

KALEIDOSCODE
SWEDesigner
SOFTWARE PER DIAGRAMMI UML
DEFINIZIONE DI PRODOTTO V1.0.0



Informazioni sul documento

Versione	1.0.0
Data Redazione	15/06/2017
Redazione	Bonolo Marco Pace Giulio Pezzuto Francesco Sovilla Matteo
Verifica	Sanna Giovanni
Approvazione	Bonato Enrico
Uso	Esterno
Distribuzione	<i>Prof. Vardanega Tullio</i> <i>Prof. Cardin Riccardo</i> <i>Zucchetti s.p.a.</i>

Diario delle Modifiche

Versione	Data	Autore	Descrizione
0.0.1	01/05/2017	Pace Giulio	Creazione scheletro del documento e stesura della sezione Introduzione

Contents

1	Introduzione	1
1.1	Scopo del documento	1
1.2	Scopo del prodotto	1
1.3	Glossario	1
1.4	Riferimenti utili	1
1.4.1	Riferimenti normativi	1
1.4.2	Riferimenti informativi	1
2	Standard di progetto	3
2.1	Progettazione architetturale	3
2.2	Documentazione del codice	3
2.3	Programmazione	3
2.4	Strumenti e procedure	3
2.5	Denominazione relazioni ed entità	3
3	Architettura dell'applicazione	4
3.1	Architettura client	4
3.1.1	Diagrammi editabili	5
3.2	Architettura server	5
3.2.1	Comunicazioni server-client	6
4	Specifica delle componenti	7
4.1	SWEDesigner	7
4.2	SWEDesigner::Client	7
4.3	SWEDesigner::Client::Model	8
4.3.1	SWEDesigner::Client::Model::DataManager	8
4.3.2	SWEDesigner::Client::Model::ProjectModel	9
4.3.3	SWEDesigner::Client::Model::ToolbarModel	11
4.3.4	SWEDesigner::Client::Model::Project	12
4.3.5	SWEDesigner::Client::Model::RequestHandler	13
4.4	SWEDesigner::Client::View	14
4.4.1	SWEDesigner::Client::View::ProjectView	15
4.4.2	SWEDesigner::Client::View::TitlebarView	17
4.4.3	SWEDesigner::Client::View::ToolbarView	18
4.4.4	SWEDesigner::Client::View::PathView	19
4.4.5	SWEDesigner::Client::View::EditPanelView	20
4.5	SWEDesigner::Server	22
4.6	SWEDesigner::Server::CodeGenerator	22
4.6.1	SWEDesigner::Server::CodeGenerator::CodeGenerator	23
4.7	SWEDesigner::Server::CodeGenerator::Builder	23
4.7.1	SWEDesigner::Server::CodeGenerator::Builder::Builder	23
4.8	SWEDesigner::Server::CodeGenerator::Coder	24
4.8.1	SWEDesigner::Server::CodeGenerator::Coder::Coder	24
4.8.2	SWEDesigner::Server::CodeGenerator::Coder::JavaCoder	25
4.8.3	SWEDesigner::Server::CodeGenerator::Coder::JavascriptCoder	25

4.8.4	SWEDesigner::Server::CodeGenerator::Coder::CoderClass	25
4.8.5	SWEDesigner::Server::CodeGenerator::Coder::CoderOperation	25
4.8.6	SWEDesigner::Server::CodeGenerator::Coder::CoderParameter	26
4.8.7	SWEDesigner::Server::CodeGenerator::Coder::CoderAttribute	26
4.8.8	SWEDesigner::Server::CodeGenerator::Coder::CoderActivity	26
4.8.9	SWEDesigner::Server::CodeGenerator::Coder::CodedProg	26
4.8.10	SWEDesigner::Server::CodeGenerator::Coder::CoderElement	27
4.9	SWEDesigner::Server::CodeGenerator::Parser	27
4.9.1	SWEDesigner::Server::CodeGenerator::Parser::Parser	27
4.10	SWEDesigner::Server::CodeGenerator::Zipper	28
4.10.1	SWEDesigner::Server::CodeGenerator::Zipper::Zipper	28
4.10.2	SWEDesigner::Server::DAO	28
4.11	SWEDesigner::Server::RequestHandler	28
4.11.1	SWEDesigner::Server::RequestHandler::Sender	28
4.11.2	SWEDesigner::Server::RequestHandler::Receiver	29
5	Diagrammi di sequenza	30

List of Tables

List of Figures

1	Architettura del client	4
2	Architettura del server	6
3	Esempi delle possibili comunicazioni client-server	6
4	Architettura del client	7
5	Architettura di Model	9
6	Architettura di View	14
7	Architettura del server	22
8	Architettura di Coder	24

1 Introduzione

1.1 Scopo del documento

Con il presente documento si intende definire la progettazione in dettaglio della struttura e del funzionamento delle componenti del progetto *SWEDesigner*.

Verrà presentato innanzi tutto l'architettura secondo la quale verranno organizzate le componenti software. Successivamente verranno specificate le componenti nel dettaglio per la parte client e server. Infine verranno mostrati i diagrammi di sequenza delle principali azioni lato back-end al fine di facilitare la comprensione del funzionamento del programma. Il documento ha la funzione di servire da guida ai *Programimatori* del gruppo.

1.2 Scopo del prodotto

Lo scopo del progetto è la realizzazione di un software di costruzione di diagrammi UMLG con la relativa generazione di codice JavaG e JavascriptG utilizzando tecnologie web. Il prodotto deve essere conforme ai vincoli qualitativi richiesti dal committente.

1.3 Glossario

Al fine di evitare ogni ambiguità di linguaggio e massimizzare la comprensione dei documenti i termini tecnici, di dominio, gli acronimi e le parole che necessitano di essere chiarite sono riportate nel documento *Glossario v3.0.0*.

La prima occorrenza di ciascuno di questi vocaboli è marcata da una "G" maiuscola in pedice.

1.4 Riferimenti utili

1.4.1 Riferimenti normativi

- **CapitolatoG d'appalto:**
<http://www.math.unipd.it/~tullio/IS-1/2016/Progetto/C6.pdf> (09/03/2017);
- **Norme di progetto:** *Norme di progetto v3.0.0*;
- **Analisi dei requisiti:** *Analisi dei requisiti v3.0.0*;
- **Specifica tecnica:** *Specifica tecnica v2.0.0*;
- **Verbali esterni:**
 - Verbale incontro con *Zucchetti s.p.a.* in data 05/05/2017.

1.4.2 Riferimenti informativi

- **Slide dell'insegnamento di Ingegneria del Software 1o semestre:**
 - Design pattern strutturali:
<http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E04.pdf> (02/05/2017);

- Design pattern creazionali:
<http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E05.pdf> (02/05/2017);
- Design pattern comportamentali:
<http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E06.pdf> (02/05/2017);
- Design pattern architetturali:
<http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E07.pdf> (02/05/2017),
<http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E08.pdf> (02/05/2017);
- Stili architetturali:
<http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E09.pdf> (02/05/2017);
- **Design Patterns: Elements of reusable object-oriented software**
E. Gamma, R. Helm, R. Johnson, J. Vlissides - 1st Edition (2002)
 - Capitolo 3: Creational patterns;
 - Capitolo 4: Structural patterns;
 - Capitolo 5: Behavioral patterns.
- **Jointjs**: <https://www.jointjs.com/opensource> - 02/05/2017
- **jQuery**: <https://jquery.com/> - 02/05/2017
- **Lodash**: <https://lodash.com/> - 02/05/2017
- **Backbone.js**: <http://backbonejs.org/> - 02/05/2017
- **Node.js**: <https://nodejs.org/it/> - 02/05/2017
- **RequireJS**: <http://requirejs.org/> - 02/05/2017
- **MySQL**: <https://www.mysql.com/> - 02/05/2017

2 Standard di progetto

- 2.1 Progettazione architettuale
- 2.2 Documentazione del codice
- 2.3 Programmazione
- 2.4 Strumenti e procedure
- 2.5 Denominazione relazioni ed entità

3 Architettura dell'applicazione

SWEDesigner è realizzato utilizzando un'architettura client-server, in particolare:

- Il **client** corrisponde alla parte dell'applicativo che funzionerà nel browser dell'utente;
- Il **server** avrà il compito di fornire la pagina dell'applicativo al client e ne gestirà le richieste ricevute riguardanti la generazione del codice sorgente o le attività "bubble" da inserire nell'editor.

3.1 Architettura client

Il client (parte front-end) è una Single Page Application (SPAG) scritta con i linguaggi HTML5, CSS3 e Javascript.

La sua architettura è costruita utilizzando il framework Backbone.js che offre un'architettura di tipo Model-View ed è quindi principalmente suddivisa nei seguenti moduli:

- **Model:** organizza la logica alla base dei diagrammi dell'editor.
- **View:** gestisce l'interfaccia grafica dell'editor e, seguendo la struttura definita da Backbone.js, "contiene" la componente controller per la gestione degli eventi;

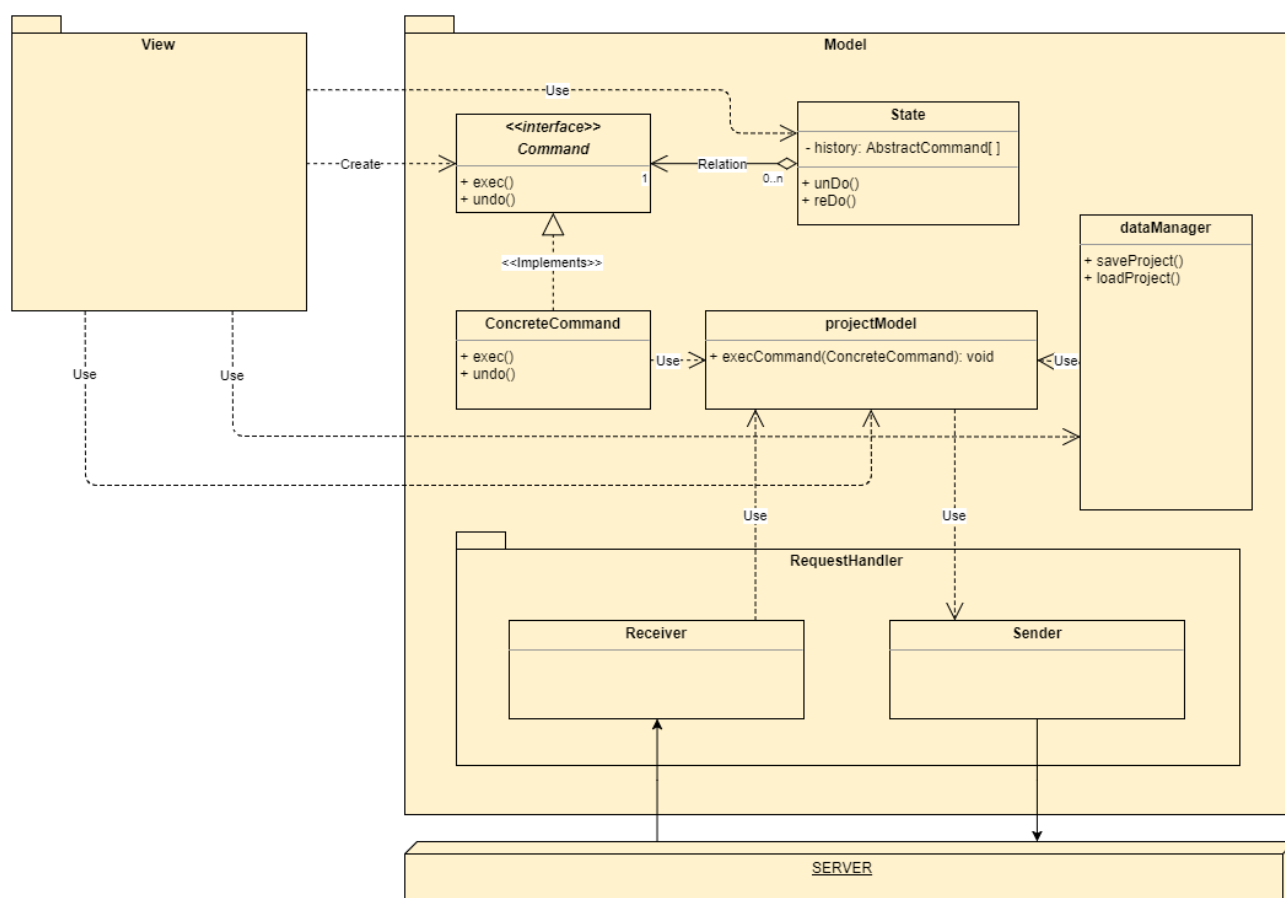


Figure 1: Architettura del client

3.1.1 Diagrammi editabili

In ogni diagramma creabile all'interno dell'applicazione è offerto solamente un sottoinsieme del totale dei formalismi definiti dal linguaggio UML standard. Si possono individuare tre tipi di diagrammi:

- Diagramma dei packageG;
- Diagramma delle classi;
- Diagramma delle bubble.

Il diagramma dei package è logicamente correlato con il diagramma delle classi. Per ogni elemento (package o classe) all'interno di questi diagrammi è possibile assegnare un livello di importanza attraverso il quale si può "filtrare" gli oggetti a schermo visualizzabili nell'editor.

Per la corretta generazione del codice, nei diagrammi delle attività è previsto che l'utente approfondisca il loro livello di astrazione fino ad arrivare ad un diagramma costituito solamente da bubble (diagramma delle bubble) che verranno fornite nell'editor come se fossero delle attività specifiche.

3.2 Architettura server

Il server (parte back-endG) è sviluppato in Node.js ed offre i seguenti servizi:

- Fornire la Single Page Application ai client che la richiedono;
- Fornire la lista di bubble utilizzabili nell'editor;
- Generare il codice sorgente, nel formato desiderato dal client, del progetto inviatogli.

In particolare, la componente che genera il codice sorgente è stata realizzata utilizzando un'architettura di tipo Pipe And Filter, in modo tale da assegnare un compito ben preciso ad ogni modulo per attuare una procedura sequenziale a "catena di montaggio". L'ultimo modulo ha il compito di creare un file compresso .zip del codice generato che sarà poi inviato al client.

Le bubble saranno salvate in una base dati per poter garantire una futura estendibilità del numero di queste ultime, eventualmente anche in altri domini da quello considerato al momento (i giochi da tavolo).

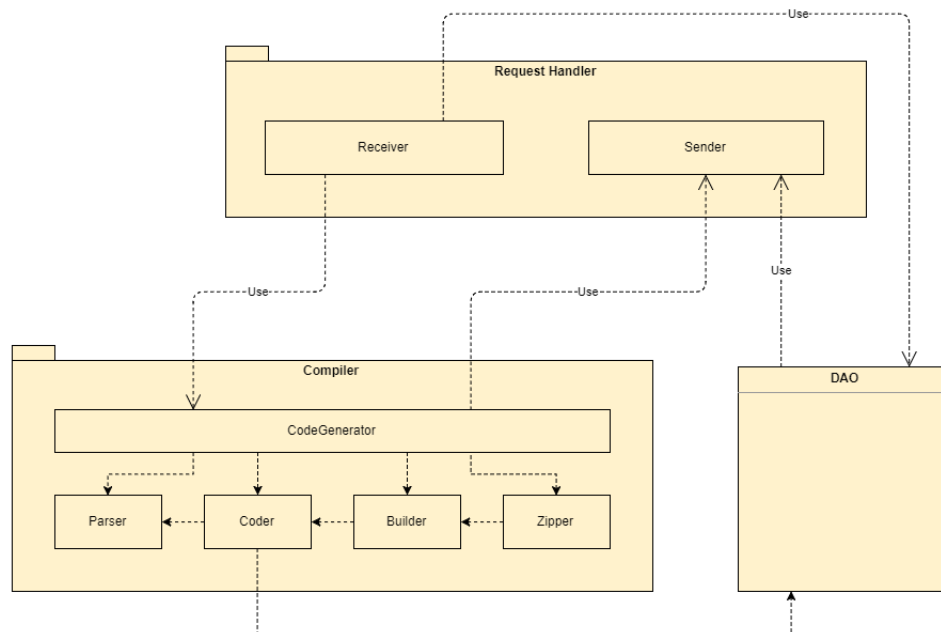


Figure 2: Architettura del server

3.2.1 Comunicazioni server-client

La Single Page Application viene fornita al client semplicemente attraverso una pagina HTML.

Per la richiesta e fornitura delle bubble, client e server utilizzano AJAX per lo scambio di dati in formato JSON in modo tale da alleggerire il traffico.

Per la richiesta della generazione del codice, il client invia i dati del progetto in formato JSON utilizzando AJAX ed il server una volta elaborata la richiesta procede con l'invviare il file .zip precedentemente descritto.

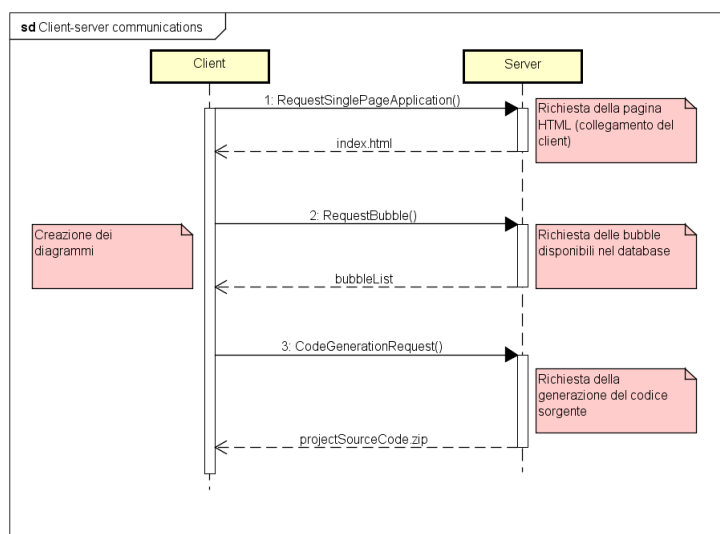


Figure 3: Esempi delle possibili comunicazioni client-server

4.3 SWEDesigner::Client::Model

I package contenuti al suo interno sono:

- SWEDesigner::Client::Model::Items.

Le classi contenute al suo interno verranno elencate qui di seguito.

4.3.1 SWEDesigner::Client::Model::DataManager

- **Tipo:** *Classe statica*;
- **Descrizione:** Si occupa della persistenza dei dati, in particolare del salvataggio su file system locale del progetto già esistente.
;
- **Metodi:**
 - *newProject(): void*
Dopo aver chiesto conferma all'utente, crea un nuovo progetto sovrascrivendo quello correntemente aperto;
 - *openProject(): void*
Legge un file JSON e ne salva il contenuto in project e nel projectModel come progetto attualmente aperto;
 - *save(fileName: String): void*
Salva i dati del progetto, li converte in formato JSON e avvia la procedura di download in locale del browser;
Parametri:
 - * *fileName: String* Nome del file generato da scaricare;
 - *saveAs(): void*
Estrae la stringa inserita dall'utente nella schermata per il salvataggio con nome e invoca la il metodo per il salvataggio del progetto corrente in un file con il nome desiderato;
- **Relazioni con le altre classi:**
 - OUT *SWEDesigner::Client::Model::ProjectModel*: si occupa di gestire la parte logica dell'editor;
 - OUT *SWEDesigner::Client::Model::Project*: si occupa di gestire gli elementi contenuti nel diagramma.

4.3.2 SWEDesigner::Client::Model::ProjectModel

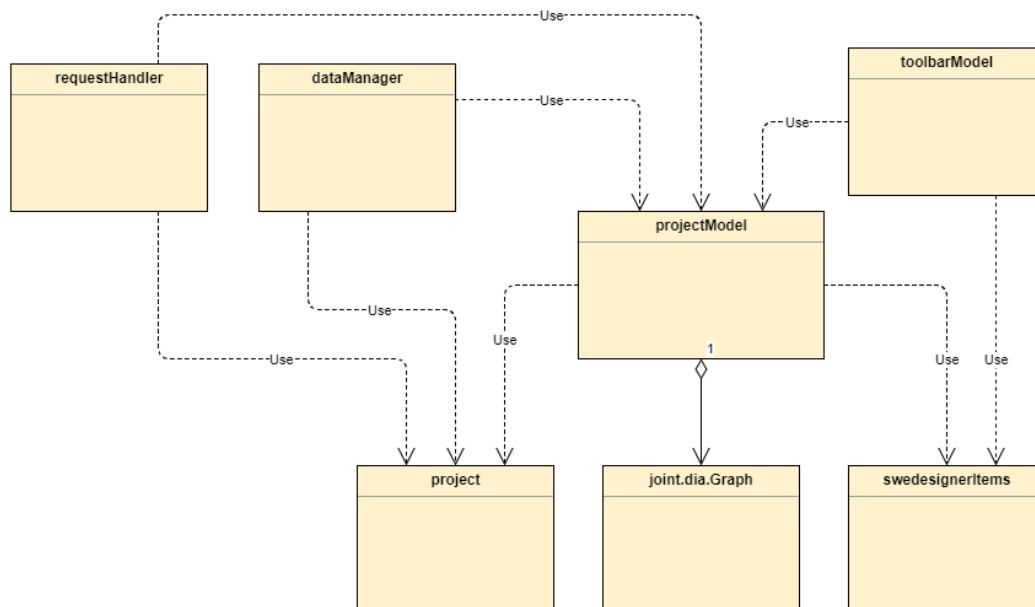


Figure 5: Architettura di Model

- **Tipo:** *Classe*;
- **Descrizione:** Model del progetto corrente. Si occupa di gestire il graph (*joint.dia.Graph*) e tutti gli eventi ad esso associati;
- **Padre:** *Backbone.model*;
- **Attributi:**
 - *currentDiagram: String*
L'id del diagramma correntemente caricato nel graph (null se è il diagramma dei package);
 - *currentDiagramType: String*
Il tipo del diagramma correntemente caricato nel graph ("packageDiagram", "classDiagram" o "bubbleDiagram");
 - *graph: joint.dia.Graph*
Il model dell'area di disegno associata al paper della *SWEDesigner::Client::View::ProjectView*;
 - *itemToBeAdded: String*
Salvataggio temporaneo dell'elemento da aggiungere al graph corrente;
- **Metodi:**

- *addItem(item: Object): void*

Salva in itemToBeAdded l'elemento passato in input che è un oggetto di Swedesigner::Client::M

Parametri:

- * *item: Object* Elemento del diagramma;

- *addItemToGraph(): void*

Aggiunge un elemento al grafo del diagramma corrente;

- *changedPosition(graph: joint.dia.Graph, cell: joint.dia.Cell, newPosition: Object, opt: Object): void*

Gestisce la traslazione di un elemento selezionato nel grafo;

Parametri:

- * *graph: joint.dia.Graph* Grafo del diagramma corrente;
- * *cell: joint.dia.Cell* Elemento correntemente selezionato;
- * *newPosition: Object* Posizione attuale dell'oggetto nel grafo;
- * *opt: Object* Traslazione dell'oggetto dalla posizione iniziale alla posizione "newPosition";

- *deleteCell(): void*

Rimuove un elemento dal grafo eliminando anche gli eventuali diagrammi derivati (classi o bubble);

- *deleteOperation(): void*

Rimuove un'operazione ed eventualmente anche il diagramma delle bubble associato;

- *getCellFromId(cellId: String): void*

Ritorna l'elemento del graph avente l'id passato come parametro in input;

Parametri:

- * *cellId: String* Identificativo dell'elemento nel graph;

- *graphSwitched(): void*

Genera l'evento "switchgraph";

- *initialize(): void*

Inizializzazione del ProjectModel: inizializzazione del graph, del currentDiagramType, degli eventi verificabili;

- *resizeParent(parent: Object): void*

Esegue il resize di un elemento del diagramma ingrandendolo;

Parametri:

- * *parent*: *Object* Elemento del diagramma;
- *saveCurrentDiagram()*: *void*
Salva il diagramma correntemente aperto all'interno della struttura definita nella classe *Project*;
- *switchInGraph()*: *void*
Esegue lo switch in profondità al diagramma selezionato svuotando il graph dagli elementi correntemente presenti e caricando gli eventuali nuovi elementi;
- *switchOutGraph()*: *void*
Esegue lo switch all'antistante tipo di diagramma selezionato svuotando il graph dagli elementi correntemente presenti e caricando gli eventuali nuovi elementi;

- **Relazioni con le altre classi:**

- IN *SWEDesigner::Client::Model::DataManager*: si occupa della persistenza dei dati, in particolare del salvataggio su file system locale del progetto e del caricamento di un progetto già esistente; *SWEDesigner::Client::Model::ToolbarModel*: È il componente del programma che si occupa di gestire la parte logica della toolbar;
- IN *SWEDesigner::Client::Model::RequestHandler*: si occupa di gestire i dati ricevuti dal server;
- OUT *SWEDesigner::Client::Model::Project*: si occupa di gestire gli elementi contenuti nel diagramma;
- OUT *SWEDesigner::Client::Model::Items::Swedesigner*: è il contenitore degli elementi che si possono inserire in un diagramma.

4.3.3 SWEDesigner::Client::Model::ToolbarModel

- **Tipo**: *Classe*;
- **Descrizione**: È il componente del programma che si occupa di gestire la parte logica della toolbar;
- **Padre**: *Backbone.model*;
- **Attributi**:
 - *items*: *Object*
Contiene tutti gli elementi definibili nel diagramma corrente;
- **Metodi**:

- *addElement(id: String): void*

Salva lo strumento selezionato interagendo con il *SWEDesigner::Client::Model::ProjectModel*;

Parametri:

- * *id: String* Identificativo del tipo di strumento/elemento da inserire;
- *createItems(): void*
Assegna al campo dati "items" il set di strumenti utilizzabili nel diagramma corrente;
- *currentDiagram(): String*
Ritorna il tipo del diagramma corrente;
- *initialize(): void*
Inizializzazione del ToolbarModel: chiama il metodo createItems;

● **Relazioni con le altre classi:**

- OUT *SWEDesigner::Client::Model::ProjectModel*: si occupa di gestire la parte logica dell'editor;
- OUT *SWEDesigner::Client::Model::Items::Swedesigner*: è il contenitore degli elementi che si possono inserire in un diagramma.

4.3.4 SWEDesigner::Client::Model::Project

- **Tipo:** *Classe*;
- **Descrizione:** Contenitore di tutti gli elementi del progetto correntemente aperto nella Single Page Application;
- **Padre:** *Backbone.model*;
- **Attributi:**
 - *classes: Object*
Contiene: *classesArray* (array contenente diagrammi delle classi; in ogni indice è presente un oggetto id: *idPackagePadre*, items: [arrayClassiDelDiagramma]) e *dependenciesArray* (array contenente i link del corrispondente diagramma delle classi; in ogni indice è presente un oggetto id: *idPackagePadre*, items: [arrayLinkDelDiagramma]);
 - *operations: Array<Object>*
Contiene un array di oggetti; in ogni indice è presente un oggetto id: id dell'operazione, items: [arrayBubbleDelDiagramma]);
 - *packages: Object*
Contiene: *packagesArray* (array contenente i package item del diagramma dei

package) e dependenciesArray (array contenente i link del diagramma dei package);

- **Metodi:**

- *deleteClassesDiagramOfPkg(id: String): void*

Elimina il diagramma delle classi associato al package e tutti i diagrammi delle bubble associati alle operazioni delle relative classi;

Parametri:

- * *id: String* Identificativo del package;

- *deleteOperationDiagram(id: String): void*

Elimina il diagramma delle bubble associato all'operazione;

Parametri:

- * *id: String* Identificativo dell'operazione;

- *getClassIndex(id: String): Number*

Cerca ed eventualmente ritorna l'indice dell'array classesArray del diagramma delle classi associato al package;

Parametri:

- * *id: String* Identificativo del package;

- *getOperationIndex(id: String): Number*

Cerca ed eventualmente ritorna l'indice dell'array operations del diagramma delle bubble associato all'operazione;

Parametri:

- * *id: String* Identificativo dell'operazione;

- **Relazioni con le altre classi:**

- IN *SWEDesigner::Client::Model::ProjectModel*: si occupa di gestire la parte logica dell'editor;
 - IN *SWEDesigner::Client::Model::DataManager*: si occupa della persistenza dei dati, in particolare del salvataggio su file system locale del progetto e del caricamento di un progetto già esistente;
 - IN *SWEDesigner::Client::Model::RequestHandler*: si occupa di gestire i dati ricevuti dal server;

4.3.5 SWEDesigner::Client::Model::RequestHandler

- **Tipo:** *Classe*;

- **Descrizione:** Si occupa della gestione delle comunicazioni tra client e server (lato client);

- **Padre:** *Backbone.model*;
- **Metodi:**
 - *caricaJa(): void*
Carica il file json nel server e ne genera il codice Java restituendo il nome della cartella compressa;
 - *caricaJs(): void*
Carica il file json nel server e ne genera il codice Javascript restituendo il nome della cartella compressa;
- **Relazioni con le altre classi:**
 - OUT *SWEDesigner::Client::Model::ProjectModel*: si occupa di gestire la parte logica dell'editor;
 - OUT *SWEDesigner::Client::Model::Project*: si occupa di gestire gli elementi contenuti nel diagramma.

4.4 SWEDesigner::Client::View

È il componente del programma che si occupa di gestire l'interfaccia grafica. Nella particolare declinazione MVC adottata da Backbone.js, si occupa anche di gestire gli input dell'utente.

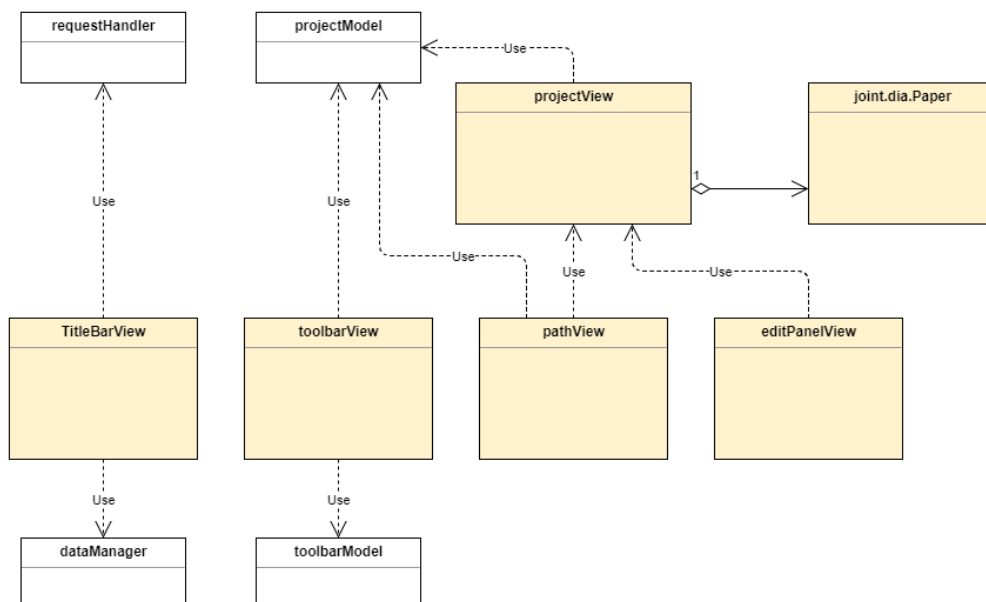


Figure 6: Architettura di View

Questo package non contiene dei sottopackage. Le classi contenute al suo interno verranno elencate qui di seguito.

4.4.1 SWEDesigner::Client::View::ProjectView

- **Tipo:** *Class*;
- **Descrizione:** Questa classe gestisce il diagramma disegnato e le interazioni dell'utente con esso;
- **Padre:** *Backbone.View*;
- **Attributi:**
 - *model*
Istanza di *ProjectModel* del programma;
 - *paper*
Oggetto *joint.dia.Paper* della libreria esterna *JointJS*;
- **Metodi:**
 - *resetSelectedCell(): void*
Pone *this.paper.selectedCell* a null e genera l'evento "changed-selected-cell";
 - *mouseMoveFunction(event: JavaScriptEvent): void*
Provoca la traslazione del paper nella direzione del trascinamento del mouse;
Parametri:
 - * *event: JavaScriptEvent*: Evento;
 - *blankPointerDown(elem: CellView, event: JavaScriptEvent, x: Double, y: Double): void*
Salva le correnti coordinate al click del mouse nello spazio vuoto del paper;
Parametri:
 - * *elem: CellView*: Elemento *cellView*;
 - * *event: JavaScriptEvent*: Evento;
 - * *x: Double*: Coordinata dell'asse delle ascisse;
 - * *y: Double*: Coordinata dell'asse delle ordinate;
 - *blankPointerUp(elem: CellView, event: JavaScriptEvent, x: Double, y: Double): void*
Elimina le coordinate iniziali al click del mouse nello spazio vuoto del paper;
Parametri:
 - * *elem: CellView*: Elemento *cellView*;
 - * *event: JavaScriptEvent*: Evento;
 - * *x: Double*: Coordinata dell'asse delle ascisse;
 - * *y: Double*: Coordinata dell'asse delle ordinate;
 - *onMouseWheel(elem: CellView, event: JavaScriptEvent): void*
Trasla verticalmente il paper effettuando uno zoom in avanti o indietro a seconda della rotazione della ruota del mouse;
Parametri:
 - * *elem: CellView*: Elemento *cellView*;

- * *event: JavaScriptEvent*: Evento;
- *render(): void*
Provoca il render della *projectView*;
- *addCell(elem: CellView, event: JavaScriptEvent, x: Double, y: Double): void*
Aggiunge un nuovo elemento al graph chiamando il relativo metodo di *ProjectModel*;
Parametri:
 - * *elem: cellView*: Elemento *CellView*;
 - * *event: JavaScriptEvent*: Evento;
 - * *x: Double*: Coordinata dell'asse delle ascisse;
 - * *y: Double*: Coordinata dell'asse delle ordinate;
- *deleteCell(event: JavaScriptEvent): void*
Elimina un elemento dal graph chiamando il relativo metodo di *ProjectModel*;
Parametri:
 - * *event: JavaScriptEvent*: Evento;
- *unembedCell(event: JavaScriptEvent): void*
Rimuove l'innestamento della cella selezionata;
Parametri:
 - * *event: JavaScriptEvent*: Evento;
- *pointerDownFunction(prView: projectView, elem: cellView, event: JavaScriptEvent, x: double, y: double): void*
Gestisce l'evento generato dal click (non rilasciato) del mouse nel paper. Se viene cliccato un elemento, genera a sua volta l'evento "changed-selected-cell" gestito da *EditPanelView*;
Parametri:
 - * *prView: ProjectView*: Istanza di *ProjectView*;
 - * *elem: cellView*: Elemento *CellView*;
 - * *event: JavaScriptEvent*: Evento;
 - * *x: Double*: Coordinata dell'asse delle ascisse;
 - * *y: Double*: Coordinata dell'asse delle ordinate;
- *pointerUpFunction(prView: ProjectView, elem: CellView, event: JavaScriptEvent, x: Double, y: Double): void*
Gestisce l'evento generato dal click (al rilascio) del mouse nel paper (rimozione di un elemento, nesting di un elemento in un'altro, collegamento di una relazione tra elementi);
Parametri:
 - * *prView: projectView*: Istanza di *projectView*;
 - * *elem: CellView*: Elemento *cellView*;
 - * *event: JavaScriptEvent*: Evento;
 - * *x: Double*: Coordinata dell'asse delle ascisse;
 - * *y: Double*: Coordinata dell'asse delle ordinate;

- *switchIn(id: String): void*
Gestisce lo switch in profondità (dall'elemento selezionato il cui id è parametro in input) invocando il relativo metodo di *ProjectModel*;
Parametri:
 - * *id: String*: Identificativo dell'elemento;
- *switchOut(diagramType: String): void*
Gestisce lo switch verso un diagramma (il cui tipo è parametro in input) antistante da quello corrente invocando il relativo metodo di *ProjectModel*.
Parametri:
 - * *diagramType: String*: Tipo di diagramma di destinazione;
- *deleteOperationAt(ind: Int): void*
Gestisce l'eliminazione di un diagramma delle bubble invocando il relativo metodo di *projectModel*.
Parametri:
 - * *ind: Int*: Indice dell'array di operazioni del diagramma delle bubble da eliminare;

FAN-IN:

- *PathView*: gestisce l'interfaccia grafica della barra di indirizzo;
- *EditPanelView*: gestisce l'interfaccia grafica del pannello di editing.

FAN-OUT:

- *ProjectModel*: si occupa di gestire la parte logica dell'editor;
- *joint.dia.Paper*: gestisce l'interfaccia grafica dell'area dei diagrammi.

4.4.2 SWEDesigner::Client::View::TitlebarView

- **Tipo:** *Class*;
- **Descrizione:** È il componente del programma che fa la funzione di view per la barra del titolo, dove saranno collocati il menu dell'applicazione e gli shortcut;
- **Padre:** *Backbone.View*;
- **Attributi:**
 - *el: String*
Il tag HTML popolato dalla Titlebar;
 - *events: Object*
Gli eventi verificabili nella titlebar;
- **Metodi:**
 - *generateJava(event: JavaScriptEvent): void*
Richiede al server di generare il codice in linguaggio Java del progetto correntemente aperto invocando il rispettivo metodo di *RequestHandler*;
Parametri:

- * *event: JavaScriptEvent*: Evento;
- *generateJavascript(event: JavaScriptEvent): void*
Richiede al server di generare il codice in linguaggio JavaScript del progetto correntemente aperto invocando il rispettivo metodo di *RequestHandler*;
Parametri:
 - * *event: JavaScriptEvent*: Evento;
- *newProject(event: JavaScriptEvent): void*
Crea un nuovo progetto invocando il rispettivo metodo di *DataManager*
Parametri:
 - * *event: JavaScriptEvent*: Evento;
- *openProject(event: JavaScriptEvent): void*
Apri un progetto invocando il rispettivo metodo di *DataManager*
Parametri:
 - * *event: JavaScriptEvent*: Evento;
- *saveProject(event: JavaScriptEvent): void*
Salva il progetto correntemente aperto invocando il rispettivo metodo di *DataManager*
Parametri:
 - * *event: JavaScriptEvent*: Evento;
- *saveProjectAs(event: JavaScriptEvent): void*
Salva il progetto correntemente aperto con nome specificato dall'utente invocando il rispettivo metodo di *DataManager*
Parametri:
 - * *event: JavaScriptEvent*: Evento;

FAN-IN:

Non ci sono dipendenze IN.

FAN-OUT:

- *RequestHandler*: gestisce la comunicazione con il server;
- *DataManager*: gestisce la persistenza dei dati su file system.

4.4.3 SWEDesigner::Client::View::ToolBarView

- **Tipo:** *Class*;
- **Descrizione:** È il componente del programma che fa la funzione di view per la toolbar dove saranno collocati gli strumenti per editare i diagrammi;
- **Padre:** *Backbone.View*;
- **Attributi:**
- **Attributi:**

- *el: String*
Il tag HTML popolato dalla Toolbar;
- *events: Object*
Gli eventi verificabili nella Toolbar;

- **Metodi:**

- *addElement(event: JavaScriptEvent): void*
Aggiunge un elemento al diagramma alla selezione di uno strumento invocando il rispettivo metodo di *ToolbarModel*;
Parametri:
 - * *event: JavaScriptEvent*: Evento;
- *initialize(): void*
Inizializza ToolbarView;
- *render(): void*
Provoca il render della toolbar in base al diagramma correntemente visualizzato;

FAN-IN:

Non ci sono dipendenze IN.

FAN-OUT:

- *ProjectModel*: si occupa di gestire la parte logica dell'editor;
- *ToolbarModel*: si occupa di gestire la parte logica della toolbar.

4.4.4 SWEDesigner::Client::View::PathView

- **Tipo:** *Class*;
- **Descrizione:** È il componente del programma che fa la funzione di view per il cosiddetto breadcrumb dove viene inserita la posizione corrente;
- **Padre:** *Backbone.View*;
- **Attributi:**
 - *el: String*
Il tag HTML popolato dalla path;
 - *events: Object*
Gli eventi verificabili nella path;
- **Metodi:**
 - *initialize(): void*
Inizializza PathView;
 - *render(): void*
Provoca il render del path in base al diagramma correntemente visualizzato;

- *switchDiagram(event: JavaScriptEvent): void*
Metodo chiamato da evento generato. Switch verso un tipo antistante di diagramma.;
Parametri:
 - * *event: JavaScriptEvent*: Evento;

FAN-IN: Non ci sono dipendenze IN.

FAN-OUT:

- *ProjectModel*: si occupa di gestire la parte logica dell'editor;
- *ProjectView*: si occupa di gestire la parte grafica del model.

4.4.5 SWEDesigner::Client::View::EditPanelView

- **Tipo:** *Class*;
- **Descrizione:** ;
- **Padre:** *Backbone.View*;
- **Attributi:**
 - *currentTemplate: Object*
Il template correntemente caricato e renderizzato;
 - *el: String*
Il tag HTML popolato dalla path;
 - *events: Object*
Gli eventi verificabili nella path;
 - *tagname: String*
Il tag HTML popolato dal pannello;
- **Metodi:**
 - *confirmEdit(event: JavaScriptEvent): void*
Metodo chiamato da evento generato, salva le modifiche apportate ad una proprietà del contenuto selezionato nel pannello;
Parametri:
 - * *event: JavaScriptEvent*: Evento;
 - *execCommand(event: JavaScriptEvent): void*
Metodo chiamato da evento generato, esegue il metodo definito dal nome dell'elemento generante l'evento sul contenuto selezionato nel pannello;
Parametri:
 - * *event: JavaScriptEvent*: Evento;
 - *initialize(): void*
Inizializza la EditPanelView;

- *render(): void*
Provoca il render del pannello in base all'elemento del paper cliccato;
- *reset(): void*
Provoca il reset del pannello;
- *saveCode(event: JavaScriptEvent): void*
Metodo chiamato da evento generato, salva il codice interno ad una customBubble;
Parametri:
 - * *event: JavaScriptEvent*: Evento;
- *switch(event: JavaScriptEvent): void*
Metodo chiamato da evento generato, esegue lo switch in profondità del tipo di diagramma;
Parametri:
 - * *event: JavaScriptEvent*: Evento;
- *switchDiagram(event: JavaScriptEvent): void*
Metodo chiamato da evento generato, esegue lo switch verso un tipo antistante di diagramma;
Parametri:
 - * *event: JavaScriptEvent*: Evento;
- *unembedCell(event: JavaScriptEvent): void*
Metodo chiamato da evento generato, rimuove la bubble selezionata nel pannello dall'innesto;
Parametri:
 - * *event: JavaScriptEvent*: Evento;

FAN-IN:

Non ci sono dipendenze IN.

FAN-OUT:

- *ProjectView*: si occupa di gestire la parte grafica del model.

4.5 SWEDesigner::Server

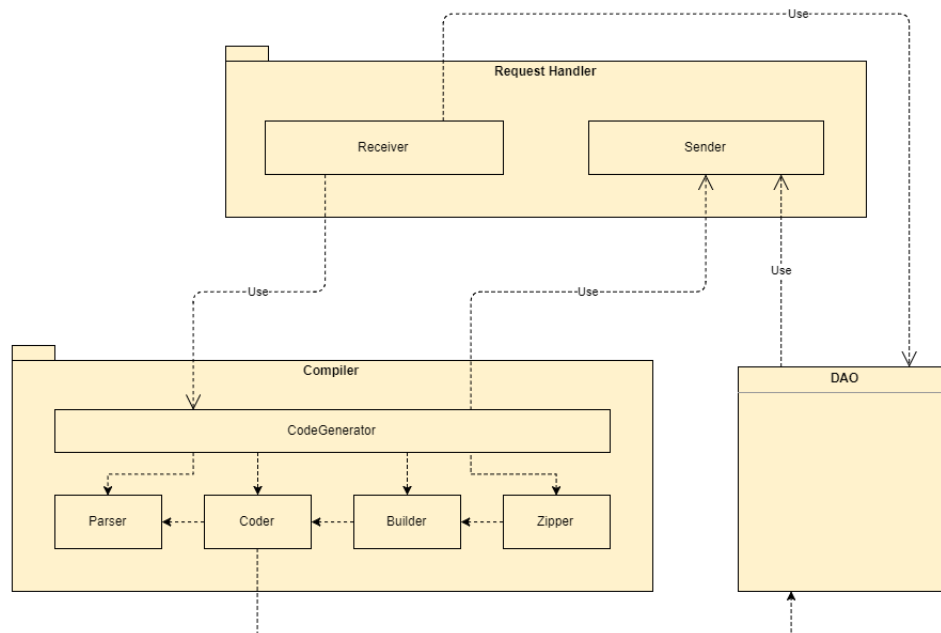


Figure 7: Architettura del server

I package contenuti al suo interno sono:

- SWEDesigner::Server::CodeGenerator;
- SWEDesigner::Server::DAORequestHandler;
- SWEDesigner::Server::RequestHandler.

Questo package non contiene delle classi.

4.6 SWEDesigner::Server::CodeGenerator

I package contenuti al suo interno sono:

- SWEDesigner::Server::CodeGenerator::Builder;
- SWEDesigner::Server::CodeGenerator::Coder;
- SWEDesigner::Server::CodeGenerator::Parser;
- SWEDesigner::Server::CodeGenerator::Zipper.

Le classi contenute al suo interno verranno elencate qui di seguito.

4.6.1 SWEDesigner::Server::CodeGenerator::CodeGenerator

E' il componente che rende disponibile la funzionalità per cui, dato un file valido in formato JSON, restituisce un pacchetto in formato .zip contenente i file del codice sorgente che costituiscono il programma rappresentato dal file in input. I file prodotti sono strutturati in packages, come indicato nel file JSON in input.

FAN-IN:

- Server::RequestHandler::Receiver: si occupa di gestire le comunicazioni in entrata dal client.

FAN-OUT:

- Server::RequestHandler::Sender: si occupa di gestire le comunicazioni in uscita verso il client;
- Parser: si occupa di creare un oggetto che contiene le informazioni ricevute in input;
- Coder: si occupa della traduzione in codice dell'oggetto ottenuto dal Parser;
- Builder: si occupa di organizzare in maniera organica il codice generato dal Coder;
- Zipper: si occupa di creare un archivio .zip contenente in codici sorgente precedentemente creati.

4.7 SWEDesigner::Server::CodeGenerator::Builder

Questo package non contiene dei sottopackage.

Le classi contenute al suo interno verranno elencate qui di seguito.

4.7.1 SWEDesigner::Server::CodeGenerator::Builder::Builder

È il componente che rende disponibile la funzionalità, dato un file JSON in input che rappresenti un programma, di ottenere un oggetto contenitore del codice sorgente corrispondente al contenuto del file di input. Tale codice è suddiviso e strutturato come indicato nel file di input.

FAN-IN:

- Zipper: si occupa di creare un archivio .zip contenente in codici sorgente precedentemente creati.

FAN-OUT:

- Coder: componente che funge da interfaccia alle operazioni di codifica di una stringa permettendo quindi di trasformare le informazioni del file in formato JSON in codice sorgente.

4.8 SWEDesigner::Server::CodeGenerator::Coder

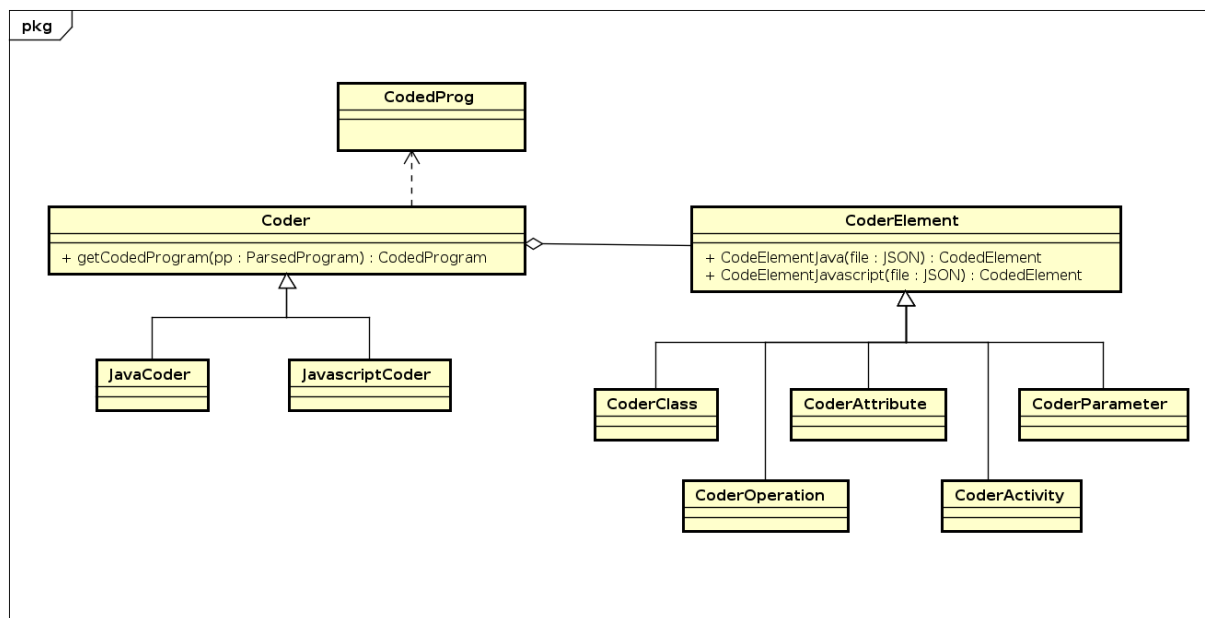


Figure 8: Architettura di Coder

Questo package non contiene dei sottopackage.

Le classi contenute al suo interno verranno elencate qui di seguito.

4.8.1 SWEDesigner::Server::CodeGenerator::Coder::Coder

Componente che funge da interfaccia alle operazioni di codifica di una stringa, in formato JSON che rappresenta un programma valido; tali operazioni permettono di ottenere un oggetto contenente il codice sorgente, in Java o Javascript, corrispondente alla stringa in input.

FAN-IN:

- JavaCoder: si occupa di trasformare un oggetto JSON ricevuto in input in un oggetto contenente il codice sorgente scritto in java;
- JavaScriptCoder: si occupa di trasformare un oggetto JSON ricevuto in input in un oggetto contenente il codice sorgente scritto in javascript.

FAN-OUT:

- CodedProg: componente che contiene il codice prodotto dal Coder;
- CoderElement: componente astratto che offre la funzionalità che permette di associare ad ogni stringa contenuta nel file JSON il corrispondente codice sorgente.

4.8.2 SWEDesigner::Server::CodeGenerator::Coder::JavaCoder

È il componente che rende disponibile la funzionalità, dato un oggetto in input che rappresenta un file JSON parsificato, di ottenere un oggetto contenente il codice sorgente, in linguaggio Java, corrispondente all'oggetto in input.

Non ci sono dipendenze IN.

FAN-OUT:

- Coder: componente che funge da interfaccia alle operazioni di codifica di una stringa permettendo quindi di trasformare le informazioni del file in formato JSON in codice sorgente.

4.8.3 SWEDesigner::Server::CodeGenerator::Coder::JavascriptCoder

È il componente che rende disponibile la funzionalità, dato un oggetto in input che rappresenta un file JSON parsificato, di ottenere un oggetto contenente il codice sorgente, in linguaggio Javascript, corrispondente all'oggetto in input.

Non ci sono dipendenze IN.

FAN-OUT:

- Coder: componente che funge da interfaccia alle operazioni di codifica di una stringa permettendo quindi di trasformare le informazioni del file in formato JSON in codice sorgente.

4.8.4 SWEDesigner::Server::CodeGenerator::Coder::CoderClass

È il componente che mette a disposizione la funzionalità, data una stringa in input in formato JSON che rappresenta una classe valida, di ottenere il corrispondente codice sorgente di tale classe.

Non ci sono dipendenze IN.

FAN-OUT:

- CoderElement: componente astratto che offre la funzionalità che permette di associare ad ogni stringa contenuta nel file JSON il corrispondente codice sorgente.

4.8.5 SWEDesigner::Server::CodeGenerator::Coder::CoderOperation

È il componente che mette a disposizione la funzionalità, data una stringa in input in formato JSON che rappresenta un'operazione valida, di ottenere il corrispondente codice sorgente di tale operazione.

Non ci sono dipendenze IN.

FAN-OUT:

- CoderElement: componente astratto che offre la funzionalità che permette di associare ad ogni stringa contenuta nel file JSON il corrispondente codice sorgente.

4.8.6 SWEDesigner::Server::CodeGenerator::Coder::CoderParameter

È il componente che mette a disposizione la funzionalità, data una stringa in input in formato JSON che rappresenta un parametro di una lista valido, di ottenere il corrispondente codice sorgente di tale parametro. È possibile scegliere fra la codifica in Java o Javascript.

Non ci sono dipendenze IN.

FAN-OUT:

- CoderElement: componente astratto che offre la funzionalità che permette di associare ad ogni stringa contenuta nel file JSON il corrispondente codice sorgente.

4.8.7 SWEDesigner::Server::CodeGenerator::Coder::CoderAttribute

È il componente che mette a disposizione la funzionalità, data una stringa in input in formato JSON che rappresenta un attributo valido, di ottenere il corrispondente codice sorgente di tale attributo. È possibile scegliere fra la codifica in Java o Javascript.

Non ci sono dipendenze IN.

FAN-OUT:

- CoderElement: componente astratto che offre la funzionalità che permette di associare ad ogni stringa contenuta nel file JSON il corrispondente codice sorgente.

4.8.8 SWEDesigner::Server::CodeGenerator::Coder::CoderActivity

È il componente che mette a disposizione la funzionalità, data una stringa in input in formato JSON che rappresenta un diagramma delle attività valido, di ottenere il corrispondente codice sorgente di tale attività. È possibile scegliere fra la codifica in Java o Javascript.

Non ci sono dipendenze IN.

FAN-OUT:

- CoderElement: componente astratto che offre la funzionalità che permette di associare ad ogni stringa contenuta nel file JSON il corrispondente codice sorgente;
- DAO: si occupa di gestire il database delle bubble.

4.8.9 SWEDesigner::Server::CodeGenerator::Coder::CodedProg

È il componente che contiene il codice sorgente prodotto dal Coder.

FAN-IN:

- Coder: componente che funge da interfaccia alle operazioni di codifica di una stringa permettendo quindi di trasformare le informazioni del file in formato JSON in codice sorgente.

Non ci sono dipendenze OUT.

4.8.10 SWEDesigner::Server::CodeGenerator::Coder::CoderElement

Componente astratta che offre la funzionalità di ottenere, data una stringa in input in formato JSON che rappresenta un elemento di classe valido, il corrispondente codice sorgente, in Java o Javascript.

FAN-IN:

- Coder: componente che funge da interfaccia alle operazioni di codifica di una stringa permettendo quindi di trasformare le informazioni del file in formato JSON in codice sorgente;
- CoderClass: componente che permette data una stringa in input in formato JSON che rappresenta un diagramma delle classi valido, di ottenere il corrispondente codice sorgente di tale classe;
- CoderOperations: componente che permette data una stringa in input in formato JSON che rappresenta un'operazione valida, di ottenere il corrispondente codice sorgente di tale operazione;
- CoderAttributes: componente che permette data una stringa in input in formato JSON che rappresenta un attributo valido, di ottenere il corrispondente codice sorgente di tale attributo;
- CoderActivity: componente che permette data una stringa in input in formato JSON che rappresenta un diagramma delle attività valido, di ottenere il corrispondente codice sorgente di tale attività;
- CoderParameter: componente che permette data una stringa in input in formato JSON che rappresenta un parametro valido, di ottenere il corrispondente codice sorgente di tale parametro.

Non ci sono dipendenze OUT.

4.9 SWEDesigner::Server::CodeGenerator::Parser

Questo package non contiene dei sottopackage. Le classi contenute al suo interno verranno elencate qui di seguito.

4.9.1 SWEDesigner::Server::CodeGenerator::Parser::Parser

È il componente che rende disponibile la funzionalità, dato un file JSON valido in input, di ottenere un oggetto contenente le informazioni che costituiscono il file in input.

FAN-IN:

- CodeGenerator: si occupa di restituire in output un archivio zip contenente i codici sorgenti generati a partire dal file JSON ricevuto in input;
- Coder: componente che funge da interfaccia alle operazioni di codifica di una stringa permettendo quindi di trasformare le informazioni del file in formato JSON in codice sorgente.

Non ci sono dipendenze OUT.

4.10 SWEDesigner::Server::CodeGenerator::Zipper

Questo package non contiene dei sottopackage.

Le classi contenute al suo interno verranno elencate qui di seguito.

4.10.1 SWEDesigner::Server::CodeGenerator::Zipper::Zipper

E' il componente che rende disponibile la funzionalità per cui, dato un file valido in formato JSON, restituisce un pacchetto in formato .zip contenente i file del codice sorgente che costituiscono il programma rappresentato dal file in input. I file prodotti sono strutturati in packages, come indicato nel file JSON in input.

FAN-IN:

- CodeGenerator: si occupa di restituire in output un archivio zip contenente i codici sorgenti generati a partire dal file JSON ricevuto in input.

FAN-OUT:

- Builder: componente che si occupa di creare un oggetto contenitore con il codice sorgente, partendo dalle informazioni prese dal file JSON ricevuto in input che rappresenta un programma.

4.10.2 SWEDesigner::Server::DAO

Questa classe si occupa di gestire il database delle bubble.

FAN-IN:

- Coder: componente che funge da interfaccia alle operazioni di codifica di una stringa permettendo quindi di trasformare le informazioni del file in formato JSON in codice sorgente.

Non ci sono dipendenze OUT.

4.11 SWEDesigner::Server::RequestHandler

Questo package non contiene dei sottopackage.

Le classi contenute al suo interno verranno elencate qui di seguito.

4.11.1 SWEDesigner::Server::RequestHandler::Sender

Si occupa di gestire le comunicazioni in uscita verso il client.

FAN-IN:

- CodeGenerator: si occupa di restituire in output un archivio zip contenente i codici sorgenti generati a partire dal file JSON ricevuto in input.

FAN-OUT:

- Client::Model::RequestHandler::Receiver: si occupa di gestire le comunicazioni in entrata dal server.

4.11.2 SWEDesigner::Server::RequestHandler::Receiver

Si occupa di gestire le comunicazioni in entrata dal client.

FAN-IN:

- Client::Model::RequestHandler::Sender: si occupa di gestire le comunicazioni in uscita verso il server.

FAN-OUT:

- CodeGenerator: si occupa di restituire in output un archivio zip contenente i codici sorgenti generati a partire dal file JSON ricevuto in input.

5 Diagrammi di sequenza