Cupido

Relazione del progetto per il
corso di Architetture Software 2010/2011

Autori:

Lorenzo Belli	Mat. 602031
Marco Poletti	Mat. 604100
Federico Viscomi	Mat. 577326

Introduzione

Questo documento descrive come abbiamo realizzato il progetto per il corso Architetture software 2010/2011 presso l'università di Bologna. Il progetto realizzato è un applicazione web denominata Cupido.

Fase preliminare

La prima fase della realizzazione è stata un'analisi preliminare delle richieste del committente disponibili a [1]; al termine di questa fase abbiamo individuato come caratteristica rilevante che i requisiti del software saranno molto stabili perché le richieste del committente non cambieranno in modo sensibile durante lo sviluppo.

Scelta del processo di sviluppo

In seguito abbiamo scelto il processo di sviluppo da adottare per la realizzazione del software. Abbiamo considerato vari processi di sviluppo e i relativi pro e contro:

- ***Processo a cascata***

Il processo a cascata prevede: analisi dei requisiti, progettazione, codifica, e testing; tutti in successione.
Abbiamo scartato questo processo perché e' troppo rigido: risulterebbe troppo costoso in termini di tempo rimediare ad eventuali errori commessi nelle prime fasi del processo.

- ***Processo iterativo-incrementale***

Un processo iterativo-incrementale consiste di varie iterazioni, ognuna delle quali comprende fasi di: analisi dei requisiti, progettazione, codifica e testing. Ogni iterazione ha come risultato finale un prodotto testato con un numero di funzionalità aumentato rispetto all'iterazione precedente.

  - Pro: è più semplice risolvere errori introdotti nelle fasi iniziali del processo; il testing è introdotto in fasi precedenti rispetto al modello a cascata.
  - Contro: e più complesso pianificare il lavoro.

Abbiamo scartato questo processo poiché dall'analisi preliminare si evince che il nostro software avrà requisiti molto stabili e quindi e' inutile e forse dannoso suddividere le attività di analisi dei requisiti e di progettazione.

- ***Processo a Spirale***

Il processo a spirale e' iterativo e ogni iterazione comprende: analisi dei requisiti, progettazione, codifica e testing; si sfrutta la pianificazione e l'analisi del rischio per monitorare l'andamento del processo.
Abbiamo scartato questo processo poiché in relazione all'analisi preliminare del software riteniamo poco rilevante la pianificazione e l'analisi del rischio ad ogni iterazione.

- ***Processo a consegne programmate***

Un processo a consegne programmate ha come fasi iniziali analisi dei requisiti e progettazione, in seguito il processo prevede varie iterazioni di codifica e testing. Viene programmato in anticipo il lavoro da svolgere in ogni iterazione.
Abbiamo scartato questo processo perché non ci riteniamo in grado di programmare adeguatamente il lavoro da svolgere nelle varie iterazioni.

- ***Processi Agili***

I processi agili sono pensati per adattarsi ai cambiamenti dei requisiti in corso di sviluppo. Abbiamo scartato questo tipo di processi perché nel nostro caso i requisiti sono stabili.

- ***V-Model***

Processo a cascata, in cui a lavoro ultimato viene testato il risultato di ogni fase in ordine inverso. Abbiamo scartato questo processo poiché gli errori introdotti nella fase di analisi e progettazione vengono rilevati solo alla fine del processo, quando risulterebbe troppo oneroso correggerli.

- ***RUP***
    - Pro: enfasi su una documentazione accurata; risolve in modo pro-attivo i problemi legati al rischio di progetto; alto riuso di componenti.

Abbiamo scartato questo processo perché:

1. presuppone che il team sia esperto nel dominio del software da realizzare
2. e' troppo complesso
3. nel nostro caso non ci sono possibilita' di riuso di componenti
4. non siamo interessati alle analisi del rischio.

Descrizione del processo di sviluppo scelto

Nessuno dei processi di sviluppo presi in considerazinoe era soddisfacente quindi abbiamo scelto di adottare un processo di sviluppo ad hoc descritto nella Figura 1. Tale processo consiste delle seguenti fasi:

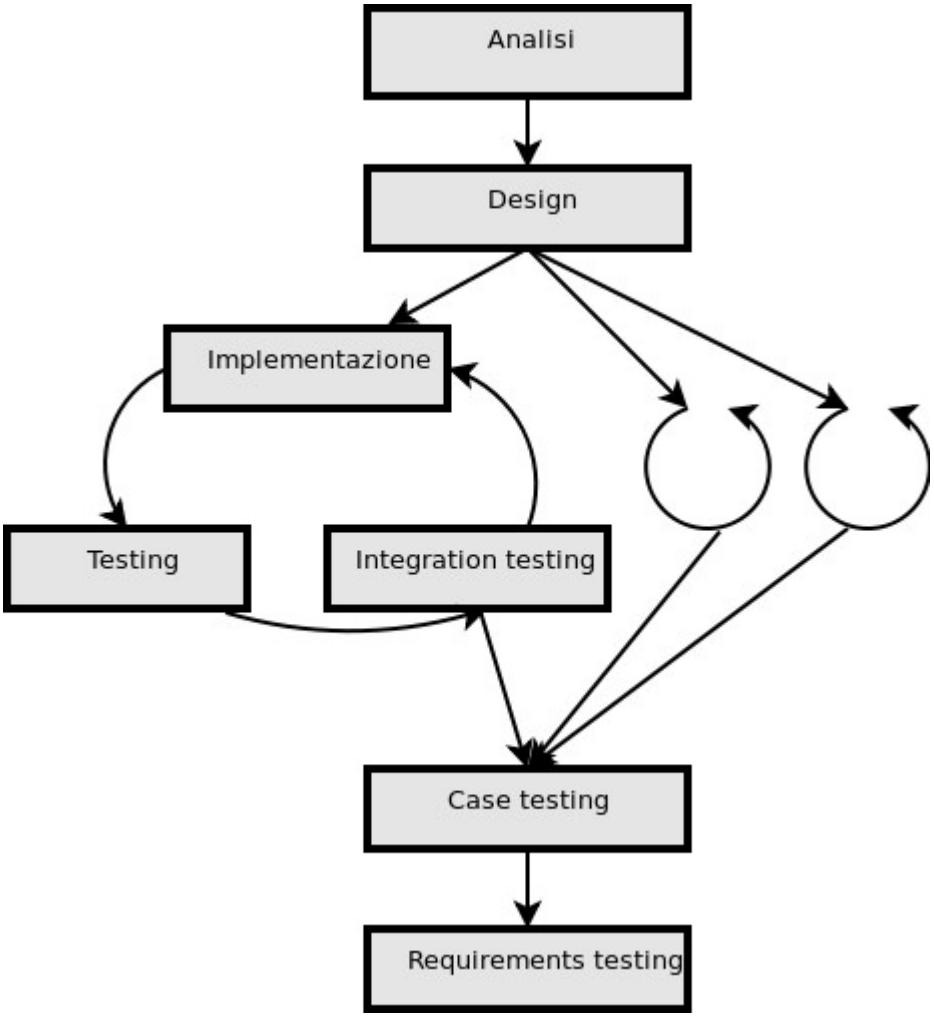


Figura 1: processo di sviluppo

- ***Analisi dei requisiti***

Questa fase deve produrre un documento di specifica dei requisiti del software.

- ***Design***

Questa fase consiste nella progettazione dell'architettura del sistema con un livello di dettaglio tale da permettere di dividere lo sviluppo in modo indipendente tra gli sviluppatori. Questa fase deve produrre: il documento che descrive l'architettura del software, le interfacce java tra i componenti del software.

- ***Iterazioni di sviluppo***

Questa fase consiste nell'implementazione del software. Il carico di lavoro si divide in tre parti ognuna da assegnare ad uno sviluppatore diverso. Ogni sviluppatore puo' a propria discrezione pianificare le proprie iterazioni e decidere la documentazione da produrre sulla parte di software da lui implementata.
Ogni iterazione consiste di tre fasi:

1. Implementazione: in questa fase lo sviluppatore implementa una parte del software.
2. Testing: in questa fase lo sviluppatore testa le funzionalità aggiunte nella fase precedente.
3. Integration Testing: in questa fase lo sviluppatore integra ciò che ha creato nell'iterazione

corrente con tutto il resto del software. Inoltre verifica che le interazioni fra i componenti dopo l'integrazione siano corrette.

Quando durante un iterazione uno sviluppatore identifica errori o mancanze derivanti dalle fasi di analisi o design, tutto il gruppo si riunisce per risolvere il problema; all'occorrenza si modifica la Specifica dei Requisiti Software, l'architettura ed il progetto. La correzione del problema sarà poi affidata ad uno o più degli sviluppatori.

- ***Case testing***

Questa fase consiste nella definizione di alcuni casi di test e nell'esecuzione di essi. Quando saranno rilevati errori, chi ha sviluppato la parte interessata avrà il compito di risolverli.

- ***Requirements Testing***

Questa fase consiste nella verifica che il software sia consistente con la specifica dei requisiti. In caso di problemi questi saranno risolti dagli sviluppatori interessati ed il software dovrà ripetere le fasi di Case Testing e Requirements Testing.
Il software che sia approvato da questa fase, sarà considerato completato.

Vantaggi del processo scelto:

- Tutto il gruppo parteciperà alla fase di progettazione, dunque ognuno di noi sarà consapevole del progetto dell'intero sistema.
- Ogni membro del gruppo dovrà conoscere l'implementazione solo della parte di software di sua competenza.
- Le iterazioni non pianificate si adattano meglio agli imprevisti dei singoli team di sviluppo e alla risoluzione degli errori.
- Non è necessario che i membri del gruppo siano esperti in un particolare dominio software.
- I Test Case sono sviluppati dopo l'implementazione, quando la conoscenza del software da parte del gruppo sarà migliore; questo porterà a Test Case più accurati e precisi.
- Le fasi di Analisi e Design sono uniche e non ripetute in accordo con la stabilità dei requisiti.

Svantaggi:

- Gli errori introdotti in fasi di Analisi e Design possono essere identificati molto tardi nello sviluppo del software, e la loro correzione richiede la presenza di tutto il gruppo.

Realizzazione

Dopo aver scelto il processo di sviluppo lo abbiamo eseguito.

Fase di analisi dei requisiti

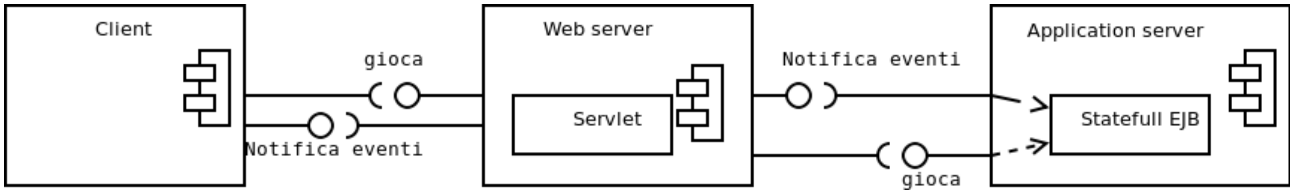
La fase di Analisi ha dato come risultato il file SRS.pdf.

Fase di progettazione dell'architettura

La fase di Design ha dato come risultato il file architettura_e_design.pdf che descrive la sola architettura scelta.
[TODO architetture scartate. Parte carente, c'è altro da inserire?]
Riportiamo di seguito le operazioni che abbiamo eseguito per arrivare a definire l'architettura software.

Java EE ed EJB

Abbiamo subito ipotizzato un architettura basata su Java EE con utilizzo di EJB per implementare la logica di gioco. L'uso di EJB permetterebbe di non dover implementare alcune caratteristiche come la gestione della concorrenza.



In questo caso una partita dovrebbe essere gestita da un Stateful Session Bean, perché lo stato della partita deve essere sempre disponibile. Tuttavia lo stato di una partita è determinato dall'interazione di più utenti che interagiscono con il Session EJB tramite servlet; questa architettura è scorretta poiché un Stateful Session Bean non può essere condiviso tra più client.

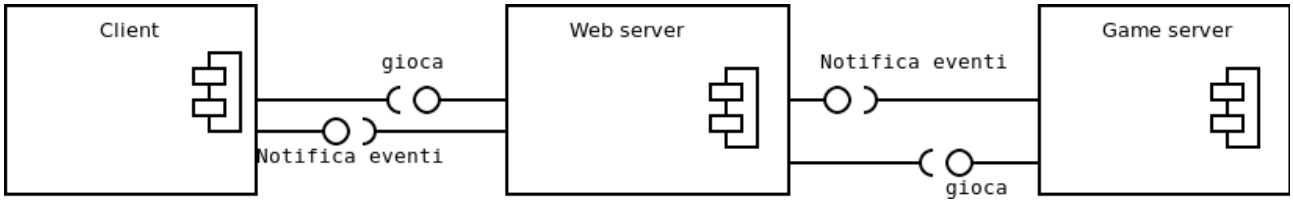
Una seconda possibilità per utilizzare EJB è di creare un componente Table che gestisce una singola partita, ed accedervi tramite Stateless EJB. In questo modo però bisognerebbe specificare ad ogni invocazione a quale partita si sta partecipando. Questo introdurrebbe un livello di astrazione poco utile, una servlet infatti potrebbe comunicare direttamente con una componente Table tramite java-rmi.

Decidiamo dunque di distogliere lo sguardo da EJB perché non ci interessano particolarmente le caratteristiche di transazionalità e sicurezza che ci fornisce.

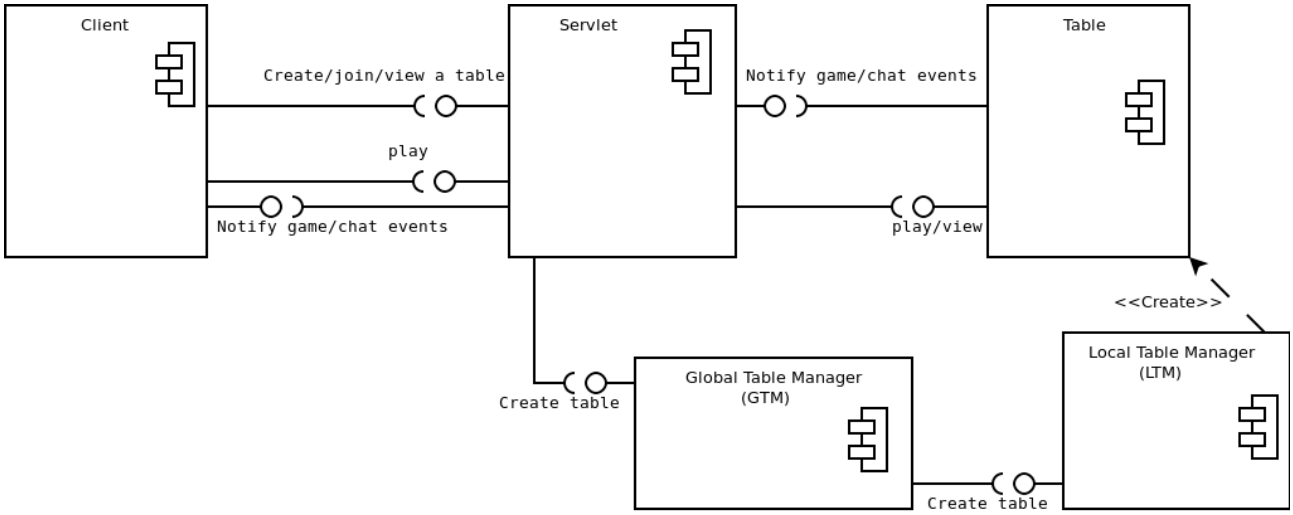
Un'altra opzione valutata è di utilizzare una tecnologia Web Service per implementare la comunicazione tra servlet e Application Server. Questa opzione però non è adatta data la natura stateless dell'interazione.

GWT ed RMI

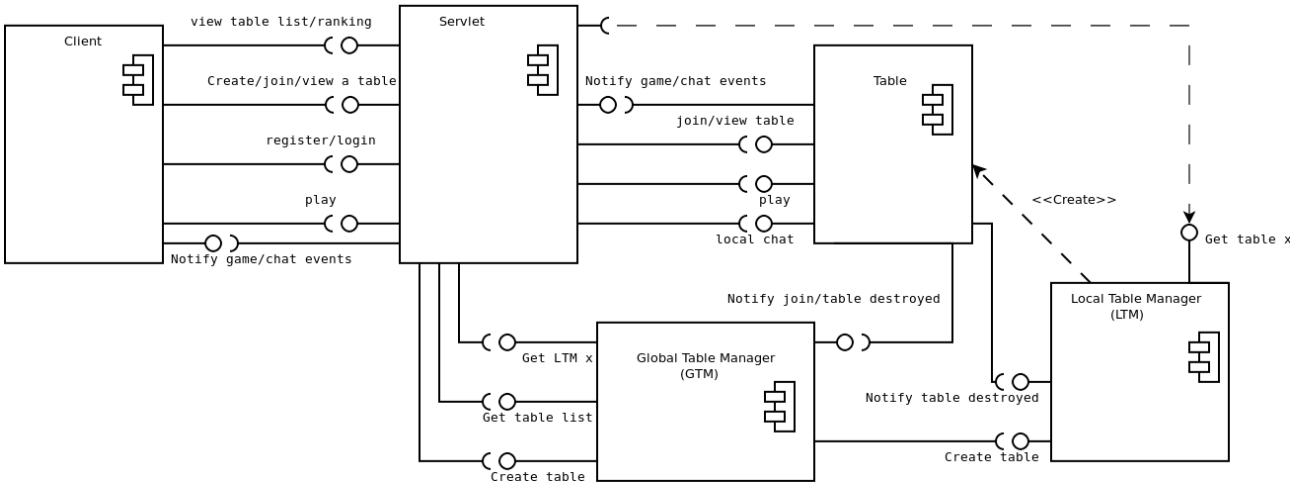
Prevediamo ora di dividere il software in tre componenti principali: un backend che gestisce la logica di gioco, un client eseguito dall'utente e da un web server che gestisce la comunicazione tra i due tramite servlet.



Qui ogni web server gestisce più servlet che gestiscono più client. Tutte le servlet comunicano con un unico game server.
Avere un unico Game server può essere un collo di bottiglia, quindi per sopperire alle richieste di scalabilità decidiamo di duplicare i game server su più macchine fisiche

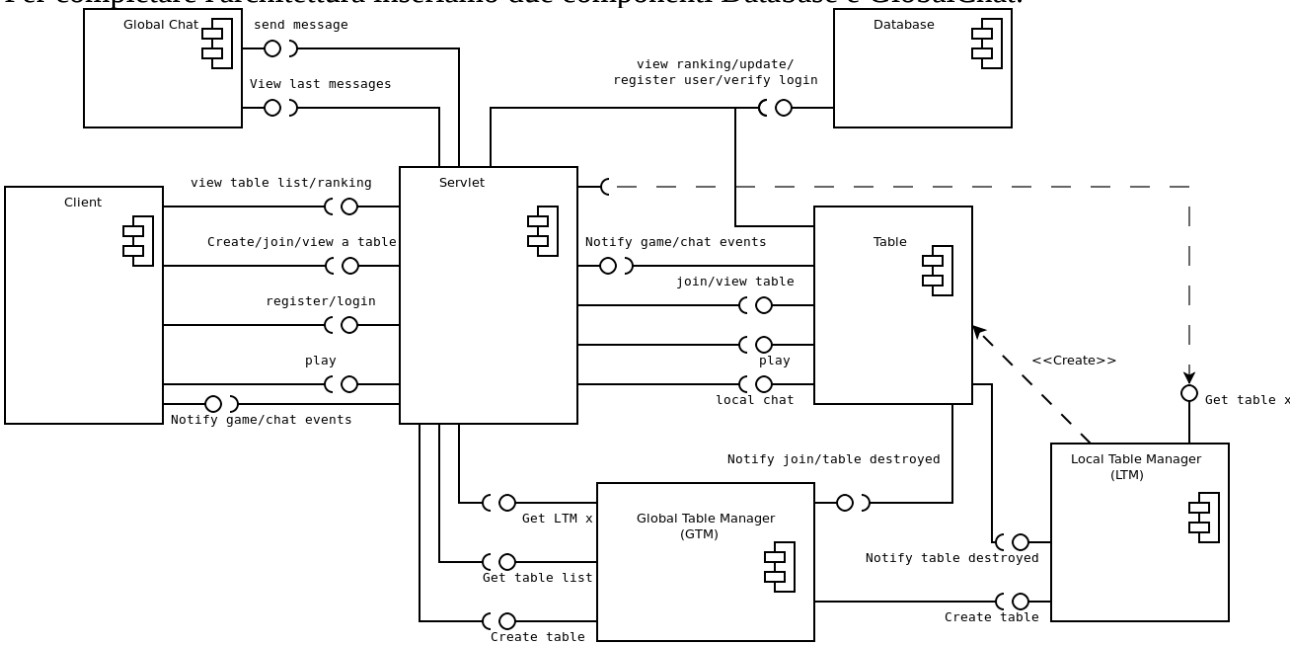


Ogni game server (ora chiamati Local Table Manager o LTM) gestisce un certo numero di componenti Table, ognuno dei quali gestisce una partita.
Global Table Manager ha il compito di suddividere il lavoro tra gli LTM: una servlet che voglia creare un tavolo deve richiedere l'interfaccia remota del Table creato a GTM, che lo creerà in un opportuno LTM secondo politiche interne. GTM deve quindi ricevere notifiche dai LTM sul loro stato, ed in particolare sul loro carico di lavoro.
Sempre per sopperire alle richieste di scalabilità è previsto che anche i Web server siano replicabili. Il carico di lavoro su di essi può essere smistato con un DNS Name Server (non mostrato in figura).



Per richiedere la lista delle partite in corso, Servlet la richiede a GTM, che dovrà quindi mantenere la lista di tutti i Table con lo stato di ogni partita. Per mantenere aggiornato lo stato di GTM, ogni Table notifica un join o la distruzione di un tavolo attraverso l'interfaccia 'Notify join/table destroyed'. Per entrare in una partita (azione di join) o di entrare come spettatore (view) la servlet deve ottenere da GTM una reference al giusto LTM attraverso 'Get LTM x'. Similmente richiede il giusto tavolo al LTM attraverso l'interfaccia 'Get table x'. Questi due identificatori vengono restituiti con la lista delle partite in corso da 'Get Table List' di GTM.

Per completare l'architettura inseriamo due componenti Database e GlobalChat.



Database contiene gli utenti registrati ed il loro punteggio. Database è condiviso tra tutti i Table e tutte le Servlet. Servlet accede al Database per le operazioni di registrazione e login. Table accede al Database per aggiornare i punteggi dei giocatori a fine partita.
Global Chat è un componente condiviso da tutte le servlet, ed è usato per la comunicazione nella chat globale.

In questa fase abbiamo anche deciso tutte le interfacce tra i moduli del sistema. Per vedere il lavoro svolto rimandiamo al file architettura_e_design.pdf ed al codice sorgente, in particolare cupidoCommon/src/unibo/as/cupido/common/interfaces/ e cupidoGWT/src/unibo/as/cupido/client/CupidoInterface.java .

Fase di sviluppo

Nella fase di sviluppo ogni membro del gruppo ha proceduto singolarmente a sviluppare.
Durante questa fase, soprattutto nelle prime iterazioni, è stato necessario modificare più volte le interfacce. Ognuna di queste modifiche è stata apportata dopo averla discussa con tutto il gruppo.

Fase di test dei casi d'uso

Nella fase di Case Testing sono stati creati dei test automatici per testare il backend. I test sono stati così progettati:

1. Gioco con un giocatore umano e tre bot
2. Gioco con due umani e due bot
3. Entrare in modalità spettatore ad un gioco con un umano e tre bot
4. Gioco con un umano, tre bot ed uno spettatore. Dopo un turno il creatore del tavolo lascia la partita.
5. Gioco con due umani e due bot. Dopo un turno il giocatore non creatore lascia la partita.

Questi test sono stati effettuati anche in modo non automatico da utenti umani, per testare il client e la servlet.
In questa fase sono emersi vari errori, alcuni bug di semplice soluzione e problemi di concorrenza tra i bot. Questo ha portato in particolare ad un nuovo design e ad una nuova implementazione dei bot, per risolvere i problemi di concorrenza.

Fase di test dei requisiti

Nella fase di Requirements Testing sono stati verificati tutti requisiti identificati nella fase di analisi e marcati come non opzionali. Solo un numero molto limitato di requisiti non era stato rispettato, e le modifiche richieste al codice sono state esigue.

Riferimenti

[1] http://courses.web.cs.unibo.it/cgi-bin/twiki/bin/viewauth/ArchitettureSoftware/ProgettoDelCorso v0.1