple relativo alle esercitazioni.

%%%%%%%%--- Inserimento nuova esercitazione ---%%%%%%%%

```
reaction(out(nuova_esercitazione(Codice, Testo, Data)),(
  in_r( nuova_esercitazione(Codice, Testo, Data) ),
  in_r( nuova_esercitazione( C, T, D ) ),
  out_r( nuova_esercitazione(Codice, Testo, Data) )
) ).
reaction(out_r(nuova_esercitazione(Codice, Testo, Data)), (
  in_r( elenco_esercitazioni( Num, Lista ) ),
  Num2 is Num+1,
  out_r(elenco_esercitazioni(Num2, [ es(Codice, Testo, Data, 0) | Lista ] )),
  out_r( consegne( Codice, 0, [] ) )
) ).
```

%%%%%%%%--- richiesta permesso di consegna ---%%%%%%%%

```
reaction( out( consegna( Codice, Studente) ), (
  in_r( consegna( Codice, Studente) ),
  rd_r( nuova_esercitazione( Codice, _ , _ ) ),
  no_r( statoEs( Codice, Studente, _ ) ),
  out_r(abilitazione_consegna( Codice, Studente, 'ok' ) )
) ).
reaction( out( consegna( Codice, Studente) ), (
  in_r( consegna( Codice, Studente) ),
  rd_r( nuova_esercitazione( Codice, _ , _ ) ),
  rd_r( statoEs( Codice, Studente, _ ) ),
  rd_r( richiesta_reinvio( Codice, Studente, _ ) ),
  out_r(abilitazione_consegna( Codice, Studente, 'ok' ) )
) ).
```

```
reaction( out( consegna( Codice, Studente) ), (
  in_r( consegna( Codice, Studente) ),
  rd_r( nuova_esercitazione( Codice, _ , _ ) ),
  rd_r( statoEs( Codice, Studente, _ ) ),
  no_r( richiesta_reinvio( Codice, Studente, _ ) ),
```

```

    out_r(abilitazione_consegna(Codice, Studente, 'esercitazione già conse-
gnata' ))
  ) ).
reaction( out( consegna( Codice, Studente) ), (
  in_r( consegna( Codice, Studente) ),
  no_r( nuova_esercitazione( Codice, _ , _ ) ),
  out_r(abilitazione_consegna(Codice, Studente, 'scaduto il termine di con-
segna' ) )
) ).
%%%--- notifica di fine consegna ---%%%
reaction( out(fine(consegna_eser,Es,Stud)) , (
  in_r(fine(consegna_eser,Es,Stud)),
  no_r( richiesta_reinvio( Es, Stud, _ ) ),
  in_r( elenco_esercitazioni( NE, [ es( Es, T, D, NS ) | Tail ] ) ),
  NSN is NS+1,
  out_r( elenco_esercitazioni( NE, [ es(Es, T, D, NSN) | Tail ] ) ),
  in_r( consegne( Es, NumC, Lista ) ),
  N2 is NumC+1,
  out_r( consegne( Es, N2, [Stud | Lista] ) ),
  out_r(statoEs(Es, Stud, 'consegnata')),
  out_r(inizio(controllo_form, Es, Stud))
) ).
reaction( out(fine(consegna_eser,Es,Stud)) , (
  in_r(fine(consegna_eser,Es,Stud)),
  rd_r( richiesta_reinvio( Es, Stud, _ ) ),
  in_r(statoEs(Es, Stud, _ )),
  out_r(statoEs(Es, Stud, 'consegnata')),
  out_r(inizio(controllo_form,Es,Stud))
) ).

%%%--- gestione del workflow ---%%%

reaction( in(inizio(controllo_form,Es,Stud)) , (
  in_r(statoEs(Es,Stud,_)),
  out_r(statoEs(Es,Stud, 'controllo formato in corso'))
) ).
reaction( out( richiesta_reinvio(Es, Stud, Motivo) ) , (
  in_r(statoEs(Es, Stud, _ )),
  out_r(statoEs(Es, Stud, Motivo))
) ).

```

```
reaction( out(fine(controllo_form,Es,Stud)) , (
    in_r(fine(controllo_form,Es,Stud)),
    in_r(statoEs(Es,Stud,_)),
    out_r(statoEs(Es,Stud, 'controllato formato')),
    out_r(inizio(controllo_origin,Es,Stud))
) ).

reaction( in(inizio(controllo_origin,Es,Stud)) , (
    in_r(statoEs(Es,Stud,_)),
    out_r(statoEs(Es,Stud, 'controllo originalità in corso'))
) ).

reaction( out(fine(controllo_origin,Es,Stud)) , (
    in_r(fine(controllo_origin,Es,Stud)),
    in_r(statoEs(Es,Stud,_)),
    out_r(statoEs(Es,Stud,'da testare')),
    out_r(inizio(test,Es,Stud))
) ).

reaction( in(inizio(test,Es,Stud)) , (
    in_r(statoEs(Es,Stud,_)),
    out_r(statoEs(Es,Stud,'test in corso'))
) ).

reaction( out(fine(test,Es,Stud)) , (
    in_r(fine(test,Es,Stud)),
    in_r(statoEs(Es,Stud,_)),
    out_r(statoEs(Es,Stud,'da correggere')),
    out_r(inizio(correzione,Es,Stud))
) ).

reaction( in(inizio(correzione,Es,Stud)) , (
    in_r(statoEs(Es,Stud,_)),
    out_r(statoEs(Es,Stud,'correzione in corso'))
) ).

reaction( out(valutazione(Es,Stud,Voto)) , (
    in_r(statoEs(Es,Stud,_)),
    out_r(statoEs(Es,Stud,'corretta'))
) ).
```

BIBLIOGRAFIA

- [1]. P. Ciancarini, A. Omicini, F. Zambonelli, *Coordination technologies for Internet agents*, Nordic Journal of Computing 6 (1999), pp. 215–240
- [2]. P. Ciancarini, A. Omicini, F. Zambonelli, *Multiagent system engineering: The coordination viewpoint*, in: N. R. Jennings and Y. Lespérance, editors, Intelligent Agents VI. Agent Theories, Architectures, and Languages, LNAI 1757 (2000), pp. 250–259, 6th International Workshop (ATAL'99), Orlando (FL), 15–17 July 1999, Proceedings.
- [3]. E. Denti, A. Omicini, A. Ricci. *Coordination tools for the development of agent-based systems*. Applied Artificial Intelligence, 16(9), Oct. 2002.
- [4]. E. Denti, A. Ricci, R. Rubino. *Integrating and Orchestrating Services upon an Agent Coordination Infrastructure*.
- [5]. D. Gelernter, N. Carriero, *Coordination languages and their significance*, Comm. ACM 35 (2) (1992) 97–107.
- [6]. N. R. Jennings. *An Agent-based approach for building complex software systems*. Communication of the ACM vol 4, (2001) pp. 35–41
- [7]. A. Omicini. *SODA: Societies and infrastructures in the analysis and design of agent-based systems*, in: Ciancarini and Wooldridge, pp. 185–193.
- [8]. A. Omicini. *Towards a notion of agent coordination context*. In D. Marinescu and C. Lee, editors, Process Coordination and Ubiquitous Computing, pp. 187–200. CRC Press, 2002.
- [9]. A. Omicini, E. Denti. *Formal ReSpecT*. In A. Dovier, M. C. Meo, and A. Omicini, eds, Declarative Programming – Selected Papers from AGP'00, volume 48 of Electronic Notes in Theoretical Computer Science, pp. 179–196. Elsevier Science B. V., 2001.

- [10]. A. Omicini, E. Denti. *From tuple spaces to tuple centres*. Science of Computer Programming, 41(3):277–294, Nov. 2001.
- [11]. A. Omicini, A. Ricci. *Integrating Organisation within a MAS Coordination Infrastructure*.
- [12]. A. Omicini, A. Ricci. *Reasoning about Organisation: Shaping the Infrastructure*.
- [13]. A. Omicini, F. Zambonelli. *Coordination for Internet application development*, Autonomous Agents and Multi-Agent Systems 2 (1999), pp. 251–269.
- [14]. A. Ricci, A. Omicini, E. Denti. *Activity Theory as a framework for MAS coordination*. In: P. Petta, R. Tolksdorf, F. Zambonelli, eds.: Engineering Societies in the Agents World III. Volume 2577 of LNCS. Springer-Verlag (2003) pp. 96-110
- [15]. A. Ricci, A. Omicini, E. Denti. *Virtual enterprises and workflow management as agent coordination issues*. International Journal of Cooperative Information Systems, 11(3/4):355–379, Sept./Dec. 2002.
- [16]. The aliCE research team – DEIS, Università di Bologna a Cesena. *TuCSon guide*, 2002
- [17]. The aliCE research team – DEIS, Università di Bologna a Cesena. *TuCSon Documentation*, 2002
- [18]. M. Wooldridge, N. R. Jennings. *Intelligent Agents: Theory and Practice*. Knowledge Engineering Review (1994)
- [19]. F. Zambonelli, N.R. Jennings, A. Omicini, M.J. Wooldridge. *Agent-Oriented Software Engineering for Internet Applications*, chapter 13. In Omicini et al., December 2000.

INDICE

INTRODUZIONE	1
CAPITOLO 1: LA COORDINAZIONE NEI SISTEMI MULTI-AGENTE	4
1.1. L'IMPORTANZA DELLA COORDINAZIONE	4
1.1.1. <i>Progettare lo spazio di coordinazione di un sistema multi-agente</i>	7
1.2. MODELLI E LINGUAGGI DI COORDINAZIONE.....	8
1.2.1. <i>Modelli di coordinazione control-driven: comunicazione diretta</i>	9
1.2.2. <i>Modelli di coordinazione data-driven: Linda</i>	10
1.2.3. <i>Possibili implementazioni del modello</i>	12
1.2.4. <i>Limiti del modello</i>	14
1.2.5. <i>Modelli di coordinazione ibridi: il Centro di Tuple</i>	15
1.2.6. <i>Programmare un centro di tuple: il linguaggio ReSpecT</i>	17
1.3. INFRASTRUTTURE DI COORDINAZIONE: TUCSON	19
1.3.1. <i>Il linguaggio di coordinazione</i>	21
1.3.2. <i>Il mezzo di coordinazione</i>	22
1.4. UN TIPICO PROBLEMA DI COORDINAZIONE: LA GESTIONE DEI WORKFLOW	22
CAPITOLO 2: METODOLOGIA DI PROGETTO AGENT-ORIENTED	26
2.1. L'IMPORTANZA DI UNA METODOLOGIA AGENT-ORIENTED	26
2.2. LA METODOLOGIA SODA.....	27
2.2.1. <i>La fase di Analisi</i>	28
2.2.1.1. <i>Modello dei ruoli</i>	28
2.2.1.2. <i>Modello delle Risorse</i>	29
2.2.1.3. <i>Modello di Interazione</i>	29
2.2.2. <i>La fase di Progetto</i>	29
2.2.2.1. <i>Modello degli Agenti</i>	30
2.2.2.2. <i>Modello della Società</i>	30
2.2.2.3. <i>Modello dell'Ambiente</i>	31
CAPITOLO 3: ANALISI DEL SISTEMA IN ESAME.....	32
3.1. L'APPLICAZIONE.....	32
3.2. ANALISI DEI REQUISITI	33
3.3. CASI D'USO	36
3.3.1. <i>Casi d'uso dell'attore Docente</i>	36
3.3.2. <i>Casi d'uso dell'attore Studente</i>	50
3.3.3. <i>Casi d'uso dell'attore Tutor</i>	58
3.3.4. <i>Casi d'uso dell'attore Amministratore</i>	59
3.3.5. <i>Workflow Individuati</i>	60

CAPITOLO 4: APPLICAZIONE DELLA METODOLOGIA SODA NEL PROGETTO DEL SISTEMA	62
4.1. FASE DI ANALISI	62
4.1.1. <i>Modello dei ruoli</i>	63
4.1.2. <i>Modello delle risorse</i>	67
4.1.3. <i>Modello di interazione</i>	68
4.2. FASE DI PROGETTO	72
4.2.1. <i>Modello degli Agenti</i>	73
4.2.2. <i>Modello della Società</i>	79
4.2.3. <i>Modello dell'Ambiente</i>	81
4.3. ARCHITETTURA DEL SISTEMA	82
4.3.1. <i>Diagramma delle classi</i>	83
4.3.2. <i>Diagrammi di sequenza</i>	85
4.4. PROGETTO DELLA BASE DI DATI	89
CAPITOLO 5: IMPLEMENTAZIONE DEL SISTEMA	91
5.1. USARE TUCSON	91
5.2. IMPLEMENTAZIONE DEL SISTEMA	93
5.3. CONFIGURAZIONE DEL SISTEMA	94
5.3.1. <i>Centro di tuple Esercitazioni</i>	95
5.3.2. <i>Centro di tuple Progetti</i>	96
5.3.3. <i>Centro di tuple Esami</i>	97
5.4. CLASSI DI INFRASTRUTTURA	98
5.4.1. <i>Scelte implementative</i>	99
5.4.2. <i>Implementazione della base di dati</i>	100
5.4.3. <i>Gestore Login</i>	102
5.4.4. <i>Gestore Dati</i>	102
5.5. CONCLUSIONI	103
APPENDICE A	105
BIBLIOGRAFIA	108
INDICE	110