wheelsoft

Specifica Tecnica

1 febbraio 2007

Sommario

Documento contenente la Specifica Tecnica per il capitolato SIAGAS da parte del gruppo di Ingegneria del Software "Wheelsoft". Documento redatto secondo le Norme di Progetto v1.6.

Informazioni documento

Produzione	WheelSoft - wheelsoft@gmail.com
Redazione	Giulio Favotto - gfavotto@studenti.math.unipd.it
	Alberto De Bortoli - adeborto@studenti.math.unipd.it
	Michele Volpato - mvolpato@studenti.math.unipd.it
Approvazione	Stefano Ceschi Berrini - sceschib@studenti.math.unipd.it
	Alessio Rambaldi - arambald@studenti.math.unipd.it
	Silvio Daminato - sdaminat@studenti.math.unipd.it
	Matteo Borgato - mborgato@studenti.math.unipd.it
File	Specifica_Tecnica_v1.6.pdf
Versione	1.6
Stato	Formale
Uso	Esterno
Distribuzione	Wheelsoft
	prof. Alessandro Sperduti
	prof. Renato Conte
	prof. Tullio Vardanega

Diario delle modifiche

1.6	01/02/07 - aggiunta sezione *Diagrammi di sequenza*, corretti schemi del *Logic Layer* e del *Data Layer* e correzioni varie (G. Favotto)
1.5	30/01/07 - aggiunta sezione *Design patterns*, corretta sezione *Requisiti Funzionali* e correzioni varie (G. Favotto, A. Rambaldi, S. Ceschi Berrini)
1.4	27/01/07 - ampliate e corrette descrizioni dei singoli componenti del *Logic Layer* e del *Data Layer* (G. Favotto, A. De Bortoli, M. Volpato)
1.3	26/01/07 - corretto paragrafo *Requisiti Funzionali* e aggiunto paragrafo *Requisiti di Qualità* (G. Favotto, A. Rambaldi, S. Daminato)
1.2	23/01/07 - aggiornati gli schemi e le descrizioni del secondo livello di decomposizione
1.1	21/01/07 - prima stesura del documento

Indice

1 Introduzione

1.1 Scopo del documento

Il documento vuole fornire una descrizione del progetto SIAGAS sulla base del documento "Analisi dei Requisiti v2.0". Verrà illustrata l'architettura del sistema identificandone le componenti e descrivendole in dettaglio.

1.2 Scopo del prodotto

Per lo scopo del prodotto si faccia riferimento al documento "Analisi dei Requisti v2.0".

1.3 Glossario

Per il glossario si faccia riferimento al documento "Glossario v1.7.pdf".

1.4 Riferimenti

1.4.1 Normativi

Per i riferimenti normativi si faccia riferimento al documento "Analisi dei Requisti v2.0".

1.4.2 Informativi

- http://wikipedia.org/
- Guide to the SWEBOK
- CakePHP
- Symfony
- http://www.html.it

2 Definizione del prodotto

2.1 Metodo e formalismo di specifica

Lo strumento utilizzato per la specifica sarà il linguaggio UML attraverso diagrammi delle classi. L'approccio utilizzato nella specifica sarà top-down; ad una descrizione generale dell'architettura del sistema seguiranno delle specifiche dettagliate dei componenti. Il dettaglio della specifica sarà mirato a rendere agevole la fase di codifica del prodotto.

2.2 Primo livello di decomposizione architetturale

2.2.1 Schema

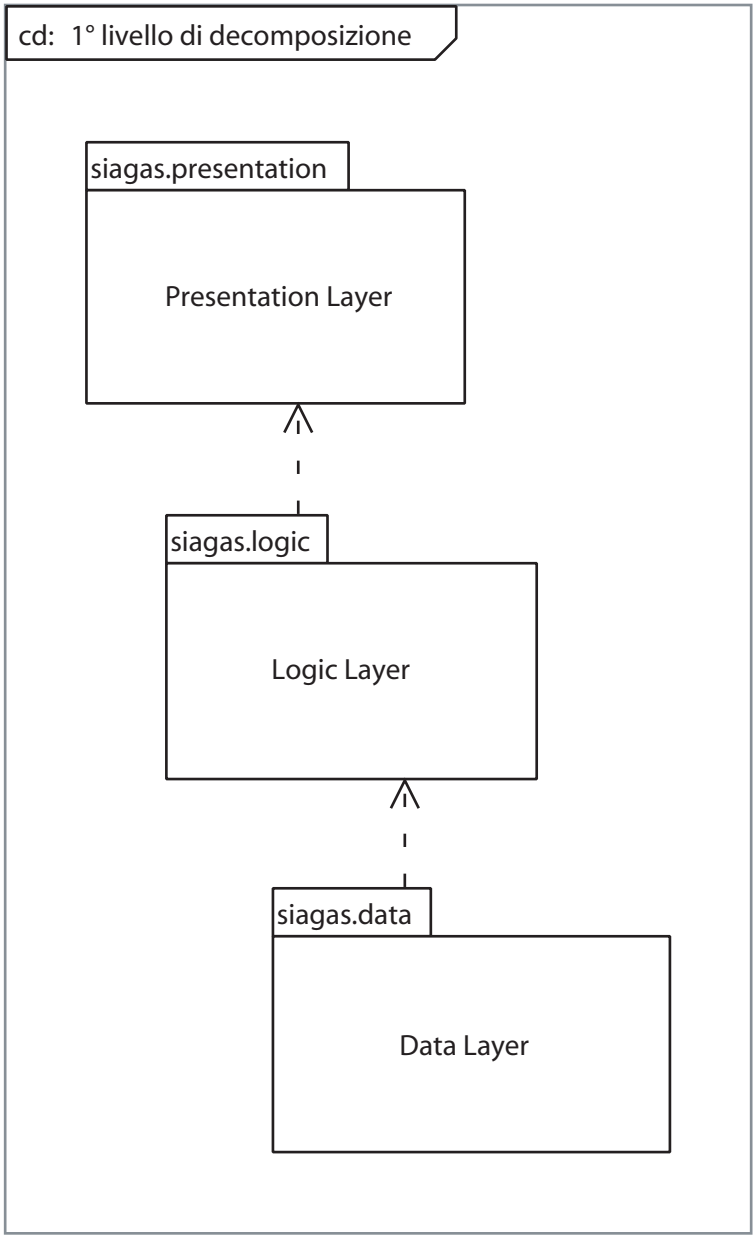

Figura 1: *Primo livello di decomposizione*

2.2.2 Descrizione

Per lo schema generale del prodotto è stato usato il pattern 3-layer che aiuta a strutturare applicazioni che possono essere decomposte in gruppi di unità operative che rappresentantano particolari livelli di astrazione.
Il diagramma seguente rappresenta la schematizzazione dell'architettura. Le sue tre componenti principali sono logicamente raggruppate in package.

1. **Presentation Layer** rappresenta l'interfaccia utente e contiene il codice XHTML per la rappresentazione grafica e l'interazione con l'utente finale. Può contenere degli aspetti logici, ma solo al fine di visualizzare le informazioni in modalità differenti e più adeguate.

2. **Logic Layer** contiene la logica di controllo dell'applicativo gestionale. In particolare ne implementa le funzionalità vere e proprie. Consiste di codice PHP contente controlli di logica, gestisce le richieste del presentation layer e interroga il database al layer sottostante.

3. **Data Layer** Questo strato comprende il database e il software che ne gestisce l'accesso e memorizza i dati nella maniera a più basso livello. Si appoggia sul database SIAGASDB.

2.3 Secondo livello di decomposizione (Presentation)

2.3.1 Descrizione

Lo strato di presentazione verrà trattato approfonditamente nelle prossime fasi, presentando al cliente un prototipo visuale dell'interfaccia web che verrà utilizzata.

Wheelsoft si propone di creare un layout semplice e immediato delle pagine, ma allo stesso accattivante tramite l'uso di *fogli di stile* (CSS). Tutte le pagine saranno scritte in formato XHTML.

Come discusso durante la presentazione della Revisione dei Requisiti, Wheelsoft si propone inoltre di far validare l'intero Presentation Layer secondo standard definiti dal W3C.

2.4 Secondo livello di decomposizione (Logic)

2.4.1 Schema

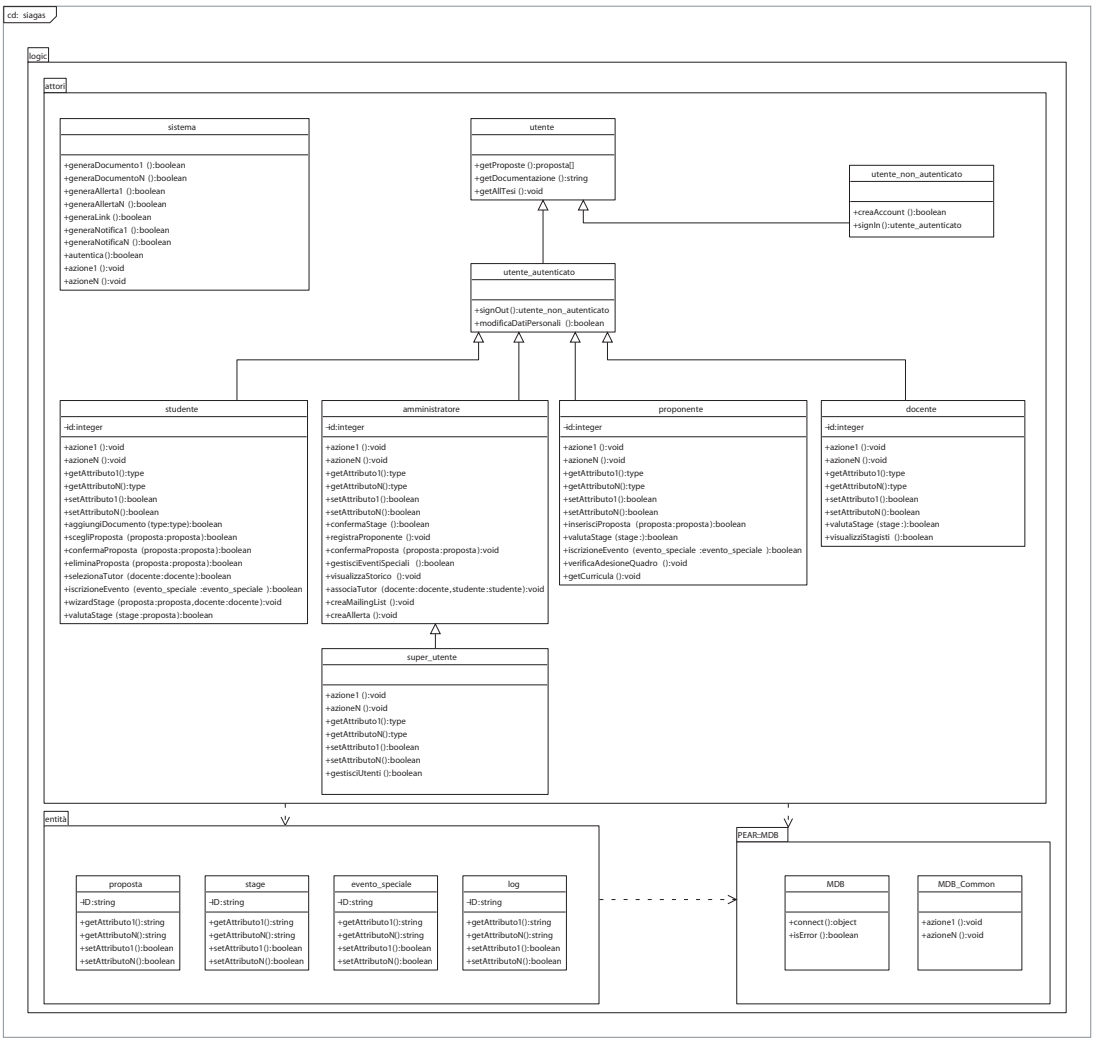

Figura 2: *Secondo livello di decomposizione - Logic Layer*

2.4.2 Descrizione

Il sistema è organizzato secondo una struttura logica a classi che rispecchia l'organizzazione dello strato dati. Essa non è però composta da veri oggetti, contenitori di dati, bensì da contenitori di funzioni riferite ad un'entità o un attore e dotate di campo dati ID univoco. In questo modo si mantengono dunque i benefici di una struttura orientata agli oggetti (incapsulazione, information hiding, polimorfismo...), ma allo stesso tempo non si appesantisce l'esecuzione con dati non necessari. Il recupero dei dati infatti avviene tramite

apposite invocazioni al *SIAGASDB* attraverso l'interfaccia *PEAR::MDB*. Sono state raggruppate nel package *attori* le componenti che ricoprono un ruolo attivo nel sistema, nel package *entità* quelle che possono essere considerate oggetti non attivi usati dagli attori, ed infine nel package *PEAR::MDB* le funzioni che fungono da interfaccia verso il *SIAGASDB*, database vero e proprio. A questo punto della progettazione, i metodi identificati risultano essere tutti con specificatore d'accesso pubblico in quanto interfacce disponibili agli attori e ai componenti del presentation layer; nella progettazione più dettagliata verranno poi inseriti metodi con identificatore d'accesso privato di supporto ai metodi pubblici.

- **package attori**: contiene le funzioni (incapsulate in classi) che serviranno per le principali azioni che l'utente può effettuare. Il suddetto package contiene gli attori che interagiscono con il sistema organizzati secondo una struttura gerarchica. A capo della struttura c'è una classe generale non istanziabile contenente le funzioni lecite a qualsiasi utente del sistema. Da questa classe, derivano da un lato l'utente non autenticato, e dall'altro tutti i tipi di utenti riconosciuti in fase di analisi: *studente*, *proponente*, *amministratore*, *superutente* e *docente*.

- **package entità**: contiene le classi che individuano le entità non attive nel sistema quali: *proposta*, *stage*, *log* e *evento_speciale*. Fondamentalmente esse realizzano oggetti importanti all'interno del sistema, serviranno ad una migliore organizzazione e a daranno più trasparenza al codice.

- **package PEAR::MDB**: contiene le funzioni che forniscono l'interfaccia delle funzioni dei package *attori* e *entità* al *SIAGASDB* in modo da semplificarne lettura e scrittura in maniera sicura e controllata. *PEAR::MDB* è principalmente basato sull'interfaccia MDB_Common che mette a disposizione moltissimi metodi. Il riutilizzo di questo package è un punto chiave del sistema.

2.5 Secondo livello di decomposizione (Data)

2.5.1 Schema

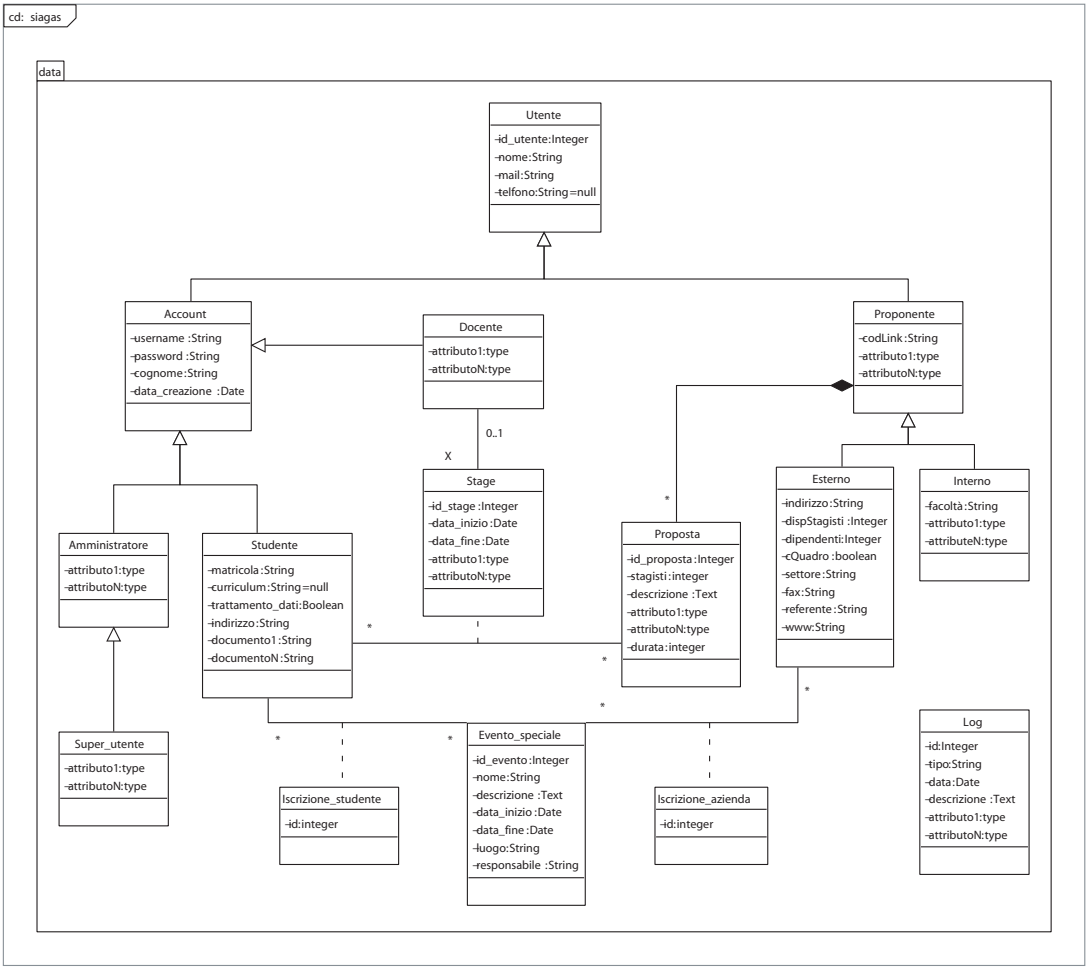

Figura 3: *Secondo livello di decomposizione - Data Layer*

2.5.2 Descrizione

Nella figura 3 è rappresentata la struttura del database secondo i criteri di modellazione UML. Risultano organizzate in maniera gerarchica le tipologie di utenti previste dal sistema, nonché tutti i dati necessari allo loro rappresentazione; a partire da *Utente* vengono derivati da un lato gli utenti dotati di account personale con autenticazione tramite username e password, dall'altro gli utenti denominati *Proponente* che fanno uso di autenticazione tramite link personale. Collegate agli utenti vengono realizzate la classe *Proposta* in relazione di composizione con *Proponente*, la classe *Stage* come associazione di una proposta di stage ad uno studente e le classi *Iscrizione_studente* e *Iscrizione_azienda*

come associazioni di *Studente* e (proponente) *Esterno* ad *Evento_speciale*. Infine la classe *Log* per il log di sistema. Il database *SIAGASDB* è di tipo MySql.

3 Design Patterns

3.1 Factory

Questo pattern ha lo scopo di fornire una classe capace di creare differenti implementazioni di un solo prodotto astratto. Nella pratica è molto utile nelle situazioni in cui un metodo deve restituire istanze di classe diverse in base ad alcuni parametri passati.

Nel nostro progetto lo utilizzeremo per il login degli utenti ed è strutturato da 4 elementi che interoperano tra loro:

- **Product**: è l'interfaccia che viene implementata da tutti gli oggetti che il ConcreteCreator è in grado di generare. Usando un'interfaccia ci assicuriamo congruenza nell'utilizzo delle istanze restituite dal ConcreteCreator;
- **ConcreteProduct**: è l'implementazione dell'interfaccia Product che dona un comportamento specifico ai prototipi dei metodi definiti nell'interfaccia implementata. I ConcreteProduct possono essere più di uno e vengono generati dal ConcreteCreator in base a determinate condizioni (che possono essere lo stato dell'applicazione, i parametri passati al metodo che genera i prodotti, ecc ...);
- **Creator**: è l'interfaccia che viene implementata dai ConcreteCreator e definisce il prototipo di un metodo che si occupa di generare istanze di ConcreteProduct. Spesso l'interfaccia viene definita come una classe astratta che fornisce un'implementazione standard del metodo factory che restituisce il prodotto di default;
- **ConcreteCreator**: è l'implementazione dell'interfaccia Creator che sovrascrive il metodo factory per restituire istanze di ConcreteProduct specifici.

In generale si opera specificando Creator come una classe astratta e fornendo un'implementazione del metodo factory che restituisce l'istanza di uno dei ConcreteProduct definiti (che verrà stabilito come prodotto di default). Poi si definiscono varie implementazioni dell'interfaccia Product ed eventualmente varie implementazioni dell'interfaccia Creator che possono generare set di ConcreteProduct differenti. L'obiettivo è comunque quello di lasciare lasciar decidere quale istanza di ConcreteProduct creare al ConcreteCreator.

Nel nostro caso, la Classe *ConcreteCreator* è *utente_non_autenticato*, il quale si occupa di creare i *ConcreteProduct*. Nel *siagas.logic.attori*, le classi astratte *utente* ed *utente_autenticato* sono i *Product* mentre le classi derivate da *utente_autenticato* sono i *ConcreteProduct*. *Creator* non è stato utilizzato in quanto l'unico *ConcreteCreator* identificato è solo *utente_non_autenticato*.

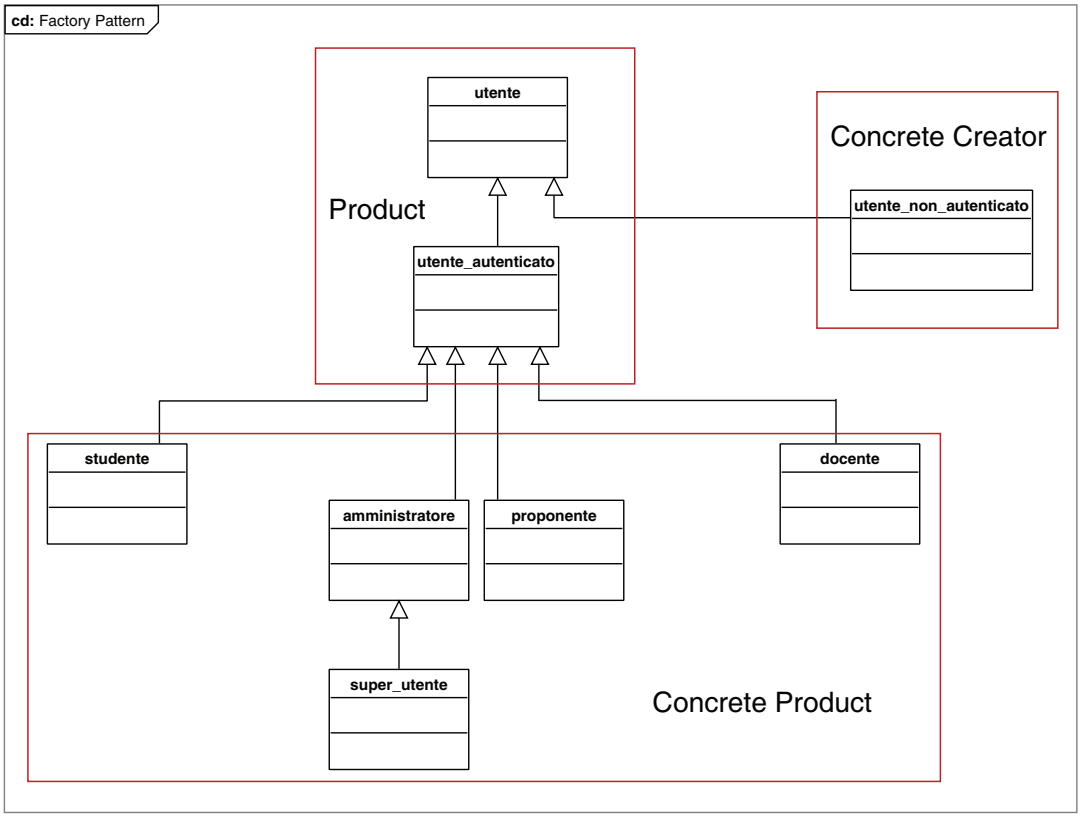

Figura 4: *Rappresentazione UML del pattern Factory*

3.2 Singleton

Si utilizza questo tipo di pattern quando è necessario avere esattamente un'unica istanza di un oggetto per coordinare le azioni all'interno del proprio sistema, oppure quando si hanno degli aumenti di efficienza condividendo una sola istanza. Tramite questa tecnica si elimina l'utilizzo di variabili globali in modo da scrivere codice più ordinato, facilmente manutenibile e meno propenso agli errori.

L'implementazione di questo pattern prevede la creazione di una classe con un metodo statico che crea una nuova istanza di un oggetto, a meno che questa non sia stata già creata in precedenza.

Nel nostro caso si utilizza questo tipo di pattern per creare una sola istanza di ogni classe di utente, quindi il *Concrete Creator* nel *Factory Pattern*, invece di creare l'oggetto vero e proprio utilizza il *Singleton pattern*. Anche la classe *sistema* è implementata rispettando questo pattern.

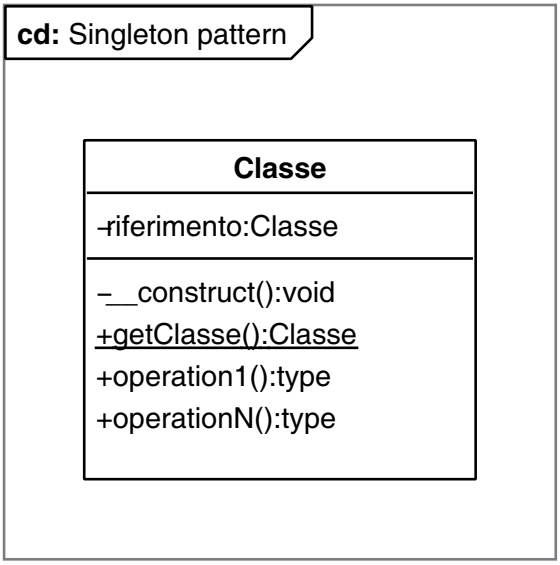

Figura 5: *Rappresentazione UML del pattern Singleton*

3.3 Façade

Questo pattern indica un oggetto che permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro, nonchè a blocchi di codice complessi. Si utilizza questo tipo di pattern in modo da avere una singola e semplice interfaccia al posto di più interfacce complesse, minimizza la comunicazione e la dipendenza tra sottoinsieme e promuove l'accoppiamento lasco fra un sottosistema ed i suoi client.

Abbiamo utilizzato questa tecnica grazie al *Pear::MDB*, che collega le parti *siagas.logic* con *siagas.data*. Nella figura successiva si può notare come sia stato implementato questo tipo di pattern:

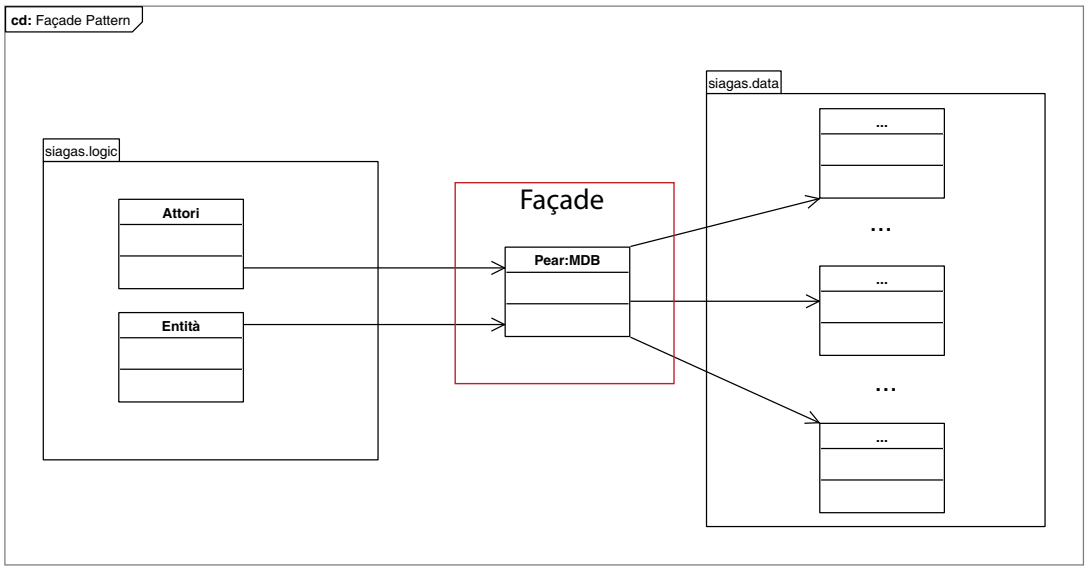

Figura 6: *Rappresentazione UML del pattern Façade*

4 Diagrammi di sequenza

Vengono presentati alcuni diagrammi di sequenza per illustrare il funzionamento di alcune azioni significative del sistema.

4.1 Azione generica

4.1.1 Schema

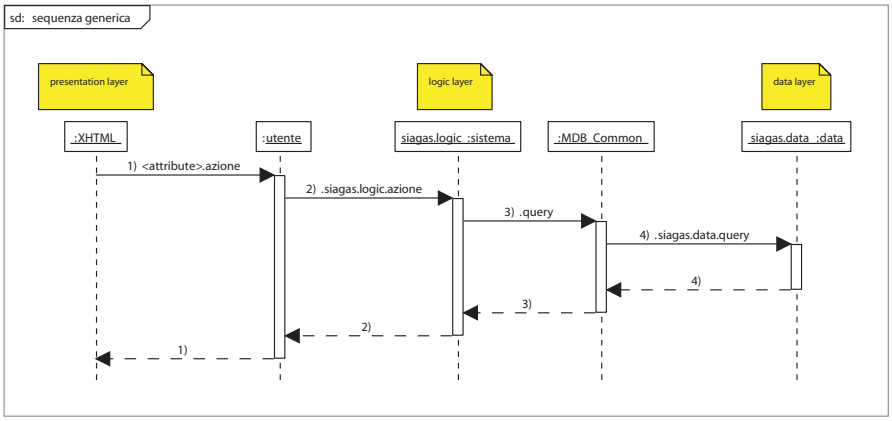

Figura 7: *Diagramma di sequenza per azione generica*

4.1.2 Descrizione

Il diagramma illustra la sequenza delle operazioni e dei messaggi scambiati tra componenti ed attori del sistema per un'azione generica. Il suo scopo è quello di mostrare come un'azione generata nel *presentation layer* si propaghi nel sistema attraverso il *layer logico* ed infine in quello *data*.

4.2 Accesso

4.2.1 Schema

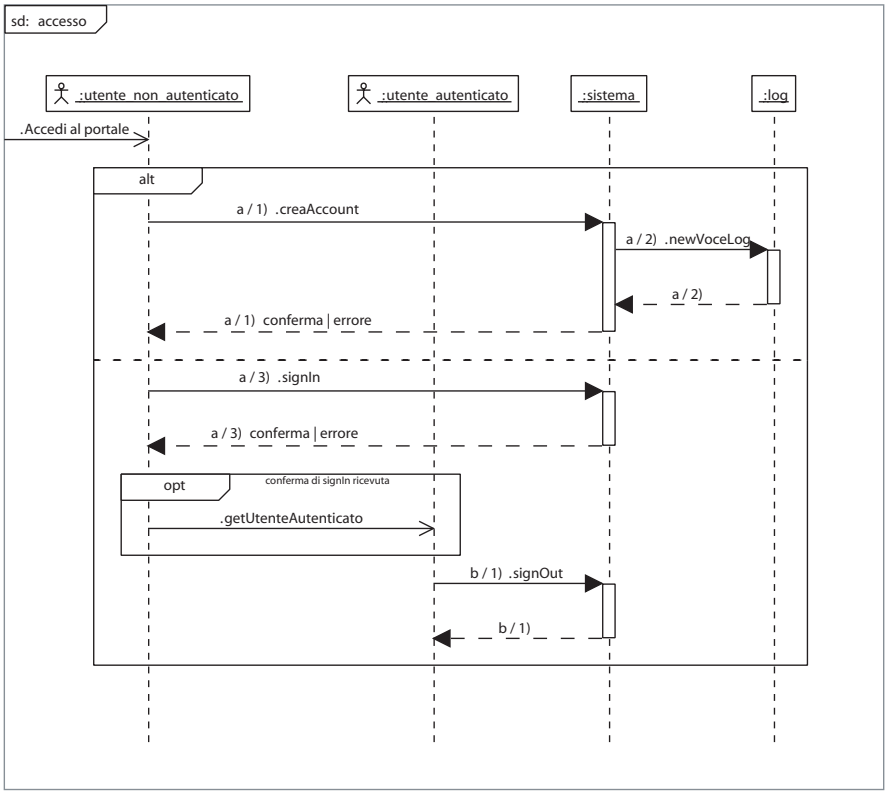

Figura 8: *Diagramma di sequenza per l'accesso al sistema*

4.2.2 Descrizione

Il diagramma illustra la sequenza delle operazioni e dei messaggi scambiati tra componenti ed attori del sistema nella fase di accesso al sistema *SIAGAS*. Da evidenziare come ad un *utente non autenticato* si presentino due alternative iniziali, creare un account nel caso non ne sia già in possesso e disponga dei requisiti necessari per farlo, oppure autenticarsi nel sistema con un account già esistente tramite la funzione *signIn*. Il sistema, ricevute le due richieste, le gestisce generando se necessario una nuova voce di log e fornisce poi risposta. Nel caso la *signIn* vada a buon fine l'*utente non autenticato* viene convertito ad *utente autenticato*. D'altra parte, un utente autenticato può uscire dal sistema tramite la funzione di *signOut*.

4.3 Registrazione proponente

4.3.1 Schema

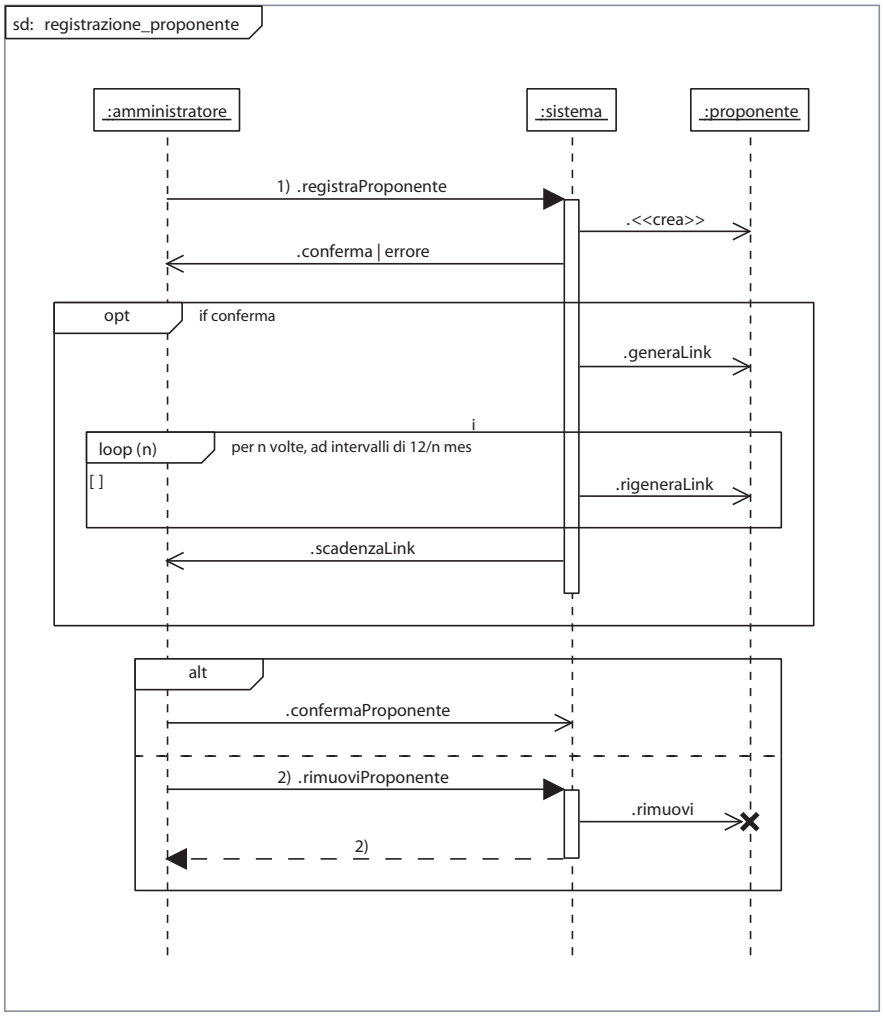

Figura 9: *Diagramma di sequenza per l'inserimento di un proponente*

4.3.2 Descrizione

Il diagramma illustra la sequenza delle operazioni e dei messaggi scambiati tra componenti ed attori del sistema nella fase di inserimento di un nuovo proponente nel sistema da parte dell'amministratore. Si dà come presupposto che l'amministratore abbia ricevuto richiesta da parte del proponente di accesso al sistema. Da evidenziare come, una volta confermato il corretto inserimento del proponente, il sistema generi automaticamente il link di autenticazione e lo invii al proponente. Data inoltre la scadenza di un anno per un account-proponente,

il sistema si occupa di reinviare automaticamente, a scadenze configurabili, un nuovo link. Infine, alla scadenza, l'amministratore viene informato e si cura di confermare o meno il proponente.

5 Descrizione dei singoli componenti

5.1 Descrizione dei componenti di alto livello

La suddivisione in componenti del modello 3-layer individua le seguenti componenti di alto livello:

- **Presentation layer, pagine XHTML**: richiamano metodi delle classi PHP del logic layer e si aspettano array di stringhe (o array di array di stringhe) dai quali cotruiscono la vista richiesta.
- **Logic layer, codice PHP**: i metodi chiamati interrogano il package PEAR e trasformano l'oggetto ritornato in un formato che il livello superiore possa interpretare con facilità; il package *PEAR::MDB* fa da strato intermedio tra il logic layer e il data layer.
- **Data layer, database MySQL**: il *SIAGASDB* vero e proprio, acceduto solo dalla classe *PEAR::MDB*.

Per meglio comprendere le attività svolte si fornisce un esempio completo di gestione di una richiesta dal livello presentazione al livello database: "*richiesta da parte del responsabile di visualizzare tutti gli studenti autenticati nel sistema*".

Il responsabile invierebbe la richiesta tramite un link[1] presente su pagina XHTML. Tale richiesta verrebbe gestita da una funzione php che fa riferimento ad una funzione prescritta che il *PEAR::MDB* mette a disposizione degli sviluppatori. Il SIGASDB verrà opportunamente interrogato con una query e restituirà il risultato (è pensabile che il valore ritornato sia un array contenente i dati ottenuti dalla query) alla funzione che ha chiamato la funzione del *PEAR::MDB*. I dati ottenuti verranno ora trattati e organizzati dalla funzione in questione e da altre di supporto per passarli in maniera adeguata al livello presentazione e presentati sul browser.

5.2 Presentation layer

5.2.1 Pagine XHTML

- **Tipo, obiettivo e funzione del componente**
  Fanno da interfaccia fra l'utente e il sistema SIAGAS, sono rese dinamiche dal codice PHP che contengono; da ogni pagina formato XHTML sarà possibile accedere alle pagine dell'aiuto contestuale, pagine che non interagiscono con i layer sottostanti.
- **Relazioni d'uso di altri componenti**
  Richiamano metodi appartenenti alle classi del package *"siagas.logic.attori"* e si appoggiano ai fogli di stile CSS per la presentazione del browser.
- **Interfacce con e relazioni di uso da altre componenti**
  Le parti dinamiche delle pagine XHTML vengono modificate dalle classi del package *"siagas.logic.attori"*.

[1] In questo caso sarà ragionevolmente creare il link staticamente dato il largo utilizzo di questa interrogazione, mentre altri link che rappresentano interrogazioni differenti potranno essere creati dinamicamente, sulla base dalle scelte del responsabile.

- **Attività svolte e dati trattati**
  Presentano all'utente i dati da lui richiesti, permettono la navigazione del portale e guidano l'utente che necessita d'aiuto tramite aiuto contestuale.

5.2.2 Fogli di stile CSS

- **Tipo, obiettivo e funzione del componente**
  Provvedono a presentare pagine web con un'interfaccia più agevole e intuitiva.
- **Relazioni d'uso di altri componenti**
  Non utilizzano altri componenti.
- **Interfacce con e relazioni di uso da altre componenti**
  Vengono utilizzati da tutte le pagine XHTML.
- **Attività svolte e dati trattati**
  Formattano adeguatamente le pagine XHTML con un determinato layout grafico.

5.3 Logic layer

Si analizzano di seguito le classi relative al packege *siagas.logic.attori*.

5.3.1 attori.utente

- **Tipo, obiettivo e funzione del componente**
  È la classe più generica del sistema e non è istanziabile. Contiene i metodi le cui azioni sono lecite per un qualsiasi tipo di utente.
- **Relazioni d'uso di altri componenti**
  Non utilizza altri componenti.
- **Interfacce con e relazioni di uso da altre componenti**
  Viene specializzata dalle classi *utente_autenticato* e *utente_non_autenticato*.
- **Attività svolte e dati trattati**
  È una classe astratta, può soltanto essere specializzata. Specifica le azioni lecite per un qualsiasi utente del sistema come visualizzare la vetrina delle proposte, consultare la documentazione di stage e il materiale di tesi esistente tramite le apposite funzioni. Non tratta direttamente dati.

5.3.2 attori.utente_non_autenticato

- **Tipo, obiettivo e funzione del componente**
  Contiene i metodi che sono leciti soltanto agli utenti non autenticati.
- **Relazioni d'uso di altri componenti**
  Non utilizza altri componenti.
- **Interfacce con e relazioni di uso da altre componenti**
  Viene utilizzata dai componenti del presentation layer.

- **Attività svolte e dati trattati**
  Rappresenta l'utente che non ha ancora effettuato l'autenticazione nel sistema. Permette azioni come la richiesta di autenticazione e la richiesta di creazione di un account tramite le apposite funzioni. Non tratta direttamente dati.

5.3.3 attori.utente_autenticato

- **Tipo, obiettivo e funzione del componente**
  Contiene i metodi che sono leciti soltanto agli utenti non autenticati. La classe non è istanziabile ma può essere realizzata in *studente*, *amministratore*, *docente* o *proponente*.

- **Relazioni d'uso di altri componenti**
  Non utilizza altri componenti.

- **Interfacce con e relazioni di uso da altre componenti**
  Viene specializzata dalle classi *studente*, *amministratore*, *proponente* e *docente*.

- **Attività svolte e dati trattati**
  Rappresenta l'utente che ha effettuato l'autenticazione nel sistema. Permette azioni come la richiesta di chiusure della sessione e la modifica dei dati personali tramite le apposite funzioni. Non tratta direttamente dati.

5.3.4 attori.studente

- **Tipo, obiettivo e funzione del componente**
  Realizza da *utente_autenticato*. Contiene i metodi le cui azioni sono lecite per un utente di tipo studente autenticato nel sistema.

- **Relazioni d'uso di altri componenti**
  Utilizza la classe astratta *utente_autenticato* e i package *entità* e *PEAR::MDB*.

- **Interfacce con e relazioni di uso da altre componenti**
  Viene utilizzata dai componenti del presentation layer.

- **Attività svolte e dati trattati**
  Permette le azioni lecite per lo studente tramite le funzioni apposite come l'avvio della procedura guidata di creazione stage e l'aggiunta di un documento. La classe ha un campo dati ID che identifica univocamente lo studente. Possiede inoltre metodi tramite i quali è possibile ottenere tutte le informazioni su tale classe dal *SIAGASDB* (getAttributo() e setAttributo()).

5.3.5 attori.amministratore

- **Tipo, obiettivo e funzione del componente**
  Realizza da *utente_autenticato*. Contiene i metodi le cui azioni sono lecite per un utente di tipo amministratore autenticato nel sistema.

- **Relazioni d'uso di altri componenti**
  Utilizza la classe astratta *utente_autenticato* e i package *entità* e *PEAR::MDB*.

- **Interfacce con e relazioni di uso da altre componenti**
  Viene utilizzata dai componenti del presentation layer.

- **Attività svolte e dati trattati**
  Permette le azioni lecite per l'amministratore tramite le funzioni apposite come la gestione degli eventi speciali e la creazione di account per i proponenti. La classe ha un campo dati ID che identifica univocamente l'amministratore. Possiede inoltre metodi tramite i quali è possibile ottenere tutte le informazioni su tale classe dal *SIAGASDB* (getAttributo() e setAttributo()).

5.3.6 attori.super_utente

- **Tipo, obiettivo e funzione del componente**
  Realizza da *utente_autenticato*. Contiene i metodi le cui azioni sono lecite per un utente di tipo super-utente autenticato nel sistema.

- **Relazioni d'uso di altri componenti**
  Utilizza la classe astratta *amministratore* e i package *entità* e *PEAR::MDB*.

- **Interfacce con e relazioni di uso da altre componenti**
  Viene utilizzata dai componenti del presentation layer.

- **Attività svolte e dati trattati**
  Permette le azioni lecite per il super-utente tramite le funzioni apposite come il backup del sistema. Possiede inoltre metodi tramite i quali è possibile ottenere tutte le informazioni su tale classe dal *SIAGASDB* (*getAttributo()* e *setAttributo()*).

5.3.7 attori.proponente

- **Tipo, obiettivo e funzione del componente**
  Realizza da *utente_autenticato*. Contiene i metodi le cui azioni sono lecite per un utente di tipo amministratore autenticato nel sistema.

- **Relazioni d'uso di altri componenti**
  Utilizza la classe astratta *utente_autenticato* e i package *entità* e *PEAR::MDB*.

- **Interfacce con e relazioni di uso da altre componenti**
  Viene utilizzata dai componenti del presentation layer.

- **Attività svolte e dati trattati**
  Permette le azioni lecite per il proponente tramite le funzioni apposite. La classe ha un campo dati ID che identifica univocamente il proponente. Possiede inoltre metodi tramite i quali è possibile ottenere tutte le informazioni su tale classe dal *SIAGASDB*.

5.3.8 attori.docente

- **Tipo, obiettivo e funzione del componente**
  Realizza da *utente_autenticato*. Contiene i metodi le cui azioni sono lecite per un utente di tipo docente autenticato nel sistema.

- **Relazioni d'uso di altri componenti**
  Utilizza la classe astratta *utente_autenticato* e i package *entità* e *PEAR::MDB*.
- **Interfacce con e relazioni di uso da altre componenti**
  Viene utilizzata dai componenti del presentation layer.
- **Attività svolte e dati trattati**
  Permette le azioni lecite per il docente tramite le funzioni apposite. La classe ha un campo dati ID che identifica univocamente il docente. Possiede inoltre metodi tramite i quali è possibile ottenere tutte le informazioni su tale classe dal *SIAGASDB*.

5.3.9 attori.sistema

- **Tipo, obiettivo e funzione del componente**
  Classe di supporto con metodi utili per la gestione dell'intero sistema tramite.
- **Relazioni d'uso di altri componenti**
  Utilizza il package *PEAR:MDB*.
- **Interfacce con e relazioni di uso da altre componenti**
  Viene utilizzata dai componenti del package attori.
- **Attività svolte e dati trattati**
  Permettere attività di supporto del sistema tramite funzioni apposite, come la generazione di documenti, la gestione delle allerte e la generazione di link destinati alle aziende per il responsabile.

Si analizzano di seguito le classi relative al packege *siagas.logic.entità*.

5.3.10 entità.proposta

- **Tipo, obiettivo e funzione del componente**
  Rappresentante una proposta di stage. Possiede un campo dati ID che la identifica univocamente. Attraverso i suoi metodi è possibile ottenere tutte le informazioni su tale classe dal *SIAGASDB*.
- **Relazioni d'uso di altri componenti**
  Utilizza il package *PEAR:MDB*.
- **Interfacce con e relazioni di uso da altre componenti**
  Viene utilizzata dai componenti del package attori.
- **Attività svolte e dati trattati**
  Tramite le funzioni apposite è possibile interagire con il *SIAGASDB* per ottenre e memorizzare le informazioni dell'oggetto.

5.3.11 entità.stage

- **Tipo, obiettivo e funzione del componente**
  Rappresentante uno stage. Possiede un campo dati ID che lo identifica univocamente. Attraverso i suoi metodi è possibile ottenere tutte le informazioni su tale classe dal *SIAGASDB*.

- **Relazioni d'uso di altri componenti**
  Utilizza il package *PEAR:MDB*.

- **Interfacce con e relazioni di uso da altre componenti**
  Viene utilizzata dai componenti del package attori.

- **Attività svolte e dati trattati**
  Tramite le funzioni apposite è possibile interagire con il *SIAGASDB* per ottenere e memorizzare le informazioni dell'oggetto.

5.3.12 entità.evento_speciale

- **Tipo, obiettivo e funzione del componente**
  Rappresentante un evento_speciale. Possiede un campo dati ID che lo identifica univocamente. Attraverso i suoi metodi è possibile ottenere tutte le informazioni su tale classe dal *SIAGASDB*.

- **Relazioni d'uso di altri componenti**
  Utilizza il package *PEAR:MDB*.

- **Interfacce con e relazioni di uso da altre componenti**
  Viene utilizzata dai componenti del package attori.

- **Attività svolte e dati trattati**
  Tramite le funzioni apposite è possibile interagire con il *SIAGASDB* per otterene e memorizzare le informazioni dell'oggetto.

5.3.13 entità.log

- **Tipo, obiettivo e funzione del componente**
  Rappresentante il log del sistema. Possiede un campo dati ID che lo identifica univocamente. Attraverso i suoi metodi è possibile ottenere tutte le informazioni su tale classe dal *SIAGASDB*.

- **Relazioni d'uso di altri componenti**
  Utilizza il package *PEAR:MDB*.

- **Interfacce con e relazioni di uso da altre componenti**
  Viene utilizzata dal componente *attori.sistema*.

- **Attività svolte e dati trattati**
  Tramite le funzioni apposite è possibile interagire con il *SIAGASDB* per otterene e memorizzare le informazioni dell'oggetto.

Si analizzano di seguito le classi relative al packege *siagas.logic.PEAR::MDB*.

5.3.14 PEAR::MDB.MDB

- **Tipo, obiettivo e funzione del componente**
  Gestisce in maniera sicura ed efficiente l'accesso al *SIAGASDB*.

- **Relazioni d'uso di altri componenti**
  Utilizza la classe *MDB_Common* per soddisfare le richieste del livello logico.

- **Interfacce con e relazioni di uso da altre componenti**
  Viene utilizzata dal logic layer per effettuare la connessione al *SIAGASDB*, data layer.

- **Attività svolte e dati trattati**
  Permette la connessione al *SIAGASDB* e la rilevazione di eventuali errori tramite le due apposite funzioni *Connect()* e *isError()*.

5.3.15 PEAR::MDB.MDB_Common

- **Tipo, obiettivo e funzione del componente**
  Gestisce in maniera sicura ed efficiente le interrogazioni verso il *SIAGASDB*.

- **Relazioni d'uso di altri componenti**
  Utilizza il data layer per ottenere le informazioni richieste.

- **Interfacce con e relazioni di uso da altre componenti**
  Viene utilizzata dal logic layer per effettuare le richieste al *SIAGASDB*, data layer.

- **Attività svolte e dati trattati**
  Permette l'interrogazione del *SIAGASDB* tramite apposite funzioni.

5.4 Data layer

Si analizzano di seguito le classi relative al packege *siagas.data*.

5.4.1 Utente

- **Tipo, obiettivo e funzione del componente**
  Utente è la classe generale che rappresenta gli utenti che utilizzano il sistema SIAGAS, contiene i campi dati in comune tra le classi degli utilizzatori.

- **Relazioni d'uso di altri componenti**
  La classe non ha relazioni d'uso verso altri componenti.

- **Interfacce con e relazioni di uso da altre componenti**
  La classe viene specializzata dalle classi *Account* e *Proponente*. Viene usata dal package *PEAR::MDB*.

- **Attività svolte e dati trattati**
  Contiene i campi dati *id_utente*, *nome*, *mail*, *telefono* (opzionale) che la classi che la specializzano hanno in comune.

5.4.2 Account

- **Tipo, obiettivo e funzione del componente**
  È la classe che rappresenta gli utenti che accedono al sistema tramite autenticazione esplicita.

- **Relazioni d'uso di altri componenti**
  La classe specializza *Utente*.

- **Interfacce con e relazioni di uso da altre componenti**
  La classe viene specializzata dalle classi *Amministratore*, *Docente* e *Studente*. Viene usata dal package *PEAR::MDB*.

- **Attività svolte e dati trattati**
  Contiene i campi dati che la classi che la specializzano hanno in comune, tra i quali *username*, *password*, *cognome* e *data_creazione*.

5.4.3 Proponente

- **Tipo, obiettivo e funzione del componente**
  È la classe che rappresenta gli utenti che accedono al sistema senza un'autenticazione esplicita.

- **Relazioni d'uso di altri componenti**
  La classe specializza *Utente*. È in relazione di composizione con *Proposta*, questo perchè se un oggetto della classe *Proponente* dovesse smettere di esistere, non avrebbe senso mantenere le proposte di stage che il proponente che l'oggetto rappresenta ha inviato. Precisamente un oggetto della classe *Proponente* viene associato a più oggetti della classe *Proposta* se è stato l'utente associato all'oggetto di *Proponente* ad inserire le proposte di stage rappresentate dagli oggetti di *Proposta*.

- **Interfacce con e relazioni di uso da altre componenti**
  La classe viene specializzata dalle classi *Esterno* e *Interno*. Interagisce con la classe *Proposta*. Viene usata dal package *PEAR::MDB*.

- **Attività svolte e dati trattati**
  Contiene i campi dati in comune con le classi che la specializzano, in particolare ha rilevante importanza il link univoco associato al proponente tramite il quale può inserire proposte di stage.

5.4.4 Amministratore

- **Tipo, obiettivo e funzione del componente**
  È la classe che rappresenta gli utenti autenticati con diritti di amministratore, ovvero il responsabile (o delegati del responsabile).

- **Relazioni d'uso di altri componenti**
  La classe specializza *Account*.

- **Interfacce con e relazioni di uso da altre componenti**
  La classe viene specializzata dalla classe *Super_utente*. Viene usata dal package *PEAR::MDB*.

- **Attività svolte e dati trattati**
  Gli oggetti di questa classe rappresentano gli amministratori di sistema. Contiene i campi dati relativi all'amministratore.

5.4.5 Studente

- **Tipo, obiettivo e funzione del componente**
  È la classe che rappresenta gli studenti autenticati dal sistema.
- **Relazioni d'uso di altri componenti**
  La classe specializza *Account*. La classe è associata a *Proposta* tramite *Stage* ed a *Evento_speciale* tramite *Iscrizione_studente*. Precisamente un oggetto della classe *Studente* viene associato a uno o più oggetti della classe *Proposta* tramite l'associazione *Stage*. In questo modo, in Stage saranno presenti sia gli stage effettivi che le scelte scartate dall'utente. Gli oggetti vengono associati anche ad oggetti della classe *Evento_speciale* tramite l'associazione *Iscrizione_studente* nel caso lo studente si iscriva ad uno o più eventi speciali.
- **Interfacce con e relazioni di uso da altre componenti**
  È associata alla classe *Proposta* tramite *Stage* ed a *Evento_speciale* tramite *Iscrizione_studente*. Viene usata dal package *PEAR::MDB*.
- **Attività svolte e dati trattati**
  Lo studente viene identificato da una matricola e dall'indirizzo personale. Sono previsti gli attributi curriculum opzionale, un booleano indicante il consenso al trattamento dei dati personali e i possibili link di documenti da lui inseriti.

5.4.6 Super_utente

- **Tipo, obiettivo e funzione del componente**
  È la classe che rappresenta gli utenti autenticati con diritti di amministratore e super utente.
- **Relazioni d'uso di altri componenti**
  La classe specializza *Amministratore*.
- **Interfacce con e relazioni di uso da altre componenti**
  La classe viene usata dal package *PEAR::MDB*.
- **Attività svolte e dati trattati**
  Gli oggetti di questa classe rappresentano i super utenti di sistema.

5.4.7 Esterno

- **Tipo, obiettivo e funzione del componente**
  È la classe che rappresenta gli utenti che accedono al sistema senza un'autenticazione esplicita e che rappresentano aziende esterne all'università.
- **Relazioni d'uso di altri componenti**
  La classe specializza *Proponente*. Gli oggetti vengono associati anche ad oggetti della classe *Evento_speciale* tramite l'associazione *Iscrizione_azienda* nel caso l'azienda si iscriva ad uno o più eventi speciali.
- **Interfacce con e relazioni di uso da altre componenti**
  È associata alla classe *Evento_speciale* tramite *Iscrizione_studente*. Viene usata dal package *PEAR::MDB*.

- **Attività svolte e dati trattati**
  Gli oggetti di questa classe vengono associati agli oggetti della classe *Evento_speciale*. Precisamente un oggetto della classe *Esterno* viene associato a più oggetti della classe *Evento_speciale* se è l'utente associato all'oggetto di *Esterno* vuole iscriversi all'evento speciale (oggetto della classe *Evento_speciale*).

5.4.8 Interno

- **Tipo, obiettivo e funzione del componente**
  È la classe che rappresenta gli utenti che accedono al sistema senza un'autenticazione esplicita e che rappresentano docenti interni all'università.
- **Relazioni d'uso di altri componenti**
  La classe specializza *Proponente*.
- **Interfacce con e relazioni di uso da altre componenti**
  Viene usata dal package *PEAR::MDB*.
- **Attività svolte e dati trattati**
  Gli oggetti di questa classe rappresentano i docenti che propongono stage secondo facoltà.

5.4.9 Proposta

- **Tipo, obiettivo e funzione del componente**
  È la classe che rappresenta le proposte inviate dai proponenti.
- **Relazioni d'uso di altri componenti**
  È in relazione di composizione con *Proponente*, questo perchè se un oggetto della classe *Proponente* dovesse smettere di esistere, non avrebbe senso mantenere le proposte di stage che il proponente che l'oggetto rappresenta ha inviato. È associata alla classe *Studente* tramite l'associazione *Stage*.
- **Interfacce con e relazioni di uso da altre componenti**
  Viene usata dal package *PEAR::MDB*.
- **Attività svolte e dati trattati**
  Viene identificata da un ID e da una descrizione della proposta di stage e dal numero di stagisti.

5.4.10 Docente

- **Tipo, obiettivo e funzione del componente**
  Gli oggetti di questa classe rappresentano i docenti del corso di laurea che sono disponibili come tutor interni per gli stage.
- **Relazioni d'uso di altri componenti**
  La classe specializza *Account* ed è associata a *Stage*.
- **Interfacce con e relazioni di uso da altre componenti**
  Viene usata dal package *PEAR::MDB*.

- **Attività svolte e dati trattati**
  Si noti il valore di associazione impostato alla variabile X in modo che il numero di stage di cui un docente può essere tutor sia limitabile dall'amministratore.

5.4.11 Stage

- **Tipo, obiettivo e funzione del componente**
  È la classe che rappresenta le scelte di stage da parte degli studenti. Uno studente può scegliere più proposte di stage "preferite", ma dovrà poi decidere quale, tra quelle scelte diventerà il suo stage.
- **Relazioni d'uso di altri componenti**
  La classe è il risultato di un'associazione tra *Studente*, *Docente* e *Proposta*.
- **Interfacce con e relazioni di uso da altre componenti**
  Viene usata dal package *PEAR::MDB*.
- **Attività svolte e dati trattati**
  La classe *Stage* rappresenta l'entità principale del sistema legando le proposte formulate da un proponente, lo studente interessato alla proposta ed il docente interno predisposto a supportarla. Contiene pertanto un ID dello stage, data di inizio e data di fine effettive. Altri attributi verranno specificati in seguito.

5.4.12 Evento_speciale

- **Tipo, obiettivo e funzione del componente**
  È la classe che rappresenta gli eventi speciali atti a facilitare l'incontro tra aziende e studenti.
- **Relazioni d'uso di altri componenti**
  La classe è associata a *Esterno* tramite l'associazione *Iscrizione_azienda* ed a *Studente* tramite l'associazione *Iscrizione_studente*.
- **Interfacce con e relazioni di uso da altre componenti**
  Viene usata dal package *PEAR::MDB*.
- **Attività svolte e dati trattati**
  Contiene i campi dati significativi per rappresentare un evento speciale (tra i quali nome, descrizione, data inizio, data fine).

5.4.13 Iscrizione_studente

- **Tipo, obiettivo e funzione del componente**
  È la classe che rappresenta l'iscrizione da parte degli studenti a determinati eventi speciali.
- **Relazioni d'uso di altri componenti**
  È la classe creata dall'associazione "molti a molti" tra *Evento_speciale* e *Studente*.
- **Interfacce con e relazioni di uso da altre componenti**
  Viene usata dal package *PEAR::MDB*.

- **Attività svolte e dati trattati**
  Essendo la classe creata dall'associazione "molti a molti" tra *Evento_speciale* e *Studente* contiene i campi dati di identificazione delle due classi (le chiavi primarie) e campi dati significativi della relazione come la data di registrazione.

5.4.14 Iscrizione_azienda

- **Tipo, obiettivo e funzione del componente**
  È la classe che rappresenta l'iscrizione da parte delle aziende a determinati eventi speciali.

- **Relazioni d'uso di altri componenti**
  È la classe creata dall'associazione "molti a molti" tra *Evento_speciale* e *Esterno*.

- **Interfacce con e relazioni di uso da altre componenti**
  Viene usata dal package *PEAR::MDB*.

- **Attività svolte e dati trattati**
  Essendo la classe creata dall'associazione "molti a molti" tra *Evento_speciale* e *Esterno* contiene i campi dati di identificazione delle due classi (le chiavi primarie) e campi dati significativi della relazione come la data di registrazione.

5.4.15 Log

- **Tipo, obiettivo e funzione del componente**
  È la classe che contiene le registrazioni delle azioni principali eseguite dagli utenti del sistema SIAGAS, nonché i possibili errori verificatisi.

- **Relazioni d'uso di altri componenti**
  La classe non si relaziona con altri componenti.

- **Interfacce con e relazioni di uso da altre componenti**
  Viene usata dal package *PEAR::MDB*.

- **Attività svolte e dati trattati**
  Contiene campi dati che identificano l'errore o l'azione, il tipo d'errore o azione e una descrizione utile al responsabile.

6 Stime di fattibilità e di bisogno di risorse

Il sistema sarà sviluppato utilizzando tecnologie che sono racchiuse nel pacchetto denominato LAMP (Linux [Windows in alternativa], Apache, MySql, PHP). Ovviamente altri linguaggi come XHTML e CSS sono indispensabili. Il pacchetto LAMP è totalmente gratuito e questa scelta risulta tra l'altro forzata rispetto, ad esempio, all'utilizzo di un database Oracle non gratuito.

Le risorse software *gratuite* necessarie sono dunque le seguenti:

- **server Apache**
- **editor testuali**
- **editor WYSIWYG** (opzionali e di suppporto)
- **linguaggio PHP**
- **database MySQL**
- **browser differenti**
- **validatori W3C**
- **spazio web** per test

Non serve specificare hardware dedicato per l'esecuzione del prodotto. Le strutture informatiche già presenti all'interno dell'Università di Padova (spazio web e server) consentono l'integrazione del sistema SIAGAS. La fase di realizzazione, verifica e validazione, verrà effettuata sul dominio **wheelsoft.org** dotato delle risorse necessarie in maniera privata.

7 Tracciamento della relazione componenti-requisiti

La seguente tabella illustra i componenti che soddisfano i requisiti identificati dal codice univoco definito nell'analisi dei requisiti.

7.1 Requisiti funzionali

Requisito	Soddisfacimento	Componente
RFOb01	funzione *getProposta()*	logic.attori.utente
RFOb02	funzione *aggiungiDocumento()*	logic.attori.studente
RFOb03	funzione *iscrizioneEvento()*	logic.attori.studente
	funzione *iscrizioneEvento()*	logic.attori.proponente
RFOb04	funzione *aggiungiDocumento()*	logic.attori.studente
RFOb05	pagine XHTML con guida contestuale	presentation
	funzione *wizardStage()*	logic.attori.studente
RFOb06	funzione *inserisciProposta()*	logic.attori.proponente
RFOb07	funzione *gestisciUtenti()*	logic.attori.super_utente
	funzione *confermaProposta()*	logic.attori.amministratore
RFOb08	funzione *gestisciEventiSpeciali()*	logic.attori.amministratore
RFOb09	funzioni *generaDocumento()*	logic.attori.sistema
RFOb10	classe *esterno*	data.esterno
RFOb11	funzione *generaLink()*	logic.attori.sistema
	funzione *registraProponente()*	logic.attori.amministratore
RFOb12	funzione *selezioneTutor()*	logic.attori.studenti
	funzione *associaTutor()*	logic.attori.amministratore
RFOb13	classe *super_utente*	logic.attori.super_utente
RFOb14	funzione *verificaAdesioneQuadro()*	logic.attori.proponente
RFDe01	funzione *getAllTesi()*	logic.attori.studente
RFDe02	funzione *getCurricula()*	logic.attori.proponente
RFDe03	funzioni *generaNotifica()*	logic.attori.sistema
RFDe04	funzioni *creaMailingList()*	logic.attori.amministratore
RFDe05	funzione *creaAllerta()*	logic.attori.amministratore
RFOp01	funzione *valutaStage()*	logic.attori.studente
	funzione *valutaStage()*	logic.attori.docente
	funzione *valutaStage()*	logic.attori.proponente
RFOp02	funzione *visualizzaStorico()*	logic.attori.amministratore
RFOp03	funzioni *selezionaTutor()*	logic.attori.studente
	funzione *associaTutor()*	logic.attori.amministratore
RFOp04	funzioni *visualizzaStagisti()*	logic.attori.docente
RFOp05	funzione *generaLink()*	logic.attori.sistema

7.2 Requisiti di qualità

Requisito	Soddisfacimento	Componente
RQOb01	utilizzo WAI	presentation
RQOb02	utilizzo *PEAR::MDB* e struttura OO	logic e data
RQOb03	pagine XHTML con guida contestuale	presentation
RQOb04	utilizzo di XHTML, CSS	presentation
	utilizzo di PHP	logic
	utilizzo di MySql	data