

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

Indice

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 architettuale	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::Command	8
4.3.2	SWEDesigner::Client::Model::ConcreteCommand	8
4.3.3	SWEDesigner::Client::Model::State	8
4.3.4	SWEDesigner::Client::Model::dataManager	9
4.3.5	SWEDesigner::Client::Model::projectModel	9
4.3.6	SWEDesigner::Client::Model::toolbarModel	10
4.3.7	SWEDesigner::Client::Model::project	10
4.4	SWEDesigner::Client::Model::RequestHandler	10
4.4.1	SWEDesigner::Client::Model::RequestHandler::Sender	10
4.4.2	SWEDesigner::Client::Model::RequestHandler::Receiver	11
4.5	SWEDesigner::Client::View	11
4.5.1	SWEDesigner::Client::View::projectView	11
4.5.2	SWEDesigner::Client::View::titlebarView	12
4.5.3	SWEDesigner::Client::View::toolbarView	12
4.5.4	SWEDesigner::Client::View::pathView	12
4.5.5	SWEDesigner::Client::View::EditPanelView	12
4.6	SWEDesigner::Server	13
4.7	SWEDesigner::Server::CodeGenerator	13
4.7.1	SWEDesigner::Server::CodeGenerator::CodeGenerator	14
4.8	SWEDesigner::Server::CodeGenerator::Builder	14

4.8.1	SWEDesigner::Server::CodeGenerator::Builder::Builder	14
4.9	SWEDesigner::Server::CodeGenerator::Coder	15
4.9.1	SWEDesigner::Server::CodeGenerator::Coder::Coder	15
4.9.2	SWEDesigner::Server::CodeGenerator::Coder::JavaCoder	16
4.9.3	SWEDesigner::Server::CodeGenerator::Coder::JavascriptCoder	16
4.9.4	SWEDesigner::Server::CodeGenerator::Coder::CoderClass	16
4.9.5	SWEDesigner::Server::CodeGenerator::Coder::CoderOperation	16
4.9.6	SWEDesigner::Server::CodeGenerator::Coder::CoderParameter	17
4.9.7	SWEDesigner::Server::CodeGenerator::Coder::CoderAttribute	17
4.9.8	SWEDesigner::Server::CodeGenerator::Coder::CoderActivity	17
4.9.9	SWEDesigner::Server::CodeGenerator::Coder::CodedProg	17
4.9.10	SWEDesigner::Server::CodeGenerator::Coder::CoderElement	18
4.10	SWEDesigner::Server::CodeGenerator::Parser	18
4.10.1	SWEDesigner::Server::CodeGenerator::Parser::Parser	18
4.11	SWEDesigner::Server::CodeGenerator::Zipper	19
4.11.1	SWEDesigner::Server::CodeGenerator::Zipper::Zipper	19
4.11.2	SWEDesigner::Server::DAO	19
4.12	SWEDesigner::Server::RequestHandler	19
4.12.1	SWEDesigner::Server::RequestHandler::Sender	19
4.12.2	SWEDesigner::Server::RequestHandler::Receiver	20

5 Diagrammi di sequenza 21

Elenco delle tabelle

Elenco delle figure

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 projectView	11
7	Architettura del server	13
8	Architettura di Coder	15

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 UML_G con la relativa generazione di codice Java_G e Javascript_G 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

- **Capitolato_G 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 1° 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_G) è una Single Page Application (SPA_G) scritta con i linguaggi HTML5, CSS_G 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;

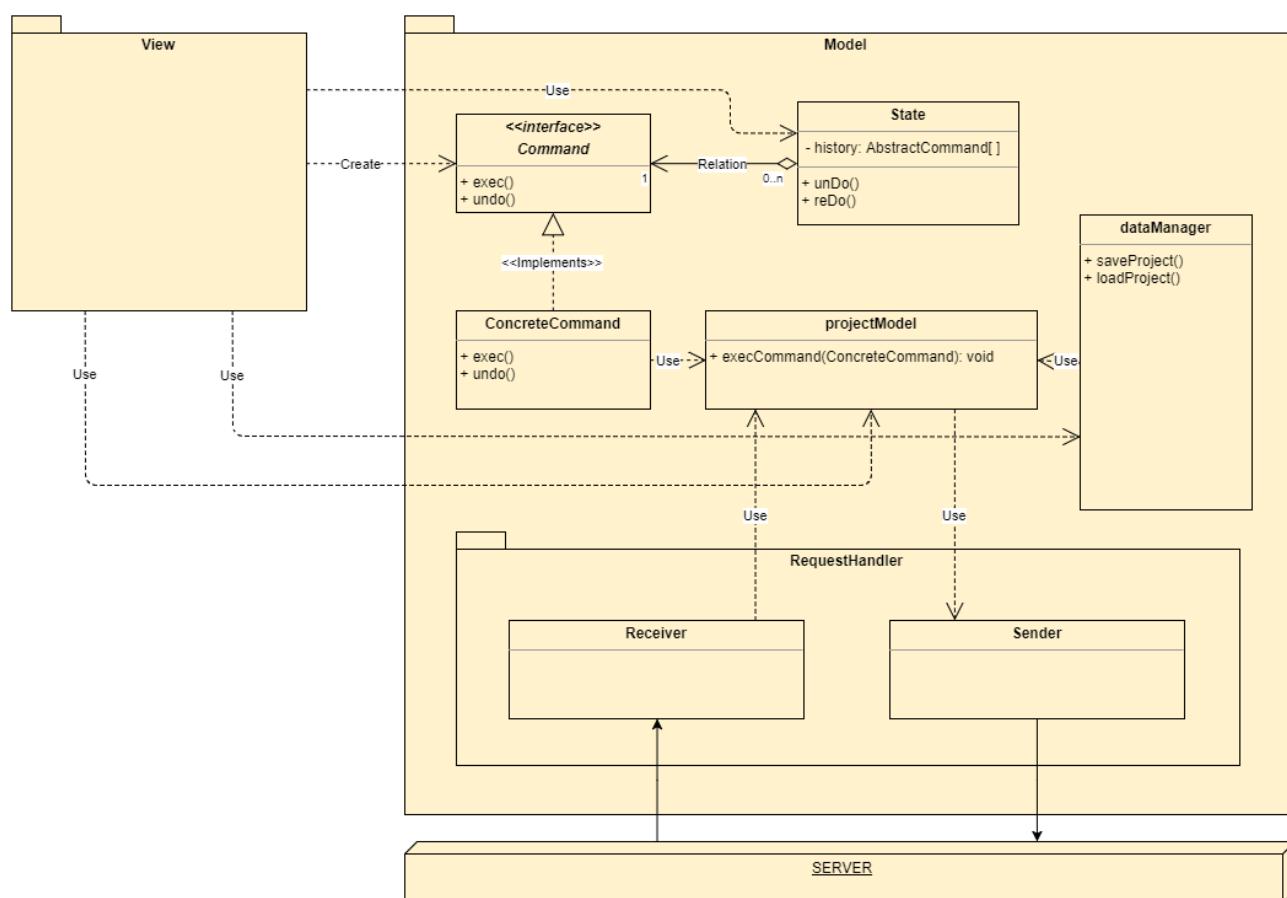


Figura 1: Architettura del client

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 package_G;
- 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-end_G) 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).

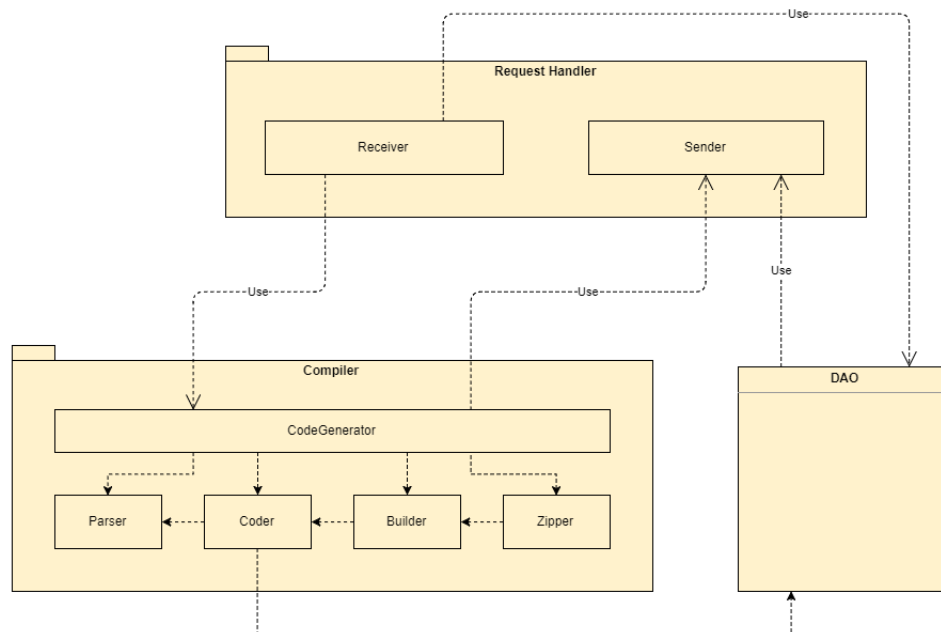


Figura 2: Architettura del server

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.

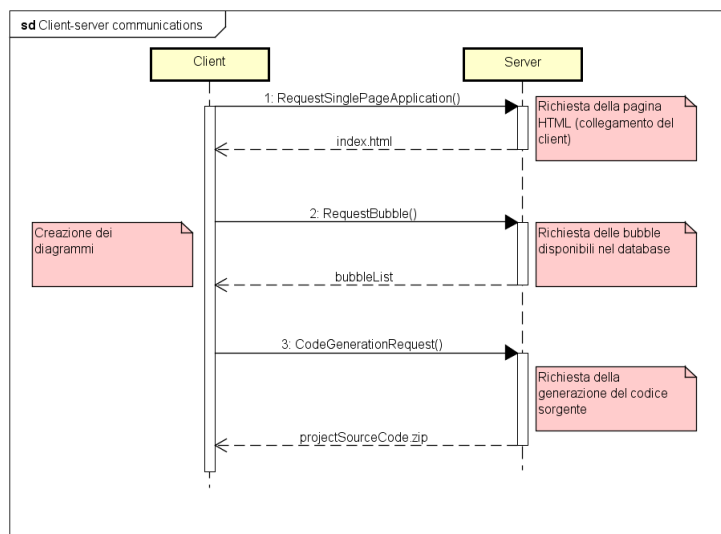


Figura 3: Esempi delle possibili comunicazioni client-server

4 Specifica delle componenti

4.1 SWEDesigner

I package contenuti al suo interno sono:

- SWEDesigner::Client;
- SWEDesigner::Server.

Questo package non contiene delle classi.

4.2 SWEDesigner::Client

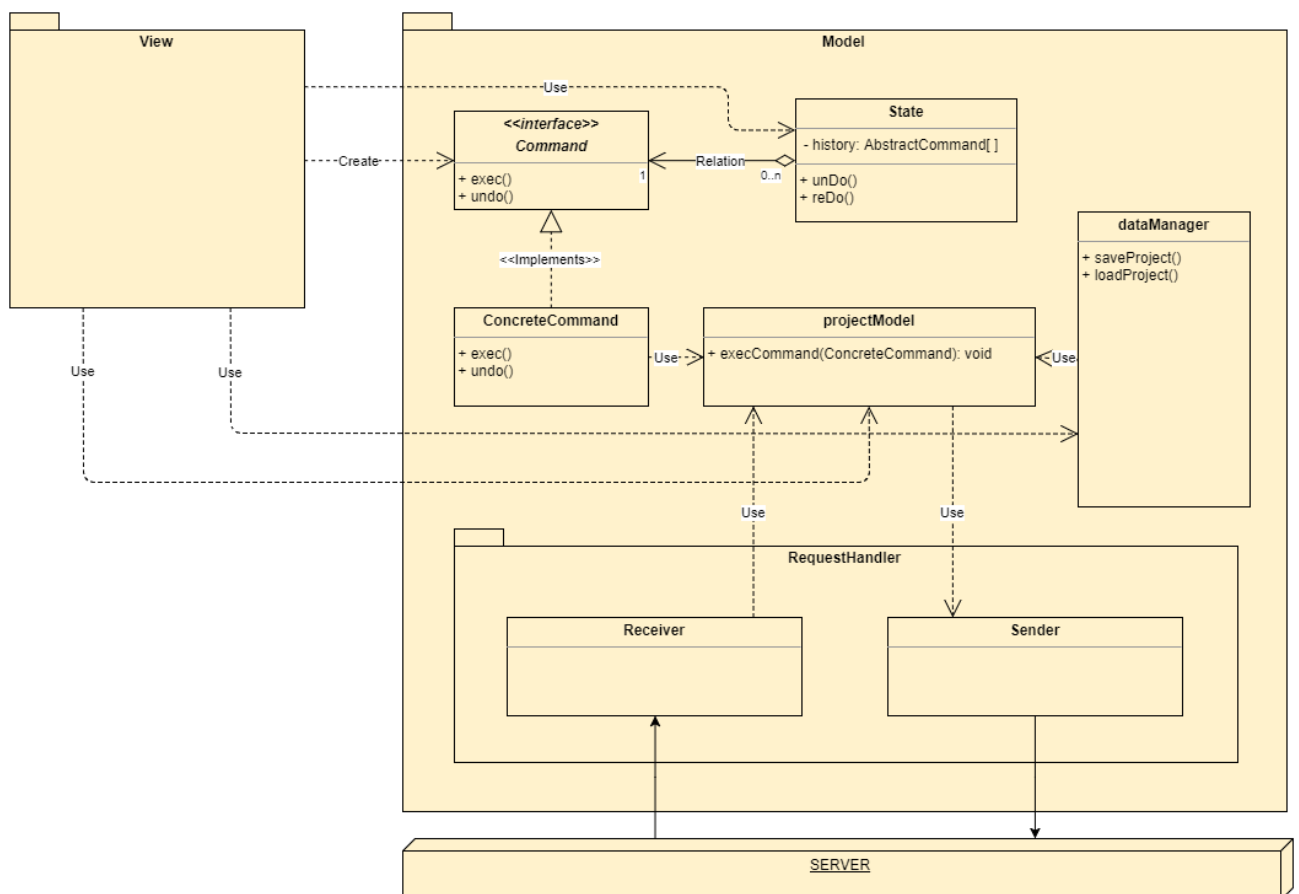


Figura 4: Architettura del client

I package contenuti al suo interno sono:

- SWEDesigner::Client::Model;
- SWEDesigner::Client::View.

Questo package non contiene delle classi.

4.3 SWEDesigner::Client::Model

I package contenuti al suo interno sono:

- SWEDesigner::Client::Model::RequestHandler.

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

4.3.1 SWEDesigner::Client::Model::Command

È l'interfaccia che rappresenta un generico comando impartito dai moduli View ai Model.
FAN-IN:

- ConcreteCommand: implementa l'interfaccia Command per la rappresentazione concreta dei singoli comandi impartiti dai moduli View ai Model;
- View: il componente del programma che si occupa di gestire l'interfaccia grafica;
- State: gestisce la cronologia delle operazioni svolte permettendo le operazioni di unDo e reDo.

Non ci sono dipendenze OUT.

4.3.2 SWEDesigner::Client::Model::ConcreteCommand

Implementa l'interfaccia Command per la rappresentazione concreta dei singoli comandi impartiti dai moduli View ai Model.

Non ci sono dipendenze IN.

FAN-OUT:

- Command: è l'interfaccia che rappresenta un generico comando impartito dai moduli View ai Model;
- projectModel: si occupa di gestire la parte logica dell'editor.

4.3.3 SWEDesigner::Client::Model::State

Gestisce la cronologia delle operazioni svolte permettendo le operazioni di unDo e reDo.

FAN-IN:

- View: il componente del programma che si occupa di gestire l'interfaccia grafica.

FAN-OUT:

- Command: è l'interfaccia che rappresenta un generico comando impartito dai moduli View ai Model.

4.3.4 SWEDesigner::Client::Model::dataManager

Si occupa della persistenza dei dati, in particolare del salvataggio su file system locale del progetto già esistente.

FAN-IN:

- View: il componente del programma che si occupa di gestire l'interfaccia grafica.

FAN-OUT:

- projectModel: si occupa di gestire la parte logica dell'editor;
- project: si occupa di gestire gli elementi contenuti nel diagramma.

4.3.5 SWEDesigner::Client::Model::projectModel

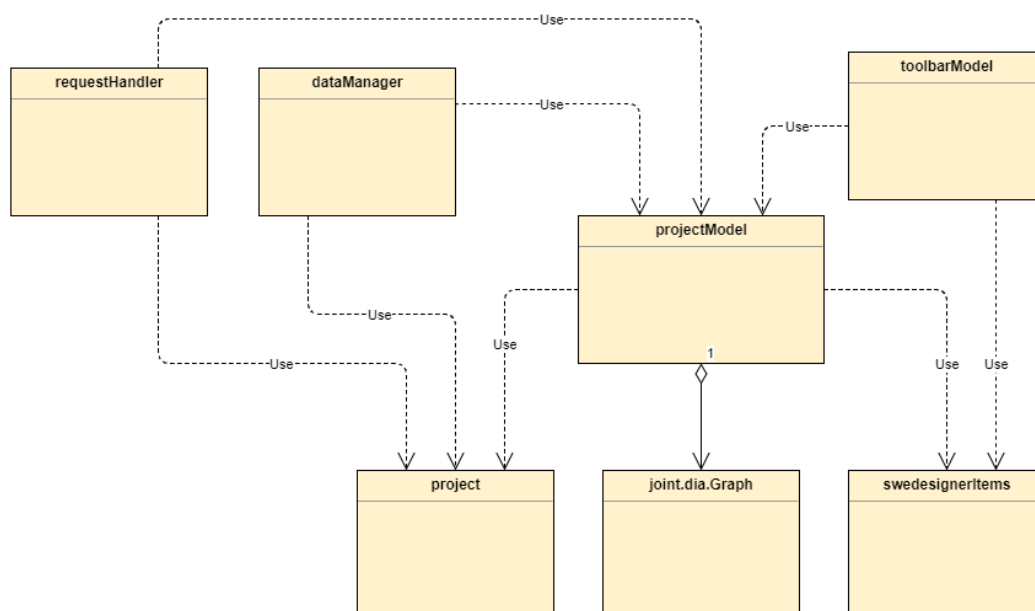


Figura 5: Architettura di Model

È il componente del programma che si occupa di gestire la parte logica dell'editor.

FAN-IN:

- ConcreteCommand: rappresenta i comandi inviati dalle View ed eseguiti poi da 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;
- View: invoca il metodo execCommand;
- Client::RequestHandler::Receiver: si occupa di gestire i dati ricevuti dal server.

FAN-OUT:

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

4.3.6 SWEDesigner::Client::Model::toolbarModel

È il componente del programma che si occupa di gestire la parte logica della toolbar.

FAN-IN:

Non ci sono dipendenze IN.

FAN-OUT:

- projectModel: si occupa di gestire la parte logica dell'editor;
- swedesignerItems: definisce il comportamento degli oggetti contenuti nel diagramma.

4.3.7 SWEDesigner::Client::Model::project

si occupa di gestire gli elementi contenuti nel diagramma.

FAN-IN:

- projectModel: si occupa di gestire la parte logica dell'editor;
- dataManager: Si occupa della persistenza dei dati, in particolare del salvataggio su file system locale del progetto già esistente;
- requestHandler: Si occupa di gestire le comunicazioni con il server.

FAN-OUT:

Non ci sono dipendenze OUT.

4.4 SWEDesigner::Client::Model::RequestHandler

Questo package non contiene dei sottopackage.

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

4.4.1 SWEDesigner::Client::Model::RequestHandler::Sender

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

FAN-IN:

- View: ne invoca i metodi.

FAN-OUT:

- projectModel: si occupa di gestire la parte logica dell'editor;
- project: si occupa di gestire gli elementi contenuti nel diagramma;
- Server::RequestHandler::Receiver: si occupa di gestire le comunicazioni in entrata dal Client.

4.4.2 SWEDesigner::Client::Model::RequestHandler::Receiver

Si occupa di gestire le comunicazioni in entrata dal server.

FAN-IN:

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

FAN-OUT:

- projectModel: si occupa di gestire la parte logica dell'editor;
- project: si occupa di gestire gli elementi contenuti nel diagramma.

4.5 SWEDesigner::Client::View

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

4.5.1 SWEDesigner::Client::View::projectView

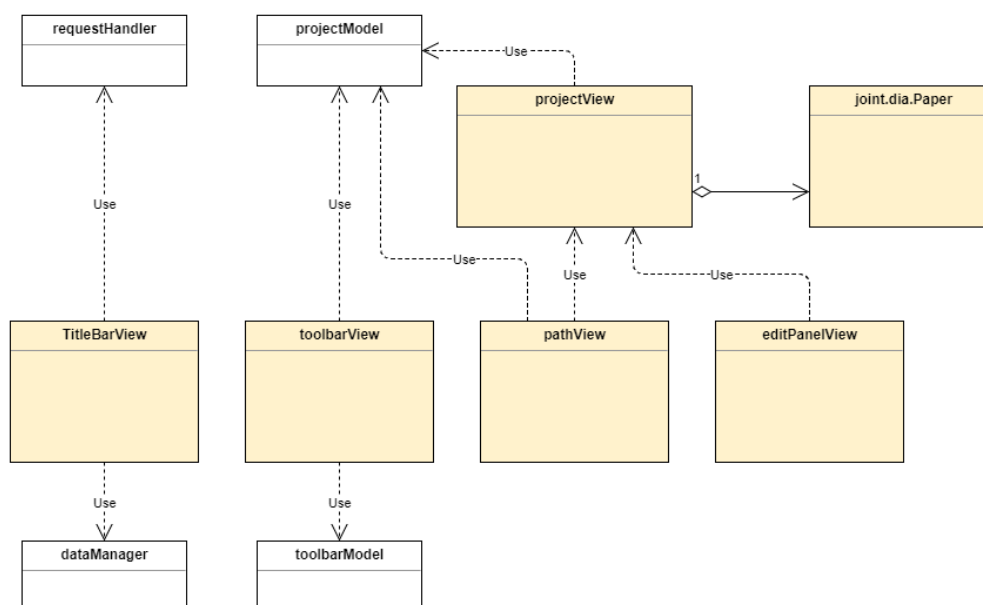


Figura 6: Architettura di projectView

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.

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;
- Paper: gestisce l'interfaccia grafica dell'area dei diagrammi.

4.5.2 SWEDesigner::Client::View::titlebarView

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.

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.

4.5.3 SWEDesigner::Client::View::toolbarView

È il componente del programma che fa la funzione di view per la toolbar dove saranno collocati gli strumenti per editare i diagrammi.

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.

4.5.4 SWEDesigner::Client::View::pathView

È il componente del programma che fa la funzione di view per il cosiddetto breadcrumb dove viene inserita la posizione corrente.

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.

4.5.5 SWEDesigner::Client::View::EditPanelView

È il componente del programma che fa la funzione di view per le informazioni editabili degli elementi che fanno parte dei diversi diagrammi.

FAN-IN:

Non ci sono dipendenze IN.

FAN-OUT:

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

4.6 SWEDesigner::Server

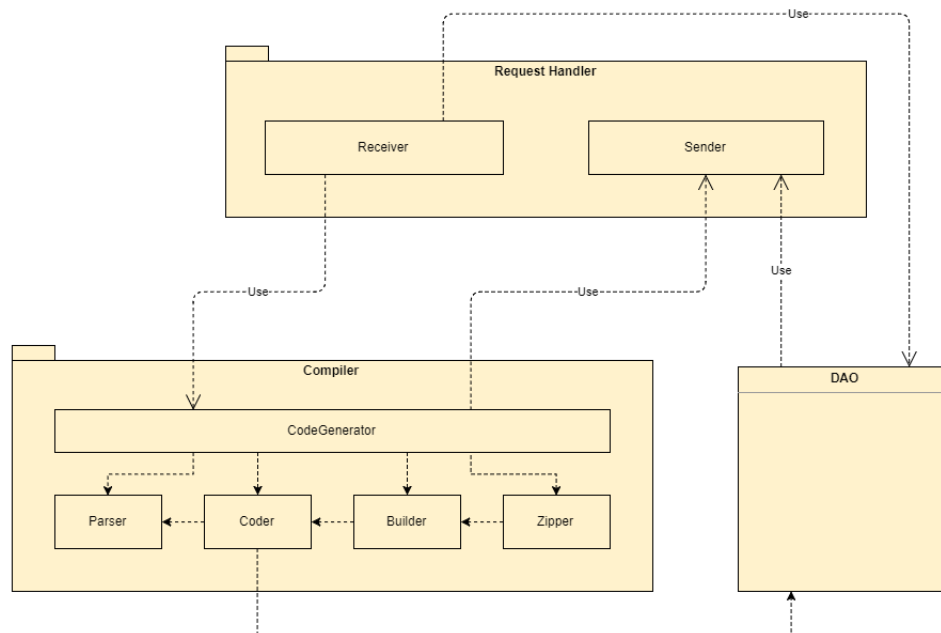


Figura 7: Architettura del server

I package contenuti al suo interno sono:

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

Questo package non contiene delle classi.

4.7 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.

4.7.1 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.

4.8 SWEDesigner::Server::CodeGenerator::Builder

Questo package non contiene dei sottopackage.

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

4.8.1 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.

4.9 SWEDesigner::Server::CodeGenerator::Coder

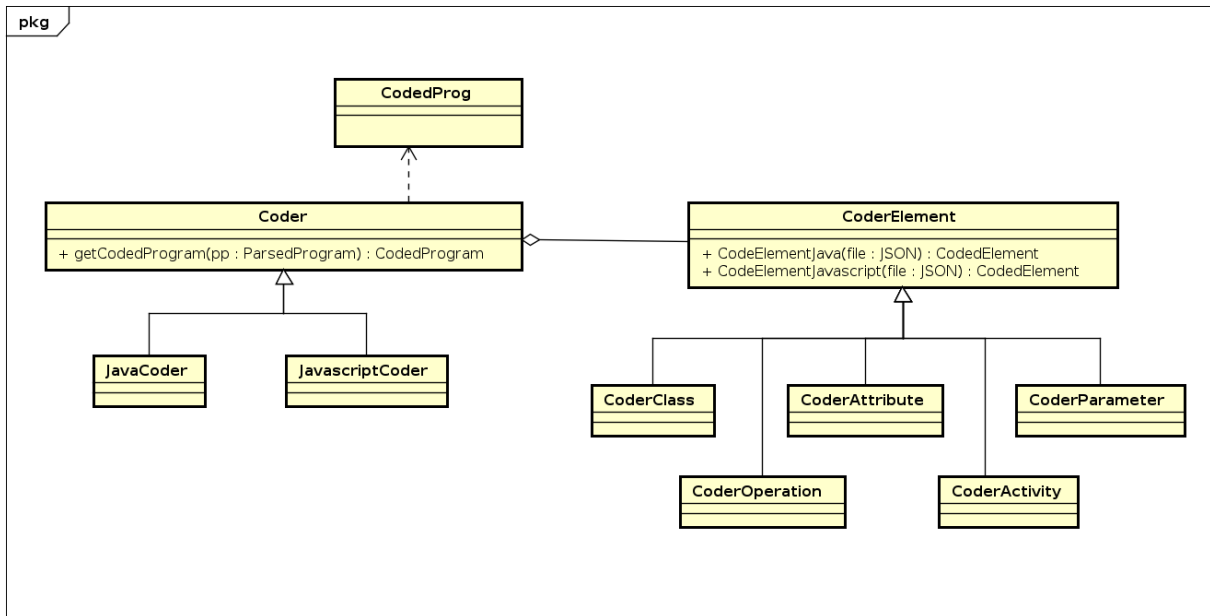


Figura 8: Architettura di Coder

Questo package non contiene dei sottopackage.

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

4.9.1 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.

4.9.2 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.

4.9.3 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.

4.9.4 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.

4.9.5 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.

4.9.6 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.

4.9.7 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.

4.9.8 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.

4.9.9 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.

4.9.10 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.

4.10 SWEDesigner::Server::CodeGenerator::Parser

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

4.10.1 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.

4.11 SWEDesigner::Server::CodeGenerator::Zipper

Questo package non contiene dei sottopackage.

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

4.11.1 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.

4.11.2 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.

4.12 SWEDesigner::Server::RequestHandler

Questo package non contiene dei sottopackage.

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

4.12.1 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.

4.12.2 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