POLITECNICO DI MILANO

MASTER OF SCIENCE IN COMPUTER ENGINEERING

AUTOMATIC GENERATION OF

WEB CRUD APPLICATIONS

Author: Marco Livraghi

Student ID: 835931

Supervisor: Marco Brambilla

Marco Livraghi
ID 835931

Abstract

Automatic generation of web based crud applications

In the recent few years web applications have gained much strength and they are used in many different scopes, replacing some old desktop software and in the mobile development. Web applications bring some advantages like integration, security, scalability, accessibility and reduction of development costs. 67% of world population has access to Internet, reaching percentages of more than 80% in some countries.

The key concept of model driven software engineering is about focusing on modeling the domain of the applications enabling automatic generation of code script. Software generators let developers focusing on business logic, taking care of the basic operation and interfaces.

Many projects and frameworks have been developed to obtain this goal. This work focuses on the idea that generated code must be as simple as possible, to let developers easily modify it. The followed approach grant options on security and documentation side.

This work is based on the open source environment as the proposed implementation is built on top of Spring framework and AngularJS.

Marco Livraghi
ID 835931

Abstract

Automatic generation of web based crud applications

Negli ultimi anni l'interesse e la diffusione delle applicazioni web sono cresciuti esponenzialmente, grazie a diversi framework che ne permettono l'utilizzo nello sviluppo mobile e come alternative al software desktop. I vantaggi garantiti da queste applicazioni sono facilmente intuibili: integrazione, sicurezza, scalabilità, accessibilità e tempi di sviluppo. Oltre il 67% della popolazione mondiale ha accesso a Internet, con picchi superiori all'80% nei Paesi più sviluppati.

I principi dell'ingegneria del software model-driven convergono sull'importanza della rappresentazione del modello del problema, da cui è possibile sviluppare l'applicazione.

Esistono alcuni progetti didattici e commerciali che hanno affrontato questo problema e portano alla generazione di software spesso complesso e di difficile gestione.

L'obiettivo di questa tesi è un generatore di applicazioni web in grado di offrire la massima semplicità nel codice generato, fornendo una solida base per lo sviluppo e facendo guadagnare molto tempo ad un team di sviluppo. L'approccio seguito consente la gestione della sicurezza e della documentazione.

L'implementazione proposta si basa su framework open source del mondo Java, Spring e AngularJS.

Contents

1. Introduction

1.1.   Context

Every year hundreds thousands people gain access to the web by using all kind of devices like smartphone, tablet or desktop computer. In Europe and USA 80% of the people go online every day (1).

This breakneck increase has changed the equilibrium in the development world on behalf of web application, accessible from every device using a browser.
In this scenario, building web application became crucial, while desktop software is losing slices of the market.

Development costs are quite high and developing team waste a lot of time in repetitive and non-core actions that would be automated.

1.2. Problem statement

Full stack developers have in charge the development of both server and client side of an application. The bootstrap of a project is not immediate because the developers must write hundreds line of standard and repetitive code that does not hit the core of the project.

The REST architecture define a standard way to communicate between client and server applications. This imposes a rigid scheme to developers that build APIs similar to each other for every project, losing time and productivity in these simple tasks of copy and paste.

Even on the client side, there is a waste of time for coding of the standard forms for the various resources.

This problem is partially solved by automatic generators that generate the standard code allowing developers focusing on the business logic of the application.

However, automatic generators create source code that is complicated to understand and it cannot be easily modified, so developers often prefer to write their own code and copy-paste code from different APIs. The existing generators do not follow the most recent technology for the web applications and they can be hard to integrate with an already existing system.

Another statement of the problem is the documentation writing: it is often underestimated or ignored, in the best cases it is entrusted to low-level employees.

This brings to documentation that is not helpful for future developers and does not show in detail the specific of the application.

In a RESTful system a complete set of documentation for the REST APIs is fundamental: client side developing can be decoupled and may be commit to a cheaper third part.

1.3.   Proposed solution

The proposed solution will generate a complete set of RESTful services, UI views and documentation. The key idea of the proposed solution is simplicity: generated code must be easy to modify, understand and integrate in the current system.

The generation is divided in two main modules: the REST APIs and the UI files.

These two modules are intended to be completely interoperable with different systems. The UI generated module can be linked with an existing set of REST APIs using every technologies and the generated REST API's completely stick the standard REST architecture.

The implementation proposed has been written in Java and use two different technologies for the server module and for the client module.

The server module is implemented on top of the Spring Framework, in the J2EE World. Among this, it uses Spring Data (a framework to manage persistence operations) over JPA (standard Java Persistence API) and Jackson (a library used to serialize POJO to JSON and deserialize JSON to Java Objects).

The generated REST APIs are documented with the Swagger library, which provides a complete description of every operation and the associated unit testing. In this scenario, the REST APIs can be tested by using a simple user interface.

The UI module is generated over AngularJS framework. AngularJS is a relatively new framework supported by Google and Amazon that brings the MVC pattern to another level in the client side. Bootstrap3 has been used to stylish the UI.

The generation use the concept of annotation, a form of metadata that provide data about a program that is not part of the program itself but they do not provide effects on the operation of the code they annotate.

The crucial point of the generation is the domain model. The input of the generation process must be a set of Java classes modeling all the aspects of the problem and can be annotated with project-specific annotations.

In the first step the information contained in the domain model are interpreted and stored in a database. The information in the database can be easily modified even after the first step by using a UI interface generated by the generator itself.

The second step is the proper generation of the different modules. It takes the data from the database and create the source code of the different modules.

In addition to the standard files, it generates other classes to manage the application configuration, the security of the APIs and the standard navigation bar of the UI interface.

Another aspect that has been implemented is the possibility to generate a small part of the project and to manage the modifies developed. This feature has been developed using GIT concepts at runtime.

1.4. Structure of the thesis

In the first chapter there are general information about the main concepts applied, deepening the concepts of web application, REST architecture, MVC pattern and some details about the technologies adopted.

Then the state of art is analyzed in the chapter 3, discussing the main advantages and critical issues of the existing solution, considering both academic works and commercial solutions.

The idea is explained in the fourth chapter, focusing on the algorithm adopted for the generation and the main features of the framework.

In the fifth chapter there are technical details and issues about the implementation realized.

After that, there is the analysis about testing the realized framework in two real world scenario, before ending with the conclusions.

The last chapters are dedicated to the content index and to bibliography.

2. Context

2.1. Web application

Internet has become a crucial part in our world.

Standing at the PWE Research Center the 67% of the world population has access to the web, with

peaks over 80% in Europe, North America and South Korea.

Figure 1 Percentage of Internet usage

In the last fifteen years number of people having access to the web is increased of 832.5%.

This leads to an evolution in the software engineering and development towards web applications at the expense of desktop software.

 A web-based application brings many advantages.

Accessibility

Users do not need to install applications on their device but they just need a browser to gain access to a service.

Since they just need a browser, that is installed on every device (computer, tablet, smartphone, PDA …), the number of people that can access a web application increases a lot.

Development cost

Users access the system through a uniform environment that is the browser. The application needs to be tested on the main browser and not on all the different operating systems.

2016	Chrome	IE	Firefox	Safari	Opera
April	70.4 %	5.8 %	17.5 %	3.7 %	1.3 %
March	69.9 %	6.1 %	17.8 %	3.6 %	1.3 %
February	69.0 %	6.2 %	18.6 %	3.7 %	1.3 %
January	68.4 %	6.2 %	18.8 %	3.7 %	1.4 %

Figure 2 Browser usage

By testing on the four main browsers developers can cover a percentage of 97.4%.

Integration

A web application has a unique point of access that can be upgraded easily. On the other side, desktop application needs to be reinstalled or upgraded on every device that hosts it.

Interoperability

By following the standards it is quite easy to make a web application communicate with another one, reducing the development time and costs.

Adaptability and scalability

The cloud architecture is perfect for web application to improve their scalability and adaptability. If the workload increases, a new server can be added in a very short time at a very small price. In a cluster of servers, a down of a machine does not affect too much the performance of the entire system.

Security

It is much easier to secure a certain number of dedicated and monitored servers than hundreds or thousands of client.

Variety of technology

There are different stack on top of which a developer can build a web application: the Java solution (J2EE), the .NET platform and the open source alternative, usually with PHP and MySQL.

2.2.   Pattern

Beside software's development there is a fundamental concept: the pattern.

"An architectural pattern is a concept that solves and delineates some essential cohesive elements of a software architecture. Countless different architectures may implement the same pattern and share the related characteristics. Patterns are often defined as "strictly described and commonly available." (2)

Many architectural pattern exist but there is a fundamental one behind web application: MVC. MVC stands for Model-View-Controller and it is a software architectural pattern that divides a generic software application in three interconnected modules, in order to separate the information, the process over the information and the way the information are presented to the user.
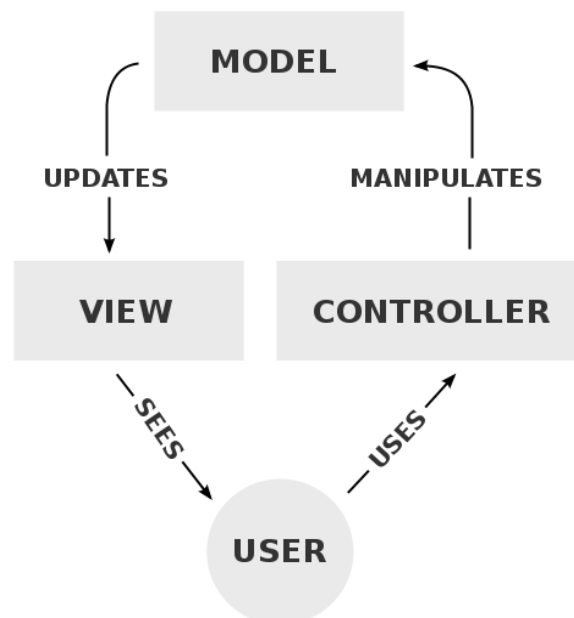
Figure 3 MVC schema

The MVC pattern comes with a few benefits in the development process.

1. Separation of concerns

    The system is divided in three subparts that makes the application completely reusable.

2. Developer specialization

    The development of the business logic is separated from the development of the UI.

3. Parallel development

    Different teams can work in parallel in the business logic and UI part. The development of the UI module, which is usually more frequent, can be done independently from the business logic.

4. Multiple view support

    The MVC pattern allows multiple views that communicate with the same business logic. This comes very useful when developing UI for different devices like smartphone, tablets or desktop computer.

2.3.   REST

REST stands for Representational State Transfer and it is a software architectural style for the World Wide Web. The focus is on components roles and a specific set of interactions between data elements rather than implementation detail.

The term REST was introduces in 2000 by Roy Fielding in his PhD thesis.
Systems are RESTful if they are conform to the REST specification. These systems communicate over Hyper Text Transfer Protocol (HTTP) with the HTTP verbs (HEAD, GET, POST, PUT, DELETE ...), the same the browsers use to retrieve contents for the user. REST systems interact with web resources identified by URI (Uniform Resource Identifier).

The REST architectural style provides a distributed hypermedia system properties like performance, scalability, simplicity, modifiability, visibility, portability and reliability.
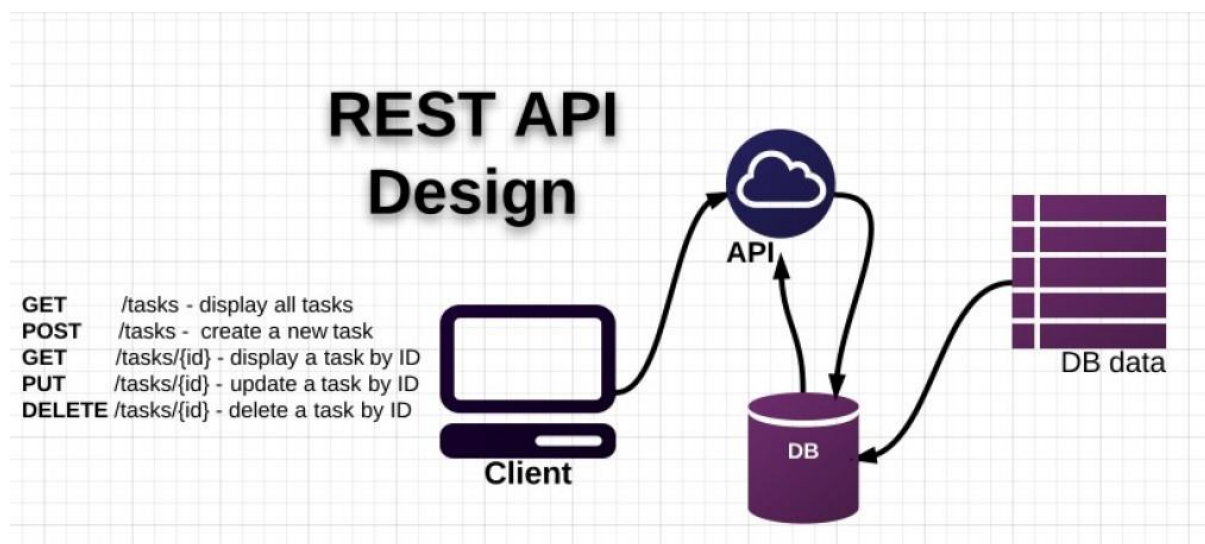

Figure 4 REST API Design

Rest constraint

Client-Server

Client and server are separated concept that implements different layers. Clients are not concerned with data storage and servers are not concerned with UI and user state.

Stateless

Each request from the client contains all the information the server needs to complete the request, the session state is maintained on the client.

Cacheable

Responses can be cached if they have been defined as cacheable. Cache can eliminate some client-server interaction improving the performance

Layered system

The client just make requests but has no idea on which is the server that fulfills them. It may be an intermediate server. This increase scalability allowing load balancing and enforce security.

Uniform interface

Resources are identified by URIs and the server use a format to send data to the client. These representations (Internet media type) may be different from the server data storage.

Many different Internet Media Type exist (XML, Atom...) but JSON is the most used.

2.4. Automatic programming

"*Computer programming is the process of constructing executable code from fragmentary information. … When computer programming is done by a machine, the process is called automatic programming. AI researchers are interested in studying automatic programming for two reasons: First, it would be highly useful to have a powerful automatic programming systems that could receive casual and imprecise specifications for a desired target program and then correctly generate that program; second, automatic programming is widely believed to be a necessary component of any intelligent system and is therefore a topic for fundamental research in its own right.*" (3)

It is a computer programming where some mechanism generates a computer program, enabling a higher abstraction level.

This branch of computer programming led to the development of interpreters, assemblers, compilers and generators, following the concept of generative programming.

The automated source code creation is based on generic frames, prototypes, classes or aspects that improve the developer productivity.

Implementations

- Acceleo: open source code generator plugin for Eclipse IDE used to generate textual language (Java, PHP …) from EMF models defined from any metamodel.
- Altova MapForce: graphical data mapping tool for generating Java, C# or C++ applications.
- Spring Roo: open source code generator for Spring Framework using AspectJ

2.5. Java reflection

"By definition, behavioral reflection allows a program to modify, even at runtime, its own code as well as the semantics and the implementation of its own programming language" (4)

The earliest computer where programmed in the native assembly language, which was reflective. Later, with the introduction of higher level programming language like C, reflection disappeared.

In 1982 Brian Cantwell Smith introduced the notion of computational reflection in programming languages and later it has been introduced in almost every high level programming language.

Reflection is used to observe and eventually modify the runtime behavior of a program. In object oriented programming language reflection allows the inspection of classes at runtime, allowing the call to unknown methods at compile time. This happens also in Java.

Reflection is also the ability of a programming language to be its own metalanguage. This is true for the main high level programming language with the notion on reflection.

2.6. Model-drive engineering

MDE is a software development methodology that focuses on creating domain models, which are the conceptual models of all the topics concerning a specific problem.

This approach leads to productivity increasing: the model are standardized and the design process is simplified as they follow common design patterns.

A model is judged effective if the user can understand it and it is the result of the work of more persons, like managers, designers, developers and users.

Eclipse has his own modeling framework called Eclipse Modeling Framework that allows the code generation starting from a structured data model. It can generate a set of Java classes from a model specification written in XMI (XML Metadata Interchange).

The core of EMF is the metamodel Ecore, which is defined by itself and allows expressing other models. It enables the usage of the whole EMF ecosystem.

2.7.   CRUD

It stands for Create Read Update and Delete, that are the four main operations of persistent storage and defines also the UI conventions based on forms.

Each letter of the acronym can be related to a database statement and a HTTP method.

Operation	SQL	HTTP	DDS
Create	INSERT	PUT / POST	write
Read (Retrieve)	SELECT	GET	read / take
Update (Modify)	UPDATE	POST / PUT / PATCH	write
Delete (Destroy)	DELETE	DELETE	dispose

Figure 5 Crud operations

Crud operation is not a feature of just relation databases as it can bel applied even to object databases, xml database or text fields.

These concepts are relevant at the UI level as they define the operation that the users can do on a resource.

2.8.   Spring

The spring framework is a Java application framework and inversion of control container (5). It contains many features for building web application and it can be seen as a replacement of the Enterprise JavaBeans model.

The first version of this framework was released in 2002.

A core part of the Spring Framework is the inversion of control (IOC) container, which provides a way for managing java objects using reflections: this container manages the objects lifecycle and it is responsible for creating, initialize and configure them.

The objects can be configured with XML files or with Java annotations that provide the configuration information needed to create them.

There are two ways of obtaining objects: dependency lookup and dependency injection.

The dependency lookup is a pattern where the caller ask the container objects with a specific name or of a specific type. On the other hand, dependency injection is a pattern where the container passes objects by name to other objects, using constructors or factory methods.
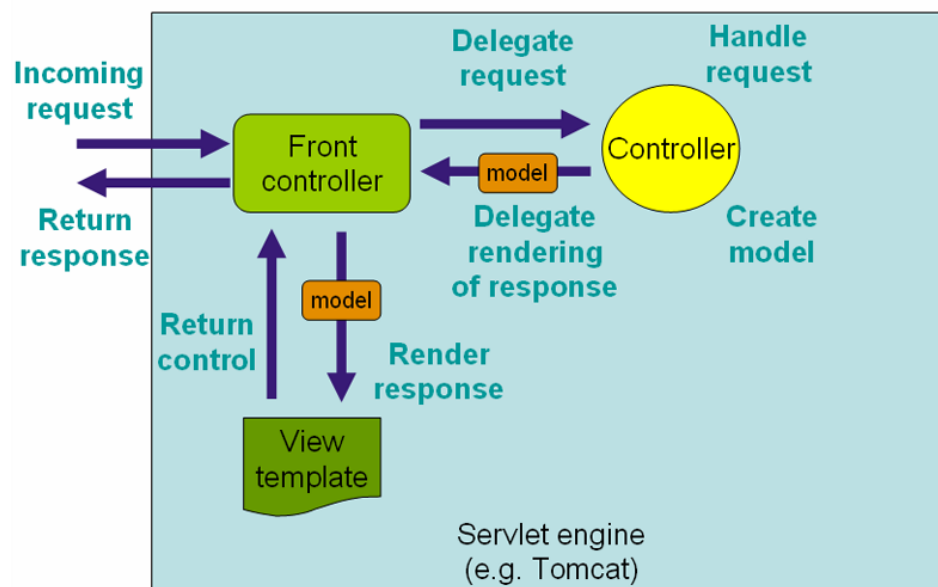

Figure 6 Spring structure

Data access framework

The Spring's data access framework solves the common problems for developers and support the entire popular Java data access framework like JDBC, Hibernate, JPA…

It provides the following features:

- Resource management: to acquire and release database resource
- Exception handling: exception are thrown to an higher level
- Transaction participation, for the transparent management of transactions
- Resource unwrapping: database objects are retrieved from connection pool
- Abstraction for BLOB

Transaction management framework

Spring provides its own transaction management framework that provides an abstract mechanism to the Java platform. It allows developers to work with:

- Global transactions
- Nested transactions
- save points

The Spring's transaction management framework works with JDBC connection, Object-relational mapping and JTA transaction manager.

Model-view-controllers framework

Spring developers thought that the Jakarta Struts Web framework was not enough for building web applications as it did not separate well the model and the handling layer. This was not a key feature in the beginning of the development but became reality after a while.

Therefore, Spring developers worked at their own web framework based on the MVC pattern. The web interfaces are coupled to the Servlet API that grant all the features of the servlets.

Spring has the concept of front controller, a dispatcher servlet that is responsible for delegating control to the various controller during the execution phase of an HTTP request.

Spring Boot

This the ConventionOverConfiguration solution by Spring for rapid application development. This module of the framework allows the developer to build a standalone Spring application that embed Tomcat or Jetty.

It also provide a basic pom.xml configuration and a default Spring configuration.

2.9.   Convention over configuration

Convention over configuration is a software design paradigm. Its goals is to reduce the configurations decided by the developer substituting them with conventions.

The concept has been introduced by David Heinemeier Hansson about Ruby on Rails web framework.

The developer only needs to specify the exceptional configuration that do not follow the convention adopted.

A typical example of convention over configuration resides in the object-relational mapping framework, where classes and field are mapped to table and column following a convention. Developer only specifies different names for those classes/fields that must have a different table/column name.

The basic principle on which this is built is the principle of least astonishment, also used in the user interface's design process. The system must follow the most common behavior if nothing is specified.

2.10. AngularJS

Angular is a web application framework developed mainly by Google but still open-source. It addresses many challenges in developing single-page applications and simplify development and testing of such applications.

The framework adheres at a client-side MVC architecture.

The html pages are enriched with specific custom tag attributes that AngularJS engine reads and interpreters.

Angular is the frontend framework of the MEAN stack, composed of MongoDB, Express, AngularJS and Node.js.

Angular separated two main concepts: user interfaces and application business logic.

For the first one AngularJS belief in declarative programming while the business logic should be written in imperative programming.

To satisfy this constraint the framework adapts and extends the HTML markup language decoupling the DOM manipulation from the business logic.

Scope

Scope can be seen as a "glue" between views and controllers. In more detail, scope is defined as the model and all the variables declared in the scope are visible from the view and from the controller.

The scope is itself a Javascript object that follows the common Javascript notion of scope.

Bootstrap

AngularJS bootstrapper perform different tasks that are divided in three phases

- Creation of a new injector
- Compilation of the directives that enrich DOM
- Link the directives to the scope

Directives are a fundamental support to the developer to define HTML-like components and to define data binding or behavior.

Two-way data binding

A key concept of AngularJS is the two-way data binding. The $scope service detects changes in the model section and modifies HTML in the view via a controller. On the other hand, changes in the view are reflected into the model.

Digest loop

To perform two-way data binding AngularJS makes use of $watch. With the watch service it is possible to define watchers on scope variables triggered by a changement in the model values itself.

Watchers are fired during the $digest loop. This loop is called by the $scope.$digest() function and perform check on the model firing watchers.

There are many directives that automatically trigger a $digest loop, like ng-click, $timeout…

Every angular application has a single root scope, all the other scopes are descendant of the root one.

The digest loop automatically call the $digest() on the $rootScope before calling $digest() on the descendant scopes.

Digest loop is a dirty checking: at the end of the $digest loop Angular engine perform another $digest loop to check if the previous loop changed some model variables from inside some listeners.

AngularJS keeps looping on $digest cycles until no changes in the model are performed or when it reaches the limit of 10.
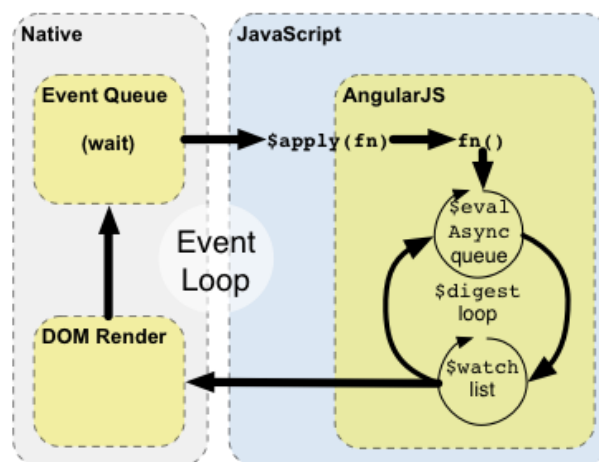
gure 7 AngulaJS life cycle

2.11. Maven

Maven is a build automation tool used in Java projects (6). It is developed by Apache Software Foundation and it is the successor of Apache Ant.

It solves two problems of software engineering as it describes how software is built and its dependencies.

Maven download the requested libraries in a local cache that is populated from a global central repository and can be integrated with artifacts created for specific use by the developer.

The configuration file is the xml representation of the Project Object Model and it is named pom.xml.

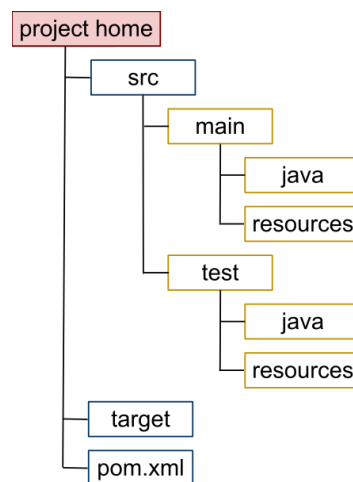A Maven project adopts a standard structure.

Figure 8 Structure of a Maven project

- src/main: contains all the source code files and the resource files
- src/test: contains all the unit test with their resource files
- Target: contains the project built

Maven can be executed from the command line, using the command "mvn".

2.12. Bootstrap

Bootstrap is an open source web framework that helps developers during the design process for websites and web applications.

It is composed of many HTML and CSS based components for forms, buttons and navigation, and some Javascript extensions.

Bootstrap focuses just on the frontend development. First version of the framework has been released in 2011 but the developing is still on as today the Bootstrap team is working at the release 4.0.0.

Bootstrap was born inside Twitter but was soon released as an independent open source framework. It is compatible will the main browsers like Google chrome, Firefox, Microsoft Edge and Safari, for all the most recent versions.

The latest releases of Bootstrap are more focused on the responsive and mobile style.

Bootstrap core technology is based on LESS, an open source style sheet language written in Javascript where style sheets are compiled into Cascading Style Sheets. The compilation allows the use of variables and blocks.

It is based on the CSS block formatting syntax and can be seen as a precompiler for the style sheets.

Bootstrap has a set of predefined stylesheets that add a standard style to an HTML page, which results in a modern and responsive page.

The framework has also a set of custom components that supply standard beahaviour, like dropdown menu, pagination, breadcrumbs, label, panel, alert, input groups and others.

Angular version

Bootstrap has been adapted to the AngularJS logic: the project is called Angular-Bootstrap.

All the original components have been rewritten as AngularJS's directives.

The current version is the 1.3.3 and provides access to helpful components like accordion, pagination, tab, datepicker, button and alerts.

2.13. Terminology

In the following chapters of the thesis main idea and the implementations will be explained in detail. Some terms will be recurrent and are detailed in the following list

- *Entity*

An entity is a single Java class out of the initial domain package. It can be of two different types: "normal" entity (entity from now on) or enumerative entity. Normal entities are POJO beans that represent a table in the database. Enumerative entities are associated with enumerative classes.

- *EntityAttribute*

Entities are composed of several entity attributes. Attributes can be grouped in three different classes: field, enumerative fields and relationships.

- *Relationship*

A relationship is a sort of link between two entities. There exist different type of relationships, based on the cardinalities.

- One to one: the source entity is linked to at most one other entity.
- One to many: the source entity can have many target entities attached
- Many to one: it's the opposite of the one to many
- Many to many: it means that a source entity can be linked to different target entity but even that each target entity can be referenced by more than one source entity.

- *Children entities*

Entities can have different relationships. The term "children entities" referred to an entity indicates the set of target entities of the entity's relationships.

- *Descendant entities*

The concept of children entities is extended at the tree composed of all the children entities of every children entity, in a recursive mode.

- GenerationRun

This term refers to the partial or complete generation from a source model. It is possible to run a generation multiple times on the same project, keeping track of the changement as explained in the chapter 5.6.25.

3. Related work

Some projects address the crud generation problem. Most of them are based on PHP and are listed below.

3.1.   Academic works

Automatic programming and software generation are two fields where the scientists and ICT technician have worked in the past years.

These techniques are applied to different fields in real life and they do not only refer to user interface and web application generators.

3.1.1. Web engineering

Some works in the late 90' and early 2000 addressed the problem of website generation and user interface design, including HDM (7) , HDM-Lite (8), RMM (9),OOHDM (10), Araneus (11), Strudel (12), Hera (13), UWE (14), MIDAS (15)and WebTE (16).

(7)define a method for the description of complex applications without much concern in implementation details. It makes use of the notions of perspective, the identification of different categories of links and the possibility of easily integrating the structure of a hypertext application with its browsing semantics. A Number of links can be derived automatically from a conceptual-design level description.

With (8) has been introduced a new method for the development of web applications, defining a notation that supports the structural, navigational and presentation layer of the application. Conceptually it is an evolution of HDM. Three different layers are formalized in three schema, hyperbase, access and presentation schema respectively.

Relationship Management (9) focuses on a methodology for design hypermedia applications and it is defined by three steps. The first step is about the ER diagram design, a model that represents the domain of the application. During the second phase entities are divided in slices that group together similar information: developer decide how information will be shown to the users. The last phase of RMM is dedicated to navigational design: each associative relationship In the ER diagram is analyzed and eventually considered as navigational path.

OOHDM (10) is a model-based approach for building large hypermedia applications and includes four different activities:

- Conceptual design
- Navigational design
- Abstract interface design
- Implementation

Conceptual design is made using UML.

The most important feature of this method is the inequality between the objects the user navigates and the conceptual objects: the firsts are built, with a view mechanism, from one or more conceptual objects. In the third phase appearance of the view is defined.

Araneus (11) is a project of "Università di Roma Tre" and addresses the problem of defining an integrated environment for unstructured and structured web content, based on a WBMS (Web Base Management System), a database technology for storing both data and metadata describing the structure of website.

Designer 2000 (17) was part of the Oracle Web Development Suite and offered a Web Generator to produce HTML pages by taking in input a model by Designer 2000. It stores metadata in the DMBS.

Strudel (12) defines a graph structure where each page is represented as a node. This project offers data integration but does not provide any user-friendly interface.
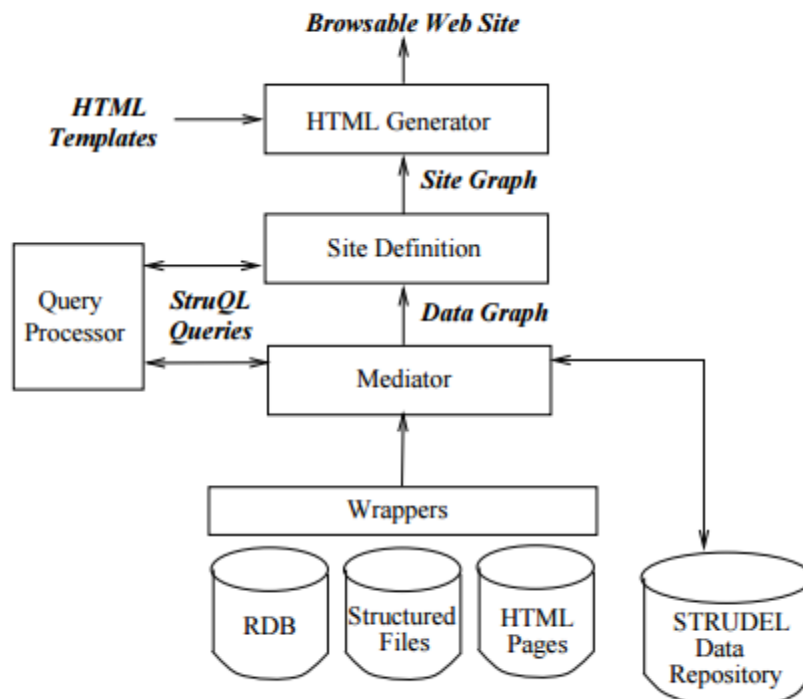
Figure 9 Structure of the Strudel project

Even Hera (13) (a WIS design framework) perspective consists in three layers: semantic, application (navigational view over data) and presentation layer (dealing with concrete rendering like HTML). The Hera framework is based on the WWW standard and has a model-based approach since WIS are data-intensive applications. It is mainly focused on data integration model.

UWE (14) is another important project in the MDE techniques, addressing the problem of model abstraction and transformations in web engineering. Model is transformed to web pages through ATL transformations. UWE makes use of Content model, Navigational model and Process Flow model before arriving at the target Presentation model. UWE metamodel is composed of presentation element categorized in presentation classes and UI elements.
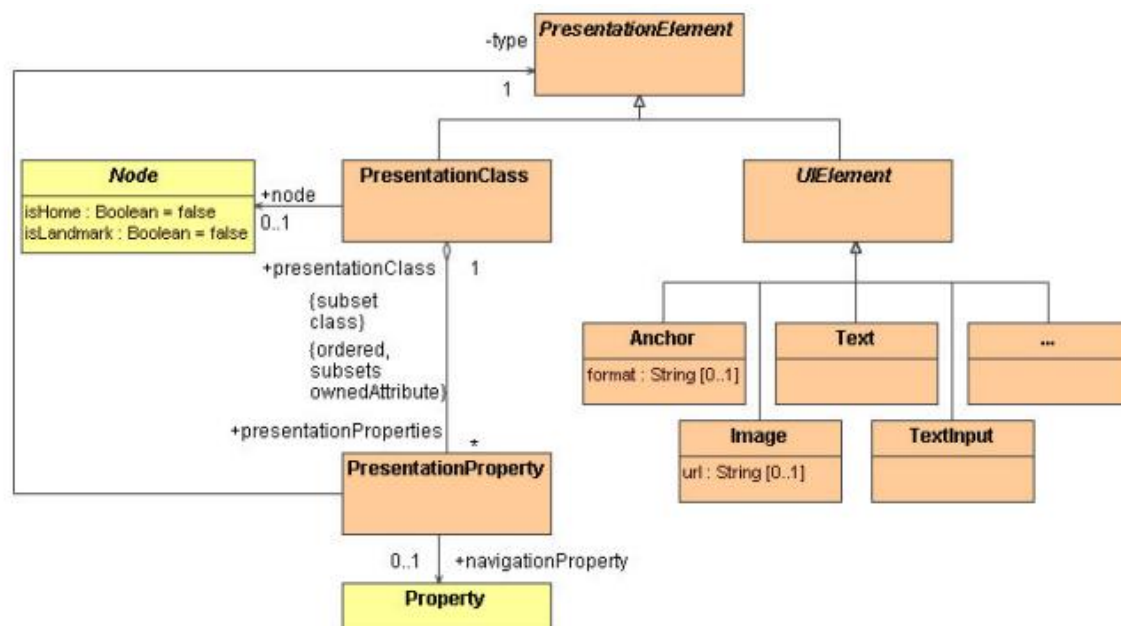
Figure 10 Metamodel of the UWE project

MIDAS (15) is a model-driven methodology based on MDA for the development of Web Information Systems. It categorizes a WIS by means of three orthogonal dimensions: levels, aspects and phases. PIMs and PSMs are represented with UML.
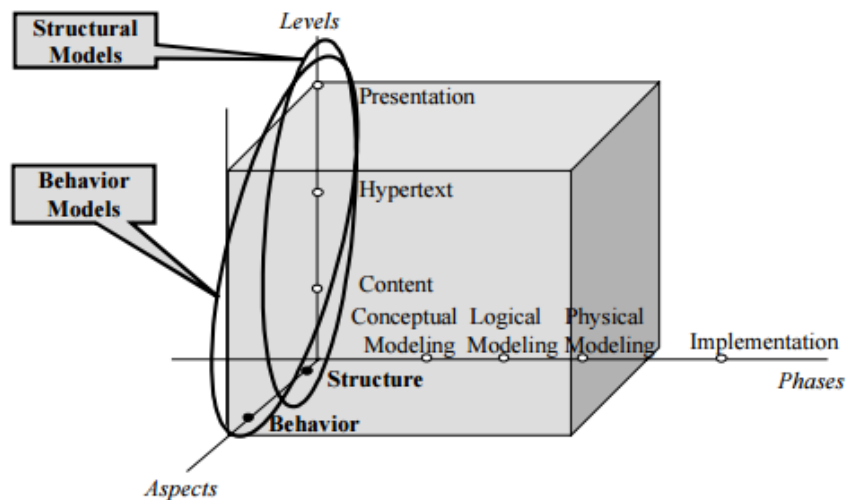

Figure 11 Dimensions of MIDAS project

WebTE (16)is a tool based on the MDWE approach and WebSA (18) that takes in input a model and return the final implementation of a web application by means of model-to-model transformations and model-to-text transformations. The advantage of this method is the complete interoperability with other tools as the models are represented with UML in XMI format readable by any UML tool. WebTE transformation engine is developed using the J2EE platform.
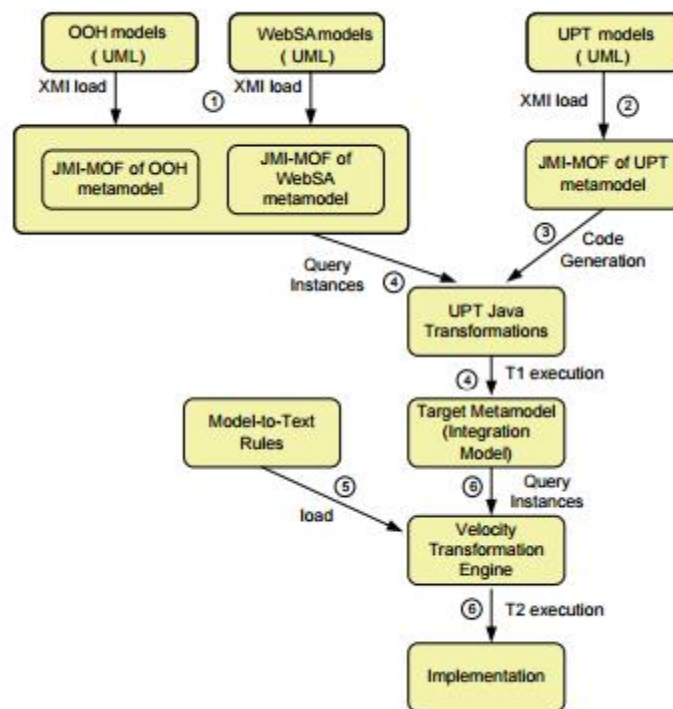
Figure 12 Diagram of the WebTE process

These projects lead to the invention of WebML.

3.1.2. WebML

Web Modeling Language (19) is a notation for specifying complex websites at the conceptual level, enabling high level description of a website considering its structural model, composition model, presentation model and personalization model. It is associated with a graphic notation and a textual XML syntax, platform independent and guarantees a model-driven construction of complex sites.

It was invented at Politecnico di Milano by Stefano Ceri and Piero Fraternali.

A first CASE environment, Autoweb System (20) has been detailed in the year 2000. It is based on HTM-Lite and it consists in two phases, the generation of the database and meta-database and the implementation of the web application.

3.1.3. IFML

Interaction Flow Modeling Language (21) is the successor of WebML (19) and has been built by a team at Politecnico di Milano, including Roberto Acerbis, Aldo Bongio, Marco Brambilla, Sara Comai, Stefano Butti and Maristella Matera.

It has arrived at the beta2 version and in March 2013 has been adopted by the Object Management Group.

It is supported by WebRatio (22) solution and by IFML-Editor (23), an open source editor based on Sirius and Eclipse.

The goal is to provide a graphic tool that permit to define the behavior of the UI with respect to the final user. The tool will then generate the code to fulfill that goal.

IFML supports the platform independent description of graphical UI for web applications as the language allows the description of the view layer.

User interface is described by different ViewContainers that can contain SubContainers and ViewComponents. ViewComponents can have input and output parameters and can be associated with Events, user action mapped in the backend software.

There exist other tools that support MDE development on similar principles of IFML, like Mendix (24), Outsystems (25),OrangeScap (26), LongJump/AgileApps Live (27), Tersus (28) and Softfluent Entities (29).

3.1.4. User interface generation

A common problem in software developing is represented by the user interface. It must be standard through all the different views of the application and can be automatized. HTML or XUIL are perfect examples for the standardization of the user interfaces, each component of the same type is rendered in the same way, based on the operating system and the browser version.

There are studies that addresses this problem which propose a different approach based on software mining.

Software mining is a branch of data mining focused on mining software artefacts such as source files and database schema for useful information related to the characteristics of a system. The main idea consists in analyzing the software metadata and inspect the back-end architecture before creating a native UI suite.

It has been implemented with the project Metawidget (30), a framework that inspects the backend software of different technologies and allows creating user interfaces for a variety of client applications and technologies.

This topic has been addressed even some years ago for the UI of desktop applications. This guaranteed a faster developing in software engineering.

Early examples of model-based tools include Cousin (31) and HP/Apollo's Open-Dialogue (32) that provided a declarative language in which the designer listed the input and output of the user interface. The system then generated the dialogs to display and request the data. These evolved into model-based systems, such as Mike (33), Jade (34) , UIDE (35), ITS (36), and Humanoid (37).

These systems used techniques such as heuristic rules to automatically select interactive components, layouts, and other details of the interface, leading to some difficulties in control.

Developers were required to learn a new language for defining models, not helping the techniques diffusion.

3.1.5. Web application's generation

Another branch of the automatic software generation is related to web applications.

The main idea is to analyze the model of the application in order to be able to generate the server-side software and the related views: the software must be allow to insert, editing, validating and managing entities.

Data-entry applications are typical examples of this pattern.

Business logic and presentation layer should be completely separated, that is why these kinds of application make use of different exchanging format for communications.
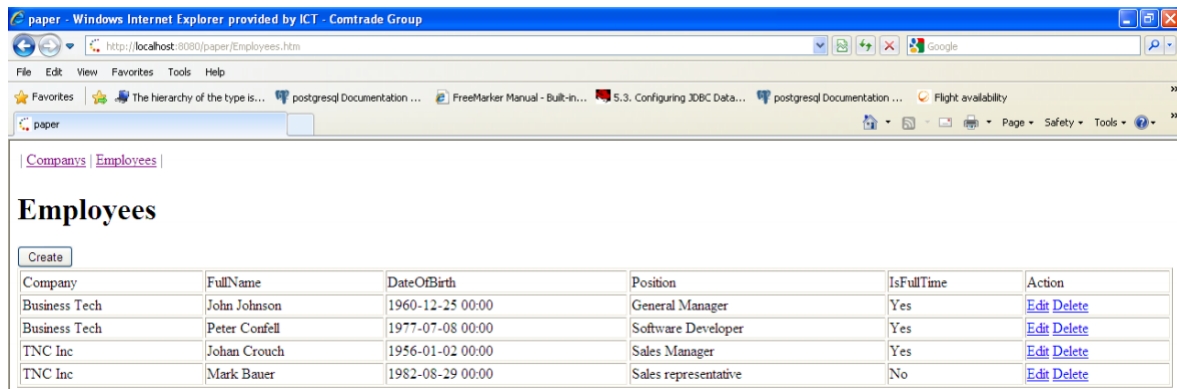
The difference between the various projects is the metamodel's specification used.

Some makes use of XML for defining elements but most are based on the Eclipse EMF framework.

EMF project is an Eclipse modeling framework and code generation facility that operates on source XMI models and produces Java classes. The input of the model may be a set of UML diagrams on top of which the framework applies model transformation in order to obtain the desired result.

A different approach (38)is given by HibernateTools toolset to analyze a database's schema metadata. By obtaining the meta information it is possible to generate the required output. It uses Freemarker ad rendering technology.

Figure 13 Generated list with (37) implementation

The technologies adopted are different as some researches have been developed.

(39) Is based on a source metamodel built with UML and applies transformation that lead to the target model, which is a simplified version of the relational database's schema. Implementation of the mapping rules is based on EMF with a programming approach.

The output of the method is an XML file containing all actions, forms and forward jsp pages.

3.1.6. Code generation from sequence diagram

Starting from a UML sequence diagram is a different approach to code generation.
Sequence diagram can represents the MVC pattern and it is possible to obtain a model transformation from the source model to the target model, by means of a Java PSM as intermediate model.
Some project follows this concept (40) and are able to generate controllers that satisfy the source use case.
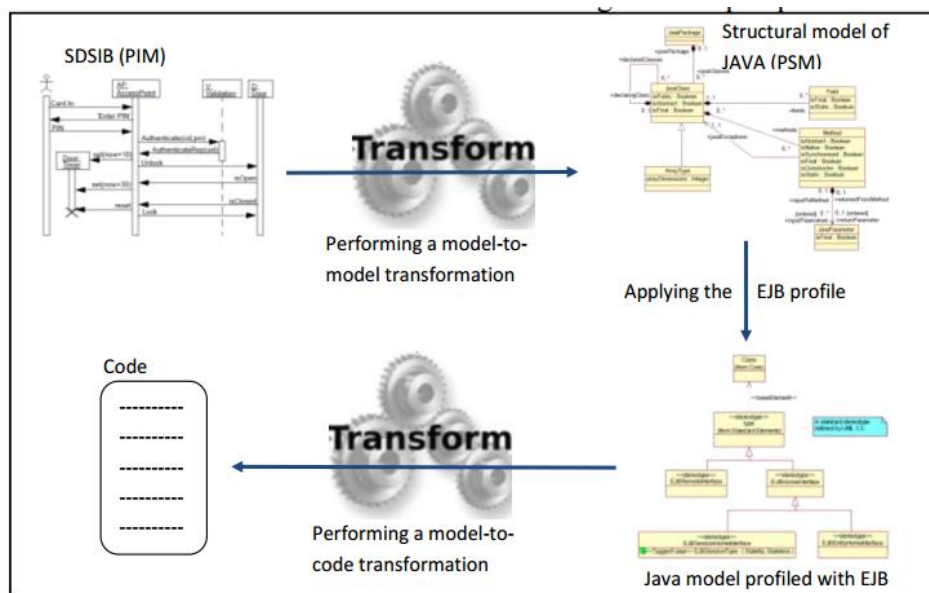

Figure 14 Structure of (39) project

The first phase consists in transforming the source model into a set of Java classes, annotated in order to support relationships. On top of these EJBs it is built an interface to support CRUD operations.
The transformations follow the EMF framework.

3.1.7. Other types of code generation

Code generation is used in many applications field above Crud software.

First use of the code generation comes together with the definition of the first programming languages. Programming languages has been developed from the first generation (machine language) till the fifth-generation (constraint-based), passing through assembly language, high level language and declarative languages (41). Coding a programming language is the first example of code generation.

The AUTOPASS project (42) refers to a system able to translate English-like sentences in software. It uses a geometric database generated prior to compilation and updated during the compilation, which represents the state of the world at each assembly step.

Other project (43) applies automatic programming to recognizing software. Given a description of a 3D object, it generates a software able to recognize the object in images, without restrictions on the orientation of the object in space.

A generator for interactive voice response applications (44) has been patent in 2002. The software is able to inspect the menu section of a website and produce an IVR menu.

Automatic programming has been applied also in the robotics field (45). The proposal of the project is to generate a computer program that enable an autonomous mobile robot to perform simple tasks. It does not apply reinforcement-learning algorithms due to their computational and knowledge issues. It is built with the genetic programming paradigm.
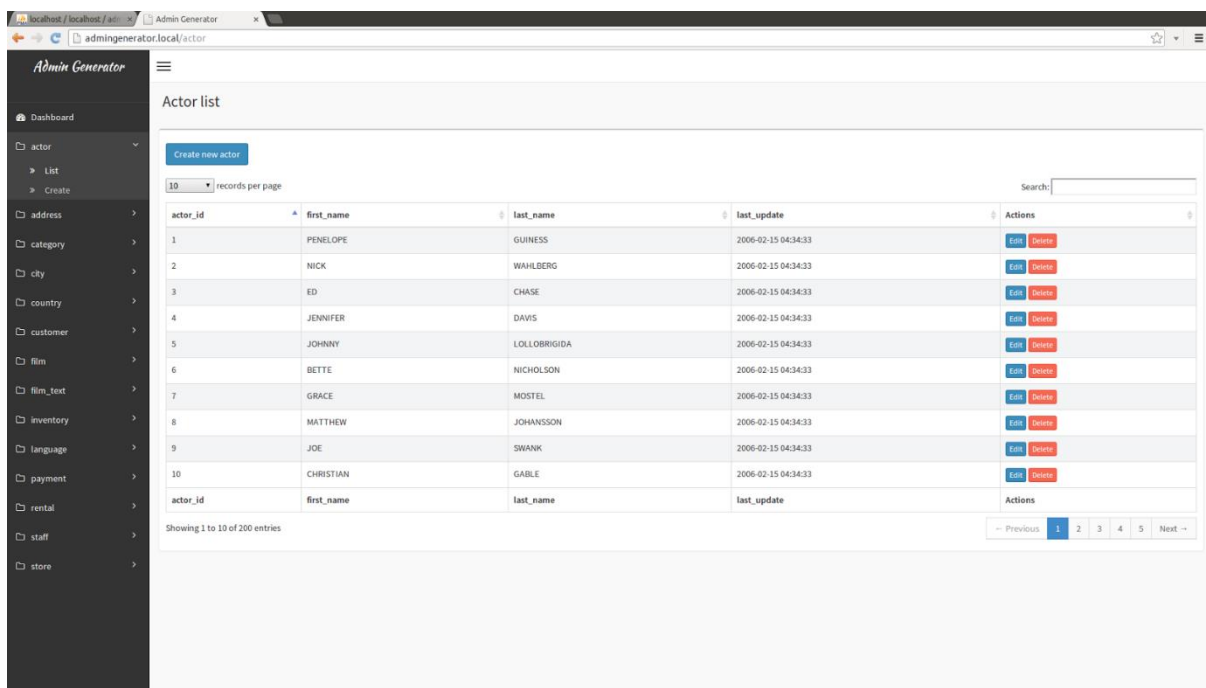
3.2. Commercial solutions

There exist some commercial solutions about this topic. A few frameworks addresses the problem, using different technologies. Some of these are based on PHO and the Silex framework, while the two most developed are based on Java and Spring.

3.2.1. Crud-Admin-Generator

Crud-Admin-Generator is a PHP framework that creates a complete backend starting from a MySQL database. It is based on the Silex framework.

This framework has been developed by a single programmer in 2014 and there is not so much work on it.

It does not support the role management and the views generated are quite basic, they do not show relationships.

Figure 15 Example list generated with CrudAdminGenerator

Another limit is the constraint of a MySQL database.

A force point (in the simplicity point of view) of the project is the complete absence of configuration.

It is enough specifying the MySQL database and the framework generates controllers and views.

This is very quick but on the other side does not permit a custom configuration to make the generate software more suitable to the developer request.

3.2.2. CrudKit

Even this framework is PHP based and it is developed by a team of five contributors since April 2015.

It uses a different approach compared to Crud-Admin-Generator since it does not read metadata information from the database schema but it needs a configuration.

This is a key concept for the developers of this framework as they let the user choose which field they want to visualize.
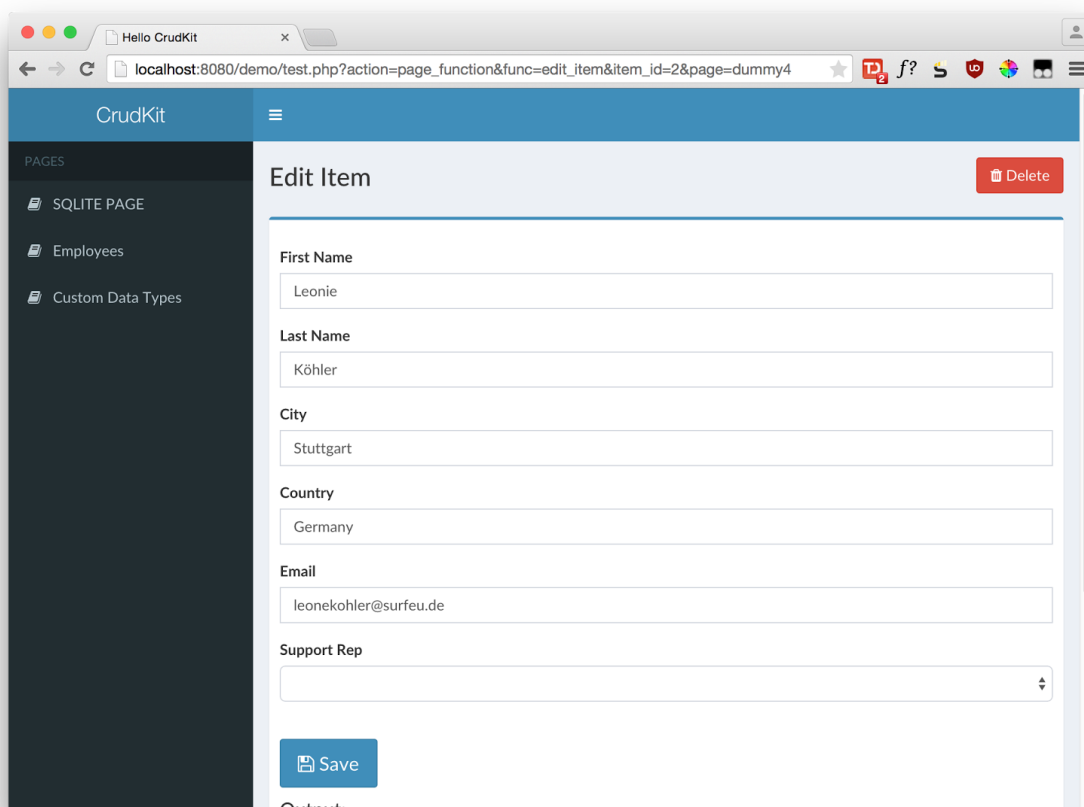

Figure 16 Example of edit page with CrudKit

They propose their framework as an advanced PhpMyAdmin that does not only manage the database table and relationships but also offers a more user-friendly interface (mobile is supported thanks to bootstrap) and the validation.

It is compatible only with MySQL and SQLite and it supports a full Ajax interface.

It does not use ORM and the role management is under development.

Since it does not detect automatically any entity or field, it can be quite hard to define the entire configuration programmatically but assures a full customization for the user of the generated code.

3.2.3. JPA-Modeler Plugin

This is a plugin for code generation integrated with NetBeans IDE 8.0.1 based on Java and Primefaces.

The base model is defined as a UML diagram where the user defines the JPA classes and relationships.

With the first version of this plugin it was possible to generate a basic crud interface based on JSF.

The last developments during the Spring 2016 changed the core logic of the plugin focusing on the entity generation.
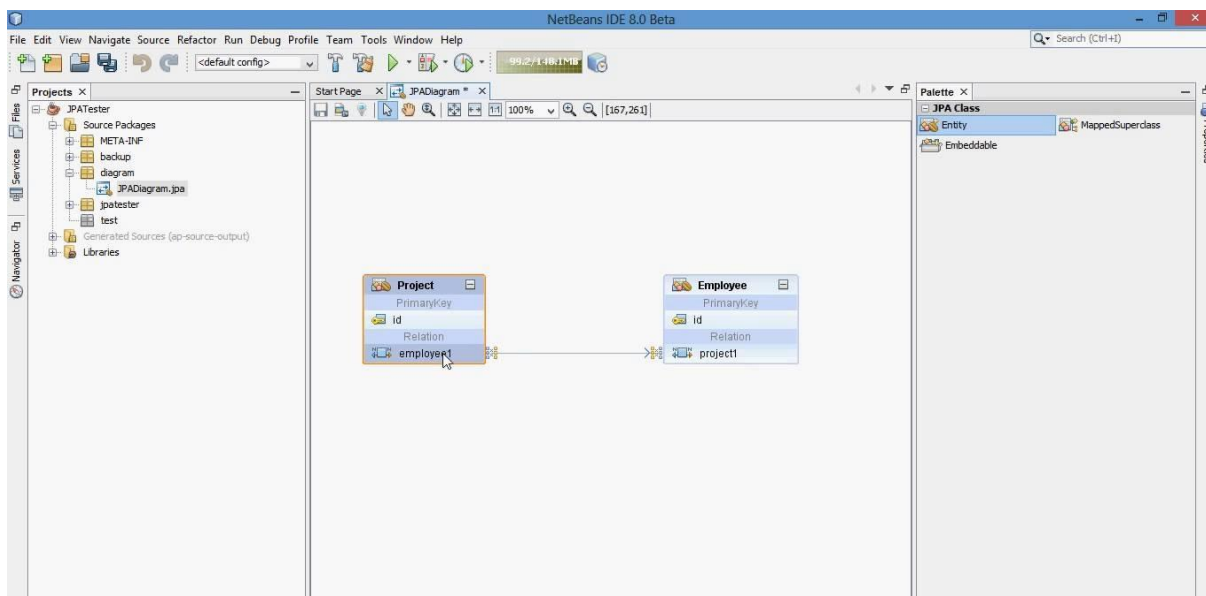

Figure 17 Editor for JPA Modeler plugin

3.2.4. JHipster

JHipster is the closest project to this thesis. The generated code is based on a solid Spring Boot server side and the client part is a modern AngularJS and Bootstrap interface. It is based on yeoman and uses Gulp to grant a powerful workflow.

This framework offers support to a variety of technologies. On the database side, it can manage MySQL, PostgreSQL, Oracle, MongoDB and Cassandra and uses Liquibase for the database versioning.

It offers a powerful search engine given by Elasticsearch and a security level with Spring Security.

The client application is developed in HTML5, using AngularJS, Bootstrap, CSS3, Saas, bower and JQuery.

Fifteen people bring this project forward with the help of many contributors.

The first main release was built in April 2014. In April 2016 the 3.3.0 version was released.

JHipster testing

Installation

Installing JHipster is very easy. After setting up a JDK8 it needs quite standards modules like Git, Node JS, Yeoman, Bower and Gulp.

The IDE must be configured with Maven or Gradle. The usage of JHipster is fully compatible with Eclipse and Intellij.

Setting up a project

The project must be created from scratch from the shell. It is enough typing the following commands to start up the set up application.

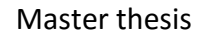*Figure 18 Shell's commands for JHipster startup configuration*

At this point JHipster will ask for the basic configuration options of the project, like name, server port, package name, database, cache and search engine.

Figure 19 JHipster configuration from shell

It is also possible to choose between Maven and Gradle for the dependency management and the unit-testing framework.

By creating a JHipster project, many configuration and management files are created.

Figure 20 Files generated by JHipster

Once the project is set up, it is possible to create entities and relationships among them. It can be done only from command line as the metainformation are stored in a .json file in the project's directory.

Figure 21 Example of entity definition with JHipster

At the end of the field's insertion the developer has to choose a few other configuration options, like DTO object (this feature is in beta), the structure of the tiers (controllers can use directly the repository, use a service eventually with their interface) and pagination options.

```
1   {
2       "relationships": [],
3       "fields": [
4           {
5               "fieldName": "bookId",
6               "fieldType": "Integer",
7               "fieldValidateRules": []
8           },
9           {
10              "fieldName": "title",
11              "fieldType": "String"
12          }
13      ],
14      "changelogDate": "20160610062325",
15      "dto": "mapstruct",
16      "service": "serviceImpl",
17      "entityTableName": "book",
18      "pagination": "no",
19      "microserviceName": "TestApplication",
20      "searchEngine": "elasticsearch"
21  }
22
```

Figure 22 Metadata of a JHipster entity

After having created the domain entities it is necessary to build the client application with the following command:

Yo jhipster:client

This command will create all the Javascript and html files and will run bower and npm install to manage the entities: this operation is quite slow and requires several minutes.

View
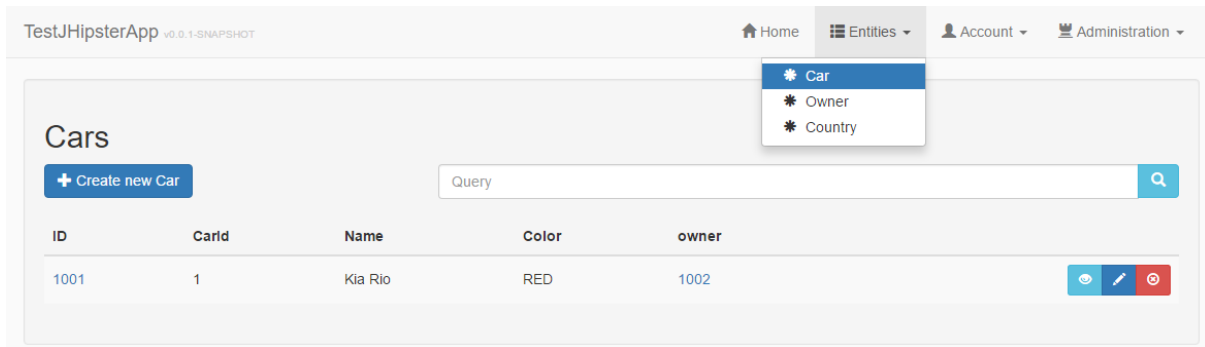
Figure 23 Example of list with JHipster
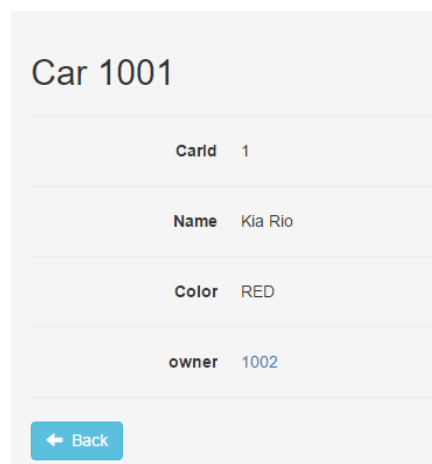
Figure 24 Example of detail with JHipster

The generated view files are user friendly as they are styled with Bootstrap.

Pros

The project is very robust and compatible with all the major systems in the Java world. It has different settings to generate only the server part or the client part, which help the developer. It also offers a complete unit testing and documentation of the REST APIs.

The integration with Elasticsearch for the searches is really well done.

Cons

Managing a second generation from the model is difficult because of the versioning of the database: adding a fields or relationships to an entity will often cause conflicts.

The creation of the entities is tricky as the user must declare them in the JHipster shell, defining all the fields and relationships. In case of complex model, generation cannot resolve all the relationships resulting in an error. A solution to this problem is the usage of JHipster-UML, a tool that transform a UML model saved in the common XMI format into the .json files required by JHipster.

The metamodel is store in JSON files and it is not easily customizable.

The interface supports only 1:n relationships.

Considerations

The generated views does not support a real search form, as it is possible to execute a query with Elasticsearch defining a generic input string.

The strongest point of the project is the capability to interpret xmi model and convert them into the correct metadata files.

3.2.5.   Spring Roo

Spring Roo is an open source software tool that uses convention-over-configuration principles to provide rapid application development of Java-based enterprise software. The resulting applications use common Java technologies such as Spring Framework, Java Persistence API, Java Server Pages, Apache Maven and AspectJ.

Roo works with a separate shell running separately from the IDE, where the developer interact and type some commands. It automatically recognizes changements in the file systems and upgrade the correct files in response.

The main goal of SpringRoo is to guarantee high productivity. A normal java developing cycle may require several days but Roo speeds up this time.

It uses common technologies of the Java stack like Spring, Maven, JSP, JPA, tiles and AspectJ. One of the main advantage is that the developer does not need to learn something new to build applications with Spring Roo.

Spring Roo community offers an integrated version of Eclipse for building projects with Roo: Spring Tool Suite. By means of this IDE, the Roo shell is integrated with the other console and it is very easy to start a Roo project.

SpringRoo testing

The key feature for managing a Roo project is the Roo shell.
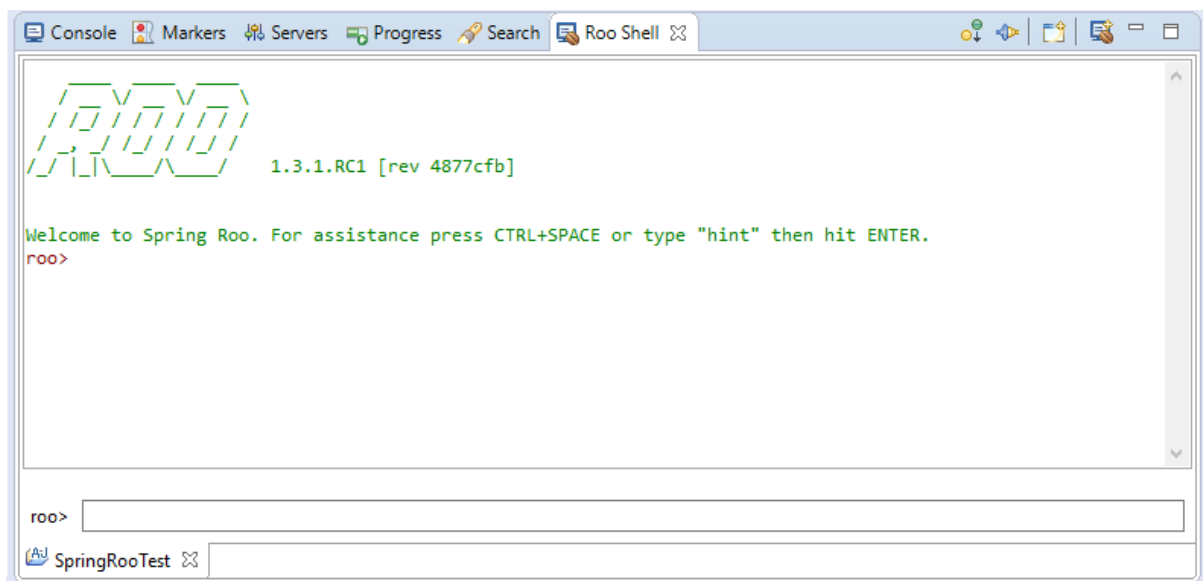
Figure 25 Spring Roo shell

By using its command line, it is possible to run Roo specific command for creating and managing entities and fields that are summarized in the following list.

- entity jpa --class ~.[PackageName].[EntityName] --testAutomatically

This command create an entity with the specified name in the specified package and add the default unit tests. It automatically annotates the entity with the Roo annotation to manage the entity.

- field [FieldType] --fieldName [FieldName] [Validation]

This command has to be run after having selected an entity or created it. It adds a field of the specified type with the specified name. By using command line parameters, the developer can add some validation constraints about size,..

- web mvc all --package ~.[webPackageName]

This command create the MVC classes to manage the entities in the package specified.

- Json all

By means of this command, it is possible to create REST controllers for managing entities.

- finder list --class ~.[PackageName].[EntityName]

This command create the "findBy" methods that can be used to retrieve entities. It is possible to specify special parameters (depth) that tell Roo How many fields should combine in the finder methods

- field   set   --fieldName   [FieldName]   --type   ~.[PackageName].[targetEntity]   --class ~.[PackageName].[sourceEntity]   --cardinality   [ONE_TO_ONE | MANY_TO_ONE | ONE_TO_MANY | MANY_TO_MANY]

This command creates a relationship between source entity and target entity. It is possible to define the typical JPA relationships type to manage the right cardinality of the association.

Views
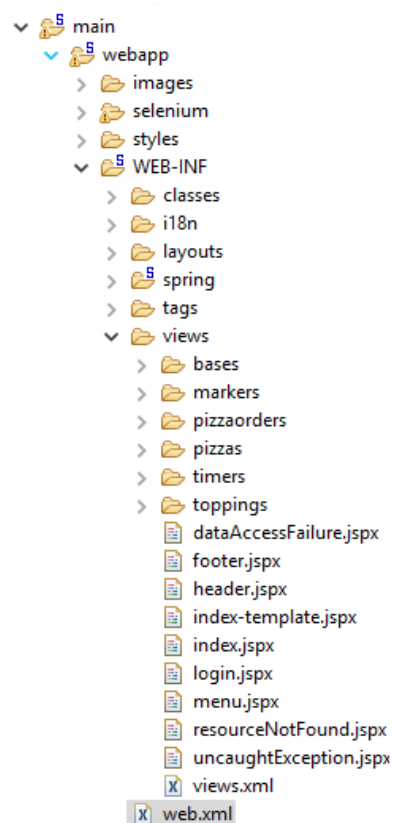
Figure 26 Project structure for Spring Roo

Roo generated view's files in the jspx format and syntax.

It generates a folder for each entity in the domain and create four .jspx for each one, to manage the following operation: create, list, show and update.

Each folder also contains an xml files that map each .jspx to its associated URL.

In the default structure, Spring Roo also generates different files to manage the layout and the typical exceptions, together with the menu file.

Figure 27 Example of list generated with Spring Roo

Consideration

Roo shell is very powerful as it allows an easy generation of entities and fields. The most important feature is the auto detection of changement in the model classes: a new field added by the developer in the Java class is instantly recognized by Roo that add the field to all the generated files.

It is also quite easy to add a complete new Java class in the domain: the developer must only annotate it with some Roo annotation like

- @RooJavaBean
- @RooToString
- @RooJpaActiveRecord
- @RooJson
- ..

The cons of this framework are related to conversion of existing project.

It is hard to convert an existing Maven/Spring project adding the Roo features, the best solution is to start with a new Roo project, add the domain classes and manually annotate them.

The generated views are very trivial and it is not simple add custom code as it is necessary to modify all the four different view files for each entity.
The other con about the view's files is the fact that relationships are not navigable by user.
The generated application is not a single page application.

Figure 28 Example of detail view with Spring Roo

Figure 29 Example of update view with Spring Roo

4. The Idea

4.1. Main goal

The thesis's goal is to generate a usable, robust, scalable and easy-to-understand basis for developers starting from the domain classes.

Generating a complete web application that the developer does not need to modify is almost impossible, so the focus is on generating the code as readable as possible, leaving out all possible details that would make the situation more complex.

4.2. General structure

The generated code can be divided in two different modules, the server part and the client one. The two modules must be standalone and the communication between them should be done in a standard way, following the REST architecture.

This guarantees interchange and interoperability between the modules and already existent part of the software.

4.3.   Requirements

4.3.1. input

The input of the framework must be the model of the project that the developer wants to automatically generate. The input model could assume different forms like Java classes, xmi model or database schema.

4.3.2. Configuration

The key feature of the project must be the convention over configuration. This means that without any configurations the framework must work in the standard way.
The configuration options should be understandable from everyone and applicable in an easy way

Relationships
It must be possible to manage all different kind of relationships between entities.

- one to one
- one to many
- many to one
- many to many

Field Type

The managed field types must be complete:

- Numeric

- String

- Date

- Enumerative

- File

- Embedded field

Validation

The generated view must guarantee a standard validation on the fields by performing commons quality check like

- Not null

- Not blank

- Size

- Email type

- Password type

- Regular expression

4.4. Algorithm

The generation can be divided in two different problem, interpret the model and render the output source files.

4.4.1. Model interpretation

This part analyze the input model and store the structured information. This leads to the creation of a metadatabase that contain the information in a structured way.

```
initProject();
initPackages();
For all package in packages
        createEntityGroup();
        createRestrictionEntityGroup();
        initClasses();
        For all class in classes
                createEntity();
                manageRestrictionData();
                initTabs();
                For all tab in tabs
                        createTab();
                        initFields();
                        For all field in fields
                                createField();
                                manageAnnotation();
                        End for
                End for
        End for
End for
```

The algorithm is quite simple thanks to the completeness of the meta database. It is a complete scan of the declared Java package by means of Java reflection.

4.5.   Components

4.5.1.   Server-side

Each entity will be generated among with different components following the three-tier logic of application design.

Repository layer's component

This layer is an abstraction between the data access layer and the business logic layer of the application.

It is responsible for data manipulation's operations on the database.

Service layer's component

This layer contains the business logic of the application and support crud operations for the objects.

Controller layer's component

This component is between the business logic of the service layer and the view layer: it offers the possible crud operations as API following the REST architecture

4.5.2. Client side

Service component

It is responsible for the communication with the server side application and provides some business logic to the client application.

Controller component

Contains all the possible action callable from the UI, like basic CRUD operations and additional function to improve the user experience.

Template component

It is responsible for the presentation of the content to the user.

4.6.   Key Concept

In this paragraph, the key concepts adopted will be introduced.

4.6.1.   Generation type

To make the framework as usable as possible was introduced the concept of GenerationType.

This helps the developer at configuration level in order to set a default on a resource.

It can have two different values

- Show include: this means that the view files will be generated using only those entity attributes annotated with @Include

- Hide Ignore: this means that by default the generator will include all entity attributes in the view files, excluding those annotated with @Ignore

This setting can be set at two different level: at project level and at entity level.

If the entity level is not specified the framework will consider the project level.

4.6.2. Security

Security was a key concept for the framework to make the difference with the state of art.

The generated application will have different levels of security and multi roles management.

Multirole

A user can be associate with multiple roles, each of them allowing or preventing a certain operation on a resource.

Security type

Security can be set in two different way, using the SecurityType:

- Block with restriction: the users are normally allowed, the framework will block those that have a restriction on that resource

- Access with the permission: the default behavior of the framework will be to block the requests, giving access only to the users with an explicit permission

The permission to a user is given at four different levels, following the crud logic

- can search

- can update

- can insert

- can delete

Based on the permissions the user will perform his operation.

The rights are defined at three different level

- entitygroup

- entity

- field

If the permissions are not set the framework will use the parent's one or the default one.

4.7. Metamodel

The metamodel of this thesis is composed of several classes, divided in four packages:

- Project

- Entity

- Field

- Security

4.7.1. Project package
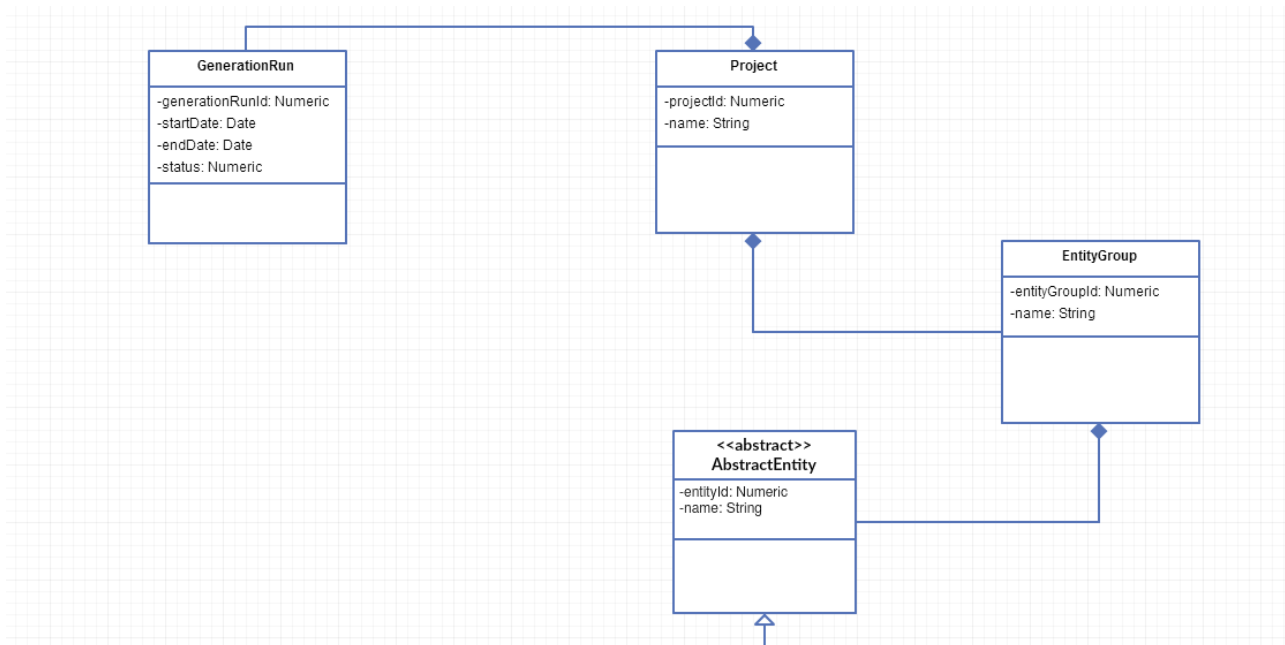
Figure 30 Metamodel's project package

A project is characterized by a name that must be unique and can be associated with many GenerationRun.

When the developer decide to generate the complete project, or a part of it, a new generationRun is saved and associated to the project. The generationRun has two timestamp values for the start/end times and a status that indicates whether the generation was successful or not.

4.7.2. Entity package
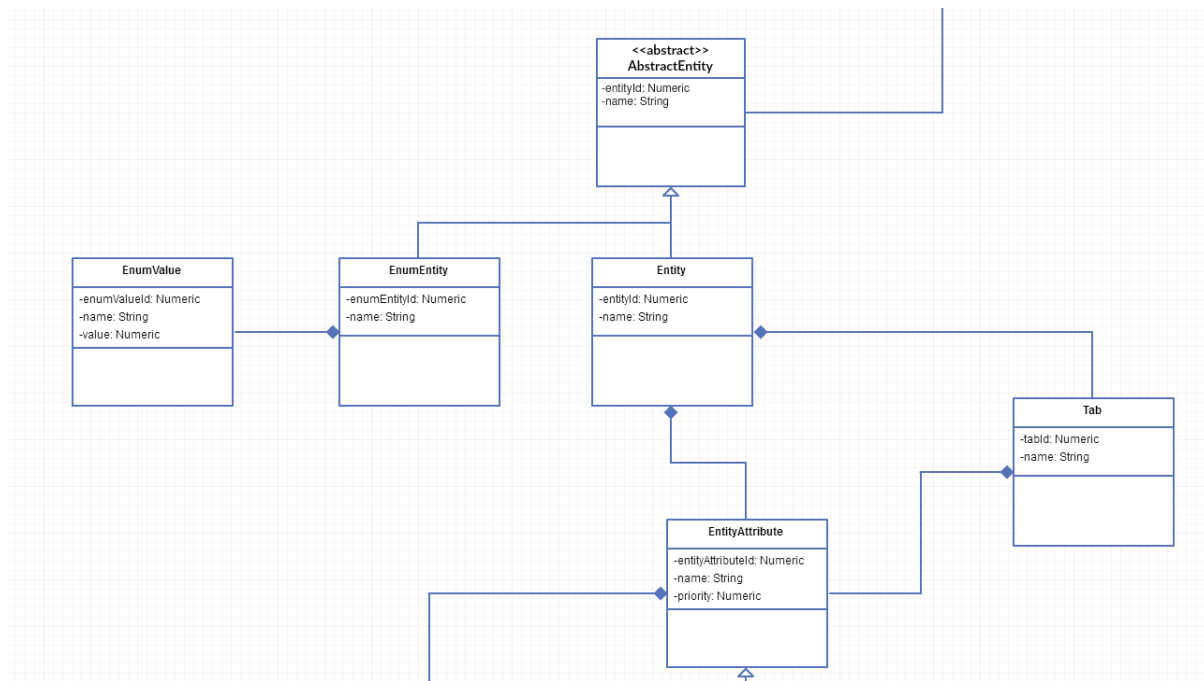
Figure 31 Metamodel's entity package

This is the key package of the thesis.

Entity can be of two different types: enumerative entities and the others.

Enumerative entities contains a certain number of values.

Entity can be grouped together in Entity groups.

All these classes are characterized by a name that describe them.

Entities contains entity attributes that can be grouped in tabs.

Entity attributes are characterized by a name and by a priority: entity attributes will be sort by priority in the form generation.

4.7.3. Field package
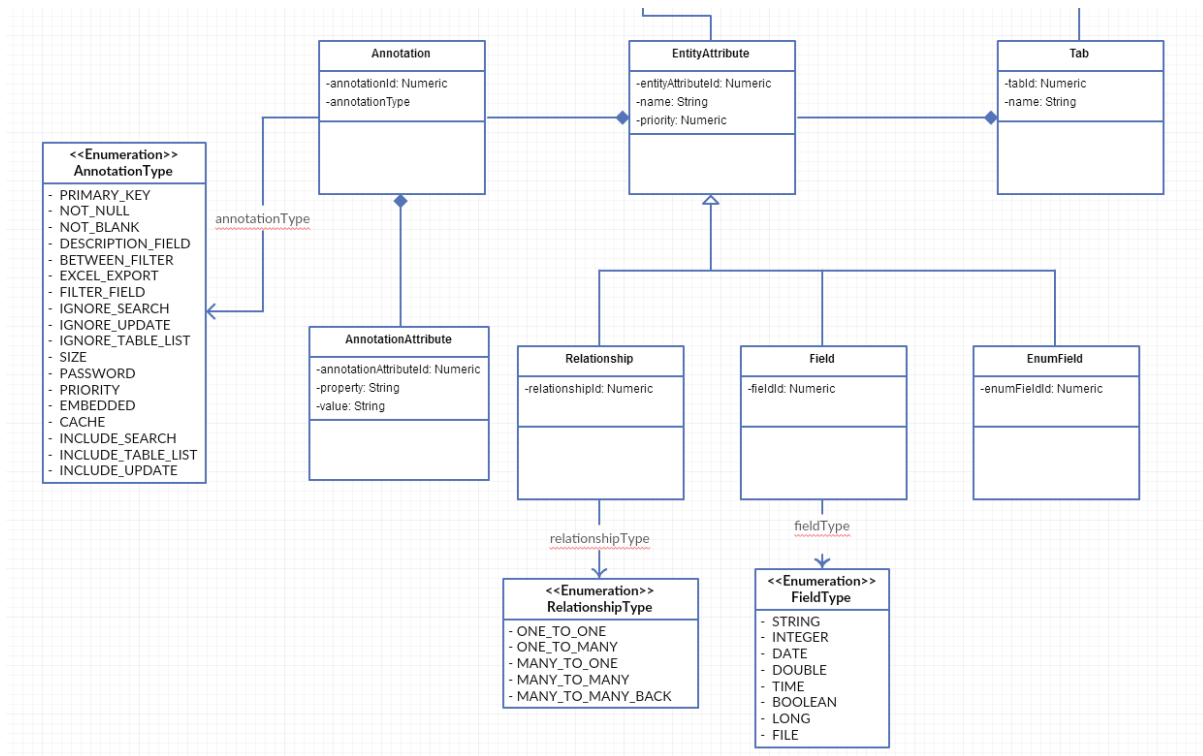
Figure 32 Metamodel's field package

An entity attribute can assume various types.

- EnumField

- Field

- Relationship

EnumField are linked with an enumerative entity.

Field can be of different types. The implementation will cover these specific types:

- String

- Integer

- Date

- Double

- Time

- Boolean

- Long

- File

Each of this field type will be properly rendered in the view files.

Relationships link together two different entities: one is the parent entity and the other is called target entity. Even this kind of entity attribute can assume different types:

- One to one

- One to many

- Many to one

- Many to many

- Many to many back

"Many to many back" has been added as a simplification for the developer.

Annotations are one of the key feature of the thesis. They are connected with entity attributes and characterized by an annotation type.

This is the list of the annotation type managed.

Annotations may also include different annotation attributes, useful to define the annotation behavior.

4.7.4. Security package
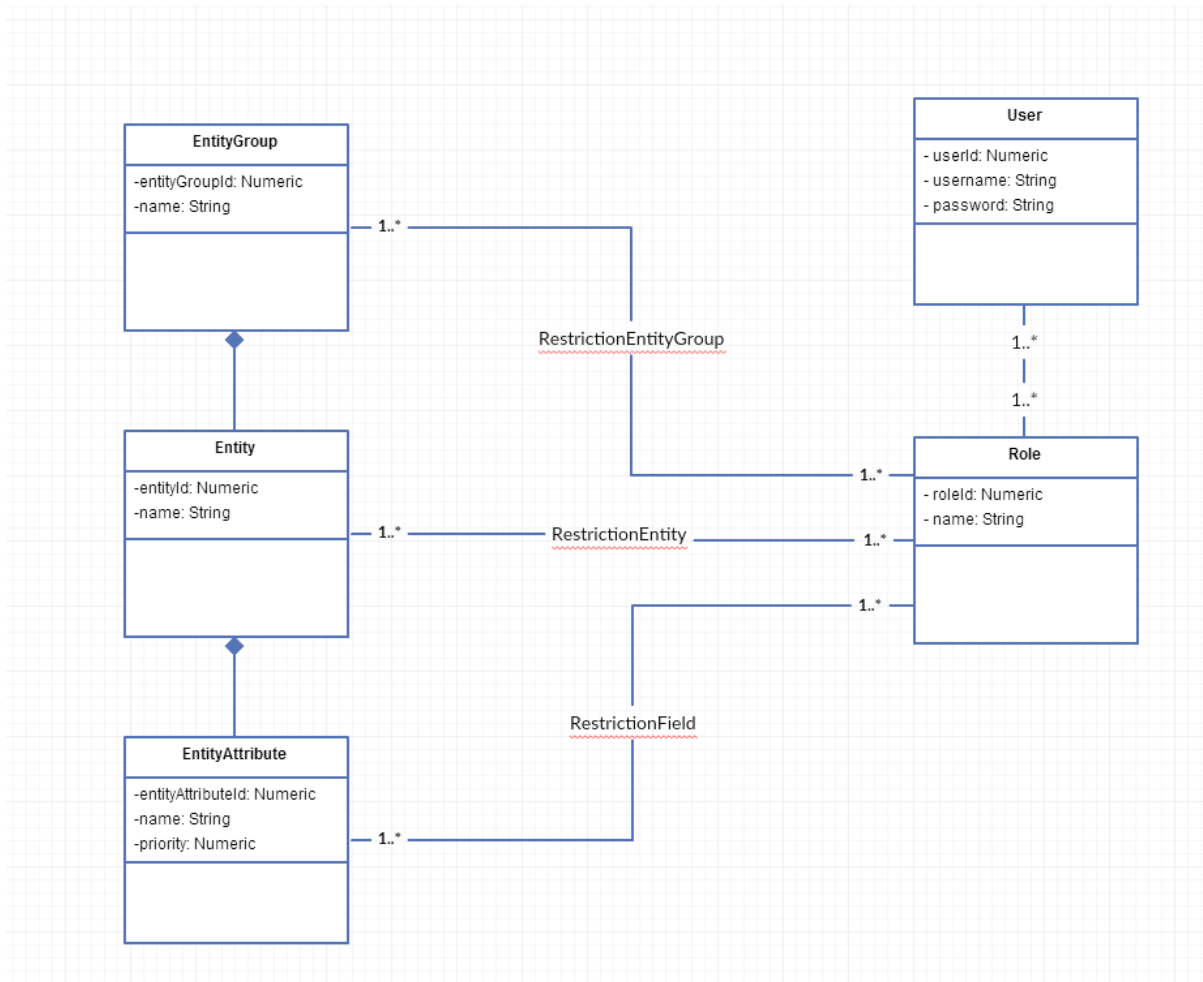
Figure 33 Metamodel's security package

Security can be defined at different levels and it is based on a multirole management system.

A user of the generated application is defined by at least one role.

Each role has three different list of access level

- restriction entity group

- restriction entity

- restriction field

A RestrictionEntityGroup define the rights of a certain role on a certain EntityGroup while RestrictionEntity links entity and roles while RestrictionField define the access rights on the Fields.

Each of these restriction contains four different privilege levels:

- search

- create

- update

- delete

Only the administrator can set these privileges.

4.8.   Logging

The system has his own logging systems to save each operation on database.

The process is based on the LogEntry classes that store information about the operation time and type, the ip address of the executor, the user, the entity and a custom detail field for a descriptive information.

By default all the standard operation are logged, like searches, insert, update, delete, login, logout, wrong password typing,..

4.9. Self generation

The designed metamodel can be interpreted as its own metamodel. This leads to the idea that the framework can generate a UI interface of itself.

The goal of this part of the project is to generate a UI over the meta-entities by using information stored in the meta-entities themselves.

By using this generated UI developers do not need to code Java domain classes. He can add and modify everything with the generated user interface that allows managing every aspect of the domain model like entities, fields, relationships, enumerative entities and fields, annotations...

As there is no need of specific programming language knowledge the framework may be used by not technical people that has just the idea of the model they want to generate.

Core classes containing the business logic of the framework make use of repository layer in order to apply the transformation.

5. Implementation experience

5.1. Starting point

The entry point of the framework was object of discussion because there were many possibilities.

A choice could be starting from a UML diagram, or from an existing database or from a set of Java classes.

Plugins and tools to convert database and UML to Java classes are really commons and easy-to-use, so the choice was on Java classes.

5.2. Model analysis

The procedure that transform a Java class into an Entity makes use of Java reflection. With Reflection is possible to analyze the content of a Java class at runtime, discovering fields, methods and annotations.

5.3. Focus on annotation

Java annotation are the crucial point of the configuration for this thesis. They are practical and easy to use, and they add value to the basic classes and variables.

The project uses a set of custom annotations to personalize the generated content.

5.3.1. Entity

- @Cache: marks the entity as a cached entity. The framework will use the second level cache of Hibernate for this entity.

- @DisableViewGeneration: the framework will not generate the view files for this entity and will be hidden from any other linked entity.

- @EnableRestrictionData: this annotation makes available a structure to add a level of security on the resource: data level. Some data may be restricted for certain users that have access to the resource.

- @MaxDescendantLevel: Indicates how many levels in the descendant tree the framework will consider.

- @SecurityType: indicates the SecurityType for the resource, the default is set in the properties

- @IgnoreMenu: the views files will be generated but the entity will not be included in the menu

- @IncludeMenu: the resource is forced to be listed in the menu

5.3.2. Field

- @Between: the generator will create a filter in the search form of the entity in order to get a minimum and a maximum value

- @DescriptionField: the field will be included in the description string of the resource, used to indicate the entity in the linked entities.

- @Embedded: mark the field as embedded, the generator will render its content as html

- @ExcelExport: the field will be included in the excel export

- @Filter: the field will be included in the search form of the parent entities

- @IgnoreSearch: the field will not be included in the search form

- @IgnoreTableList: the field will not be included in the table

- @IgnoreUpdate: the field will not be included in the detail form

- @Password: mark the field as a password in order to be encrypted when stored.

- @Priority: this annotation has a parameter that indicated the priority of the field in the form. Each group of entity attribute has a default value and the generator will create the form sorting the attributes by this value.

- @Tab: indicates the tab that will contain the entity attribute

- @IncludeSearch: field forced in the search form

- @IncludeTableList: field forced in the table list

- @IncludeUpdate: field forced in the detail form

5.3.3. Validation

- @Size: indicates the minimum and maximum value or length for the attribute

- @NotBlank: indicates that the field must be filled with a value

- @NotNull: indicates that the field must be set

5.4. Components

The algorithm consists in collecting all the entities of the project and generate the server side Java classes and the view's files.

5.4.1. Server

For each entity the framework will create different classes:

- entityRepository: Java interface based on Spring Data Framework to get data from database. Every fields of the entity has a correspondent method in this interface to search entities by that attribute. Among this, there is a generic search method that allow searching entities by means of all the different entityAttribute. It is possible to find entity whose children are like a certain entities. With the Filter annotation the method can search entities based on the children's attribute value.

- EntityService: Java interface to expose the methods of the service tier. The defaults methods consist in search, insert, update and delete.

- EntityServiceImpl: Java class that implements the correspondent interface. The update methods perform update on all the linked entities.

- EntityController: Java class that acts like a Spring REST controller. It offers different methods like searchById, find, insert, update, delete in the crud logic. There should be also the possibility to retrieve paginated results. Among these the mapping should be performed in order to pass

5.4.2. Client

The client's files are composed of both JS scripts and html files.

- src/app/components/entityName/entityName.controller.js

This is the AngularJS's controller script and contains all the function the user can recall from the user interface. Among the basic crud operations there are functions for downloading, initializing entities list and refreshing table grid.

- src/app/components/entityName/entityName.service.js

The service script is a wrapper for the communication with the server side software. It exposes the different crud operations calling the respective REST APIs.

- src/app/components/entityName/entityName.directive.js

AngularJS directive to expose the html template associated with the right controller.

- src/app/components/entityName/entityName-detail.html

This form contains all the fields and relationships. All the relationships are navigable and editable. It is managed with security authorities.

- src/app/components/entityName/entityName-search.html

It contains the search form with the attributes the user can use to search by entities. It also include the list of entities in table format. The list is paginated and the user can sort it, add and remove columns or export it in different formats.

- src/app/controller/entityName/entityName-template.html

This file contains represents content of the ng-view container. It is composed of the entity's directives contained in that entity's descendant tree.

5.5. Solution

The proposed idea has been implemented on top of Spring and AngularJS framework.

It allows annotating the entity's source Java classes in order to obtain the desired result. An algorithm analyze the source model and transform it following the metamodel of the framework presented in chapter 4.7.

This step can be avoided by means of the UI self-generated.

Once the model is defined the framework can generate an instance of it by creating the necessary files and Java classes proposed in chapter 4.5.

The generated software is a Single Page Application with multi role management.

Server side software is built on top of Spring framework following the REST architecture.

Client side application is built with AngularJS and perform REST call to the server in order to provide CRUD operations over the entities.

Login mechanism and security policies are generated among with the application, together with the logging system, API documentation and performance's analysis.

Generated software allows to scan an entire tree of descending entities and perform crud operations over them.

By means of specific annotation, it is possible to configure the generated UI, deciding which entities should be included in the menu and which entity's attribute should be displayed.

The same mechanism is provided for the security policy: developer can decide at configuration time the access level to each entity.

All these mechanism are provided in the two opposite logic as described in chapter 4.6.2.

Frontend generation has only be drafter: it shows key attributes for each entity and has support for pagination.

5.6.   Implementation details

5.6.1.   Date management

The data-type fields have been managed by using a standard Jackson serializer and deserializer. These components interact with the spring application when it needs to generate an output JSON. The field of the type Date are parsed by the Jackson serializer and read with the Jackson deserializer. By using these components, it was possible to treat all kind of date formats in the same way and the Angular application could read and send it.

5.6.2.   Enumerative management

The enumerative fields needed a custom management. The enumerative values are loaded on the application startup in order to be available from any AngularJS controllers and views.

They are represented by an html select that contains all the possible values.

5.6.3.   Menu generation

The implementation of this feature was the last one before the release of the 1.0 version of the framework.

The menu jsp was generated by using the Bootstrap navigation bar component: entity groups were the main item of the menu and, by using the dropdown, every entity group contained all the entities inside itself.

The menu generation is not changed over the different versions.

5.6.4. Export

It is possible to export the entity's list. The supported format are Excel, CSV and pdf.

It is possible to select the fields to export.

The referenced entities in the list are exported with their description fields.

This feature has been implemented by mean of the AlaSql library, a JS library that manage the export of a JSON list in different formats.

5.6.5. Easytree menu

Easytree is a JS library that allows creating sidebar menu with folders and items.

It saves the status in cookies so the state of the menu is granted.

The framework supports the generation with this second type of menu, the developer needs to configure the project with the Easytree option.

5.6.6. Tab

The detail form cannot contains too much information. To solve this problem the form has been organized using Bootstrap tabs. The fields can be annotated with a custom annotation @Tab that indicates the name of the tab that includes that field.

5.6.7. Between filter

This implementation lead to the automatic generation of between filter in the search form. By annotating a field with the @Between annotation its search form will include the start and end points for that field.

5.6.8. Filter

An entity can be searched by means of fields of the children entities.

Those fields need to be annotated with the framework annotation @Filter.

For example, an order entity can be searched by the name of the product of one of its order row.

5.6.9. Security

Security was the last feature implemented before the release of the 1.3 version of the framework.

The security structure involves a multi-role management system. Only the view files were affected

by security checks. The menu automatically adapted itself based on the user's role.

5.6.10. Validation

Fields can be annotated with framework specific annotation to grant some constraint about nullable

fields, not blank fields, size of the fields...

The implementation involved the AngularJS form validation and the bootstrap component to show

validation errors.

5.6.11. Advanced security management

Security management needed an upgrade and it was introduced the concept of Restriction between

roles and entities. The modify was propagated to server side components and in the view forms.

A user with a restricted access to a resource cannot call the resource's API and will not see any

reference to that resource in the views.

The concept of restriction has been extended to Field and EntityGroup, to provide a complete set

of security option.

Even the menu has been upgraded with respect to the new security management.

5.6.12. Webapp generation

This implementation is a consequence of the Spring Boot adoption. The framework is able to generate all the configuration classes needed by the server to be run. It includes Security configuration, Mvc configuration, Spring boot application, CORS filters, user service for the login.

5.6.13. File management

An entity can support File field: they can be uploaded or modified through the Multipart file in the http request.

The files are then stored in a directory of the server specified at configuration time by the developer.

The uploads are divided in a directory structure with the actual upload date in the ISO format.

In the view file-type fields are visible and it is possible to open the attached files.

5.6.14. Restriction Data

A new concept has been introduced to restrict the user access to certain data. Sometimes the restriction over a resource is not what the administration wants because he only needs to restrict the access to special data of that resource, for example administration data of the user resource.

Entities can be annotated with the @RestrictionData annotation and the structure between roles and data will be generated. It is a computationally expensive feature so by default it has been disabled.

5.6.15. Frontend generation

The frontend generation has been implemented as an alpha version. The main idea was to show a paginated list of entities in a more user-friendly logic.

To achieve this it has been implemented pagination on both client and server side.

In this section are displayed only the important fields of an entity, annotated by the developer. Embedded fields are represents as content instead of editable code.

5.6.16. Custom annotation: priority, max descendant level, security type

By developing this feature there were introduced some new annotations to manage crucial aspects of the framework.

Priority

By using this annotation the developer can control the order by which the entity's attribute are included in the forms. By default the framework will assign the following priorities.

- Primary key: 1

- Fields: 2

- Enumerative fields: 3

- Relationships: 4

MaxDescendantLevel

Sometimes it is not necessary to generate the complete descendant tree of an entity. With this annotation, the developer can specify the level of the tree he wants to reach for the selected entity.

SecurityType

This annotation allows setting up the security type over entities. The two security types supported are BLOCK_WITH_RESTRICTION and ACCESS_WITH_PERMISSION.

5.6.17. Embedded fields

Embedded fields implementation regards the management of embedded field type.

This fields are different from a normal text fields because in the view

5.6.18. Server statistics & metrics

In a complex application it is important to monitor the consumption of resource and the quality of service. This is the main reason why a new development has been done on the framework, by using the Dropwizard metric library.

By annotating the Spring controller it is possible to analyze their behavior, analyzing the average response time and the number of requests.

Another interesting thing is the monitoring of the cached resources.

5.6.19. Log system

In a real world crud application it is fundamental knowing when a resource has been created, removed or updated. A log system has been implemented in order to keep track on database about the changes in the dataset.

The log entry keep track about the timestamp of the operation, the user who performed it, the entity updated, the ip address and some information about the user agent.

5.6.20. Bower and Gulp

Framework is quite big, with many references and dependencies to Javascript libraries and respective style sheet files.

Bower has been introduced: it is a package manager for the web built on top of NodeJS.

It creates a repository of the Javascript libraries and CSS files and keep them up to date.

With this feature another utility tool has been introduced: Gulp, a Javascript task runner that automates developing operations and increase the productivity.

Gulp allows to automatically bundle and minify libraries and stylesheets, compile Saas, run unit tests and code analysis.

With Gulp, creating a build of the client application is a lot easier as it minifies the scripts and inject them into the main template.

At this point, the framework generated all the standard files that are part of the scaffolding and the bower files that contains all the required client dependencies.

5.6.21. UI-route navigation

The navigation of the generated application has been implemented with respect to UI-router, a module of AngularJS's UI.

UI-router has the concept of state and the user can navigate from one state to another. In this framework's case, each entity has its own state and all the action performed on a single entity and its descendant entities rely on a single state.

5.6.22. Login generation

With this implementation, the login page is generated together with the project. Each sector of the application is accessible through certain credentials: if the user is not authorized a login prompt will be shown.

5.6.23. $scope management

In the first releases of the framework Angular controllers and views communicated through the $scope variable which cannot hide any information of the $scope to the view.

That is why in the next release it has been decided to convert this logic to the "controller as" angular pattern. Using this pattern the view can only see the "controller as" and not the entire $scope.

With this development, the generated scripts are also compatible with Angular2, in a long-term horizon.

5.6.24. Swagger

Documentation is another key aspect in the real world environment: having a complete documentation is helpful but requires a lot of time.

This development brought to the complete generation of the documentation files, together with all the other entity's files.

This was possible with the Swagger library: it exposes a set of annotation for the main controller's method and automatically build a set of documentation for the API.

The API's can be tested through a proper interface.

5.6.25. Continuous generation

This was the last feature implemented to make possible generating just a part of the goal project without re-generating everything.

All the main entities in the metamodel contain a field that denotes the timestamp relative to the last update.

The second goal of this development was to maintain the custom code written by the developer since the last generation.

To achieve this the framework is totally integrated with Git.

A new GenerationRun open a new branch where the framework commits the modified or new files.

After that the new branch is compared with all the commits since the last generation and it is merged with them. In case of hard modifies by the developers he will need to manually merge the files.

6. Experiment and discussion

The implementation has been tested on two real world examples of management software.

The first one relies on a PostgreSQL well-formed database composed of 150 tables and the second one manages a MySQL database with 20 tables with some exceptions of the normal rules.

6.1. Additional implementation

During the testing phase the software has been evolved to supply some features that really help the developer who wants to use the framework. As the configuration time may be long, few more mechanisms has been developed.

6.1.1. Menu

The menu can be generated in two ways, depending on a property set on the project. If the generation type is "SHOW_INCLUDE", only the entities annotated with the @IncludeMenu annotation will be rendered into the menu directive. In the other case, with the generation type set to "HIDE_IGNORE" all the entities will be by default included in the menu, excluding those annotated with @IgnoreMenu.

6.1.2. Field visibility

It has been improved the mechanism of configuration of the field visibility, putting it at three different levels: search, table list and update. All these three parameters depends on the generation type of the entity if set (or on the project one...).

An entityAttribute annotated to be included in the search form or in the table list is automatically rendered in the detail form.

6.1.3. Auto annotation

The main idea is that the developer must annotate just the entities that are included in the menu, and these are not so many usually. The procedure annotates the linked entities (a linked entity is an entity linked by a relationship coming from a menu-level entity) to automatically show by default all the fields.

If the linked entity is already annotated the procedure maintains the state.

6.1.4. Tab for each relationship

If the right flag on the project is set, the algorithm will generate a tab for every relationships of the entity, calling it with the name of the relationship.

Relationships can be numerous and having each of them in a separate tab can be useful, without having to annotate all the relationships of every entity in the project

6.2.   Project setup

The generated application is based on Maven and Yeoman Generator.

Maven Project

A new Maven project in the eclipse IDE must be created, defining the name, artifact and the groupId.

Yeoman Generator

Yeoman generator is a scaffold generator that generates a standard structure for the project that has been followed by the thesis.

It is sufficient to run the command yo-angular generator in the console.

Model annotation

The developer can copy his set of model classes in the Maven Project and add all the annotations that can be useful for the generation.

After the annotation job, the developer must set his custom properties in the application.properties file of the project, defining the project name, the model package, the generated package name, the datasource connection…

Learn metadata

The developer will run the metadata learning procedure that will analyze the model classes. The framework will also perform a check on the model and present errors to the developer, if some of the classes does not pass the validation process.

Generation

The developer will run the generation procedure, starting from the learnt metadata, and the framework will generate the complete project following the developer properties and metadata.

Customization

Now the developer has a set of basic classes, AngularJS's scripts and HTML5 files that can easily modify.

Continuous generation

The developer can decide to regenerate the project after the first generation, to take care of a new part of the model or some changement.

To obtain this feature the project must be under a Git version control system.

The framework will consider all the custom modifies performed by the users and it will merge them with the new generated code.

Like every merge it will be as painful as consistent the developer modified the files.

6.3. Two real cases

6.3.1. EBSN-Backoffice

EBSN-Backoffice is a management software for an ecommerce solution.

It allows the users to manage information about customers, products, orders, order detail, warehouses, promos...

Most of the actions performed by this software are coherent with the crud logic.

The software is based on a PostgreSQL database of 150 tables.

The preparation for the automatic generation of this software took 3-4 hours.

The timetable is clear in the commit of the Git repository.

Figure 34 Repository used for EBSN-Backoffice generation

Notes about the generation

The first half hour of the work has been dedicated to the cleaning of the model classes, in order to add the @Id annotation for the primary key fields as they were annotated on the getter method and to remove some useless import.

Other two hours have been dedicated to the deletion of the business logic inside the model class and to replace the normal JPA "Entity" annotation with the project custom "GenerateEntity" annotation. The composed primary keys have been deleted and substituted by a numeric key because the framework does not support embedded keys.

At the end of this phase the model classes did not contain any errors: the automatic generation could run.

The check performed on the metamodel was wrong due to some convection adopted by the framework and not respected by the domain classes: in a few minutes they have been identified and fixed.

The generation of the basic project was successful. After two more hours of work, the project has been enriched by many custom annotations in order to make the generated packages more usable and more human.

Figure 35 Screenshot from the generated application

6.3.2. EBSN-Storepicking

This software has been developed to manage the orders in a warehouse and to pick up the products for the customers like a WMS (Warehouse Management System).

It has been coded with the Spring framework using JPA and it is built on top of a MySQL database composed of 25 tables.

The first generation was completed in less than 2 hours as the project was already built on the top of Spring world frameworks.
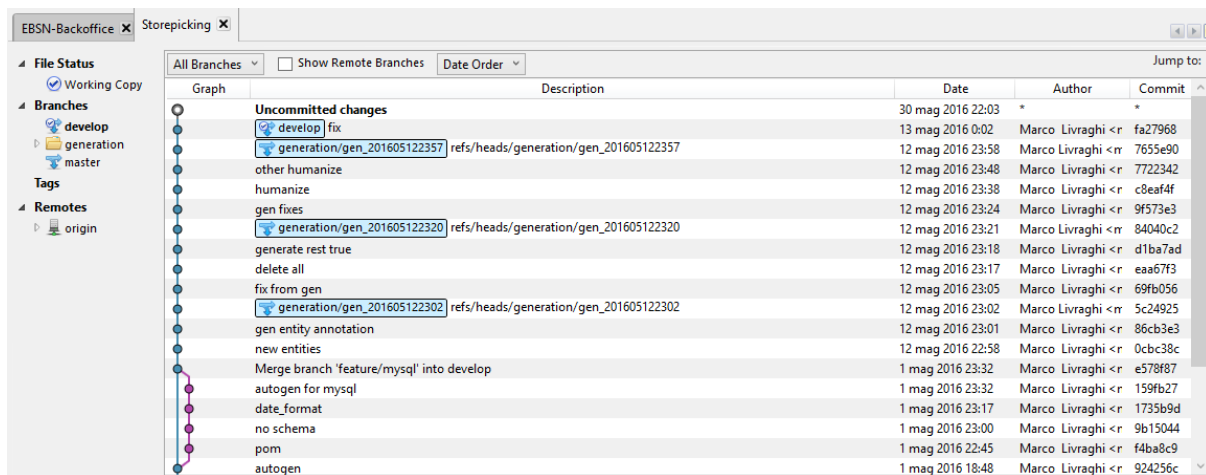

Figure 36 Git repository used for EBSN-Storepicking generation

The generated software has been annotated in order to make it more human and readable: these operations took one more hour.

Figure 37 Screenshot from the generated EBSN-Storepicking application

6.4.   Result

The test on two real software used in production since years has been helpful and motivating for the developing of the framework. This part lead to new implementations that have been done.

6.4.1.   Relationship name and type

The framework originally accepted only java.util.List for the relationships and the field names should follow a certain convention.

By testing the framework on the two real systems this bond became immediately impossible to maintain.

To achieve the goal new features have been introduced. The first one was quite easy and made the framework able to keep in the metamodel even java.util.Set.

The second one was trickier but still successful: after the implementation the framework can accept every field name for the relationships.

In case of multiple relationships between two entities, the system can fail, but this is rare and very easy to fix: it is sufficient to change a few getter method's names.

6.4.2.   Manage of MySQL

MySQL adopts a different dialect of the Hibernate Query Language. The management of the Date fields was implemented using a PostgreSQL-dialect function that on MySQL-dialect systems has a different name.

A new parameter was introduced in the application and the developer is able to generate a software over a PostgreSQL db or over a MySQL db.

The adjustments also include the schema definition for the tables, as MySQL does not support different schema over a single database.

6.4.3. Result discussion

The time dedicated for the projects generation was not large: in a few hours the projects were ready to be built.

An important factor for the generation is the original model: a model already annotated with a recent version of JPA is much easier than a project to be generated from nothing.

The work done for the configuration and the setting is not critical and can be executed from every people that knows the desired output and has a very low knowledge of programming language.

7. Conclusions

The fulfilled goal of the thesis was the building of a framework that would allow the automatic generation of crud interfaces, by meaning of simply reusable code and almost compatible with the most common software solution.

Implementation of the system covered a full-stack solution, with a Spring server side and an AngularJS client. The two modules are separated and they can be interfaced with different and already existing systems.

The code generation starts from a set of Java classes that represent the domain model. This classes may be annotated with framework specific annotations to add some informative value for the generation in order to render a more usable and human software.

The original model can be modified after a generation and the following automatic generation would modify only the parts that have been changed by the developer.

The framework generates a complete server side module with respect to repository classes, service classes and controller classes.

The repository classes are based on the Spring Data framework.

Among these, the framework generate a complete documentation for the REST APIs, providing an interface to test all the methods.

The client module follows the structure adopted by the Yeoman generator and provides a complete set of AngularJS services and controllers.

The generated code is secured by a multirole security system with password encrypted with the BCrypt algorithm.

Once the implementation has been finished, the framework was tested by applying it on two real systems, a management software for an ecommerce website and a warehouse administration system.

The first one was composed of 150 tables over a PostgreSQL database while the second one was based on a MySQL db with 20 tables.

7.1. Discussion of result

The results have been encouraging as the framework well adapted itself to the real world examples.

The proposed solution leads to a drastic reduction of the development time: the basic crud software can be generated in hours instead of months, reducing the development and designing cost for an application.

On the other hand, generated UI is standard, following the Bootstrap style, and requires time as the CSS style needs to be adapted to the desired aspect.

The limits of the project consist in the constraints about the model, like the embedded primary keys or the name itself of the primary key fields.

These constraints are not insuperable in the principle of the convention over configuration, but a real system may need to maintain its special structure.

7.2.   Possible future work

The proposed implementation can be enlarged following the idea illustrated.

The first step should be to remove the existing constraint about the starting model.

The realized framework provides a basic version of the frontend views, where the entities are not displayed in the crud logic but in a user-friendly version.

This mode ensure pagination and shows only the important field of an entity but the way can be explored much deeper.

A possible integration for the implemented framework would be Flyway, to manage modifies in the database.

Until now the Java Project has always been run with the JPA update mode on. This means that a modify in the Java domain was reflected on the database when the application starts, but this strategy can cause problems in a real world environment as it is dangerous to enable database modifications at runtime.

The flyways script files can be generated together with the entities in order to create the proper database model.

A concrete improvement to the framework would be a complete integration with Spring Roo. To achieve this result a new mode should be implementing as Spring Roo uses to pass parameters in the query string and not as a JSON object. This is not a critical development and could be integrated in order to provide a fresh AngularJS client option for Roo applications.

8. Bibliography

1. Google. https://www.consumerbarometer.com/en/. *ConsumerBarometer.* [Online] 2015.
2. Chang, Chih-Hung, et al. *An Experience of Applying Pattern-based Software Framework to Improve the Quality of Software Development: 4. The Design and Implementation of OS2F.* 2008.
3. Biermann, A. *Automatic Programming. In Encyclopedia of Artificial Intelligence.* 1992.
4. J. Malenfant, M. Jacques and F.-N. Demers D. *A Tutorial on Behavioral Reflection and its Implementation.*
5. Johnson, Rod, et al. *Professional Java Development with the Spring Framework.* 2005.
6. Sonatype Company. O'Reilly Media, Inc. *Maven: The Definitive Guide.* 2009.
7. FRANCA GARZOTTO, PAOLO PAOLINI, DANIEL SCHWABE. *HDM — A Model-Based Approach to Hypertext Application Design.*
8. Paolini, Piero Fraternali and Paolo. *A Conceptual Model and a Tool Environment for Developing More Scalable, Dynamic, and Customizable Web Applications.*
9. *RMM: A Methodology for Structured Hypermedia Design.* 1995.
10. Gustavo Rossi, Daniel Schwabe,Fernando Lyardet. *Web Application Models are more than Conceptual.*
11. Paolo Atzeni, Giansalvatore Mecca, Paolo Merialdo. *Design and maintenance of data-intensive web sites.* 2006.
12. Mary Fernandez, Daniela Florescu, Jaewoo Kang, Alon Levy, Dan Suciu. *Overview of Strudel - A Web-Site Management System.* 1996.
13. Houben, Richard Vdovjak and Geert-Jan. *A Model-driven Approach for Designing Distributed Web Information Systems.*
14. Andreas Kraus, Alexander Knapp, and Nora Koch. *Model-Driven Generation of Web Applications in UWE.*
15. Paloma Cáceres, Esperanza Marcos, Belén Vela. *A MDA-Based Approach for Web Information System Development.*
16. Santiago Meliá, Jaime Gómez, José Luís Serrano. *WebTE: MDA Transformation Engine for Web.*
17. Gwyer, M. *Oracle Designer/2000 WebServer Generator Technical Overview .* 1996.
18. Santiago Melià, Jaime Gomez. *The webSA approach: applying model driven engineering to web applications.* 2006.
19. *Web Modeling Language (WebML): a modeling language for designing Web sites.* Stefano Ceri, Piero Fraternali, Aldo Bongio. 2000.
20. *Model-Driven Development of Web Applications: the Autoweb System.* Piero Fraternali, Paolo Paolini.
21. Marco Brambilla, Piero Fraternali. *Model-Driven UI Engineering of Web and Mobile Apps with IFML.* s.l. : Morgan Kaufmann Publishers, 2015.
22. http://www.webratio.com/. *WebRatio.* [Online]
23. http://ifml.github.io/. *IFML Editor.* [Online]
24. https://www.mendix.com/. *Mendix.* [Online]

25. https://www.outsystems.com/. *Outsystems.* [Online]

26. http://www.orangescape.com/. *OrangeScape.* [Online]

27. http://www.softwareag.com/special/longjump/. *Longjump.* [Online]

28. http://www.tersus.com/. *Tersus.* [Online]

29. http://www.softfluent.com/. *Softfluent.* [Online]

30. http://metawidget.org/. *Metawidget.* [Online]

31. Philip J. Hayes, Pedro A. Szekely, and Richard A. Lerner. *Design Alternatives for User Interface Management Systems Based on Experience with Cousin.* 1985.

32. Andrew J. Schulert, George T. Rogers, James A. Hamilton. *ADM, A dialog Manager.* 1985.

33. Olsen, Dan R. *MIKE: the menu interaction kontrol environment.* 1986.

34. Brad Vander Zanden, Brad A. Myers. *Automatic, Look-and-Feel Independent Dialog Creation for Graphical User Interfaces.* 1990.

35. Piyawadee Sukaviriya, James D. Foley, Todd Griffith. *A Second Generation User Interface Design Environment: The Model and The Runtime Architecture.* 1993.

36. Charles Wiecha, William Bennett, Stephen Boies, John Gould, Sharon Greene. *ITS: A Tool for Rapidly Developing Interactive Applications.* 1990.

37. Pedro Szekely, Ping Luo , Robert Neches. *Beyond Interface Builders: Model-Based Interface Tools.* 1993.

38. Strahinja Lazetic, Dusan Savic, Sinisa Vlajic, Sasa Lazarevic. *A Generator of MVC-based Web Applications.* 2012.

39. Samir MBARKI, Mohammed ERRAMDANI. *Toward automatic generation of mvc2 web applications.*

40. EL BEGGAR Omar, BOUSETTA Brahim,GADI Taoufiq. *Automatic code generation by model transformation from sequence diagram of system's internal behavior.*

41. Sammet, J.E. *Programming Languages: History and Fundamentals (Automatic Computation).* 1969.

42. L. I. Lieberman, M. A. Wesley. *AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly.*

43. Goad, Chris. *Special purpose automatic programming for 3D model-based vision.* 1987.

44. Frederick Murray Burg, Joseph DeSimone. *Web-based generation of telephony-based interactive voice response applications. US6456699 B1* 11 30, 1998.

45. John R. Koza, James P. Rice. *Automatic Programming of Robots using Genetic Programming.*

46. DeveloperWorks, IBM. *The Principle of Least Astonishment.* 2001.

47. Stefano Ceri, Piero Fraternali, Stefano Paraboschi. *Design Principles for Data-Intensive Web Sites.*

9. Table of figures

10. Acknowledgements

I would like to thank my family for the support in these five years, which have been as hard as interesting and rich of great moments.

I would like to thank my supervisor, Marco Brambilla, for his guidance in the last months with his experience in this field.

I would like to thank all my friends and fellow students for the great time together, in particular those close friends who listened to my ideas while I was fully focused on this work.

Last but not least, I would like to thank Digitelematica, Federico Dell'Acqua, Norberto Viganò and every colleague, for the excellent work together and the great advices you gave to me in the last two years.