UNED | UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

CAPICUAGEN: GENERADOR DE CÓDIGO BASADO EN CARACTERÍSTICAS

Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos

Itinerario Ingeniería de Software

Trabajo final. Código de la asignatura: **31105128**

Realizado por
José Luis Bautista Martín

Director: **Rubén Heradio Gil**

Curso 2015 – 2016, Convocatoria de septiembre

Esta hoja está en blanco para facilitar la impresión de este documento a doble página.

Este documento comprende el trabajo final del **"Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos"**, itinerario **"Ingeniería de software"**, código de la asignatura: **31105128**.

Ha sido realizado por **José Luis Bautista Martín** (joseluisbautista@gmail.com), estudiante de la UNED cuyo usuario es: jbautista62, entre los meses de enero y julio de 2016, correspondiente al ciclo escolar 2015-2016, para presentarse en la convocatoria de septiembre.

El título del trabajo es "**CapicuaGen: Generador de código basado en características"**. La opción elegida es de **Tipo B** (trabajo específico propuesto por el alumno), basado principalmente en la asignatura de **"Desarrollo de familias de productos de software desde un enfoque generativo",** además de los contenidos tratados en las asignaturas de **"Generación automática de código"** y **"Arquitecturas para sistemas software"**, siendo el director de dicho trabajo el profesor **Rubén Heradio Gil.**

Esta hoja esta en blanco para facilitar la impresión de este documento a doble página.

Espacio reservado para la hoja de calificaciones.

Esta hoja está en blanco para facilitar la impresión de este documento a doble página.

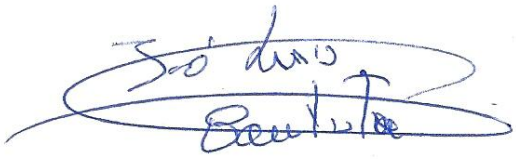
Impreso TFdM05_Autor. Autorización de publicación

y difusión del TFdM para fines académicos

Autorización

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los Autor/es



Juan del Rosal, 16
28040, Madrid

Tel: 91 398 89 10
Fax: 91 398 89 09

www.issi.uned.es

Esta hoja está en blanco para facilitar la impresión de este documento a doble página.

Agradecimientos

Quiero agradecer a mi esposa Nancy y a mis hijos Rubén y Ana, la paciencia que me han tenido en la realización de este trabajo. Igualmente quiero agradecer a mi padre, José Luis Bautista, la ayuda y el apoyo que siempre me ha brindado.

RESUMEN

En este documento se expondrá el sistema CapicuaGen, el cual es un generador automático de código empresarial basado en características creado por el autor.

CapicuaGen genera el código fuente de los diferentes aspectos de un sistema software, delegando dicha funcionalidad a los llamados **"generadores de características"**. Los generadores de características son independientes entre sí y una vez creados podrán ser utilizados por múltiples programadores a través de un repositorio centralizado al que es posible subir dichos generadores o bajar los realizados por otras personas, de forma parecida a los repositorios de gemas del lenguaje de programación Ruby.

Con la utilización de CapicuaGen se espera conseguir los siguientes objetivos:

- Agilizar la construcción de un sistema a partir del uso de la generación automática que creará de manera adecuada las características comunes de cada sistema, ensamblándolas apropiadamente.
- Reducir la programación manual de un sistema a los elementos particulares que así lo requieran, esto es, a aquellos que son exclusivos de dicho sistema y por lo tanto no están incluidos en otros.

En la fecha actual CapicuaGen es un producto desarrollado de forma personal, abarcando una colección de generadores de índole casi didáctica, pero al estar constituido como código libre, diseñado para ser ampliamente escalable y poder integrase dentro del ciclo de trabajo de un desarrollador, es de suponer que pueda ampliarse en futuro próximo con nuevas e interesantes funcionalidades.

En esta versión, podemos ver procesos de generación funcionales para sistemas de escritorio de Windows (con un set de pantallas comunes para este tipo de software), sistemas para exponer funcionalidad a través de servicios web (RESTful) y aplicaciones móviles para Android, con consumo de funcionalidad a través de servicios por HTTP.

LISTA DE PALABRAS CLAVE

.Net, Android, Arquitectura de software, C#, Capa de acceso a datos, Capa de negocio, CapicuaGen, Característica, Catálogo, Gema, Generación automática de código, Generador, Git, GitHub, GUI, Repositorio, Ruby, RubyGems, Servicio RESTful, SQL, Template, Web.

CONTENIDO

LISTAS DE FIGURAS Y DE TABLAS

Índice de ilustraciones

Índice de tablas

Índice de resultados mostrados por pantalla

Índice de ejemplos

Capítulo I. **INTRODUCCIÓN**

1.1 Contexto

CapicuaGen se ubica dentro de los tipos de herramientas conocidas como **"Generadores de código"**, siendo su especialidad la generación de código con enfoques empresariales.

Los "Generadores de código" son sistemas (o partes de sistemas) cuya funcionalidad, como dice su mismo nombre, es crear el código fuente de un sistema destino a través de una configuración o especificación de una necesidad establecida previamente.

El cómo se especifica dicha necesidad puede variar entre distintos enfoques para cada generador en particular, pudiendo ser mediante un diseño en UML, un lenguaje específico del domino (DSL) o cualquier otro mecanismo, pero teniendo todos ellos en común que son más afines a características y elementos de negocio que a temas de índole técnicos en concreto. Esto es porque intentan llevar la abstracción de la construcción de sistema a un nivel más elevado que la programación directa en un lenguaje en particular.

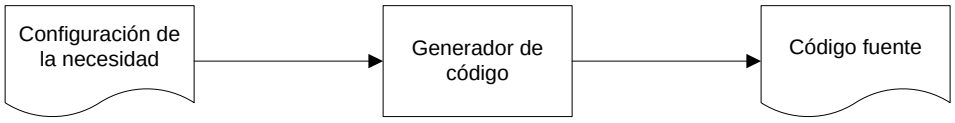


Ilustración 1. Ejemplo de generador de código.

CapicuaGen nace dentro de un contexto empresarial basándose en dos principios, por los que se considera que es beneficioso el uso de generadores de código.

- **El software va a cambiar:** Los cambios son necesarios para el negocio, además se dan de forma cada vez más frecuente, lo cual tiene un alto impacto en las empresas como mencionan Banker y Slaughter [1]. Debemos poder responder a la necesidad de dicho

cambio para que nuestra empresa siga vigente en el mercado de la forma adecuada. Los generadores de código nos pueden ayudar a gestionar dichos cambios de una manera más sencilla y rápida.

- **El software se parece entre sí:** Casi todo el software empresarial se parece entre sí. La mayoría de las compañías gestionan recursos de algún tipo (humano, económico, etc.) y generan un producto (dinero, servicios, o bienes de alguna naturaleza). Técnicamente hablando, casi todos los sistemas tienen características como acceso a datos, seguridad, interacción con el usuario o trazabilidad. Los generadores de código nos ayudarán a crear apropiadamente todos estos elementos comunes, disminuyendo el tiempo y los recursos dedicados a tal efecto. Este concepto es explicado por Velthuis y Garzás [2] en su planteamiento de fabrica de software.

1.2 Acerca de CapicuaGen

CapicuaGen es un proyecto de código abierto, bajo licencia LGPL, que puede ser usado tanto para crear software comercial, como software libre.

Es un proyecto modular, basado en un generador central, que delega en elementos más pequeños conocidos como **"Generadores de características"**, cuya misión es generar un aspecto en particular de un software destino, como puede ser la interfaz gráfica, la capa de acceso a datos, o la exposición de servicios y funcionalidades a través de una red corporativa o pública.

Los generadores de características, como menciona Czarnecki [3], son intercambiables e independientes, pudiendo elegir los que sean más convenientes para un escenario en particular, y cambiándolos cuando sea necesario. Igualmente pueden organizarse en repositorios públicos o privados (empresariales) y ser de fácil acceso para su uso, implementación y ampliación.

El uso del generador y la contracción de los repositorios se sustentan sobre lenguajes y tecnologías ampliamente adoptados por la industria, como Git, Ruby o repositorios públicos

como GitHub, o RubyGems, con lo que su curva de aprendizaje es corta y sus capacidades de distribución y ampliación son altas.

El funcionamiento general de CapicuaGen es representado por el siguiente diagrama:

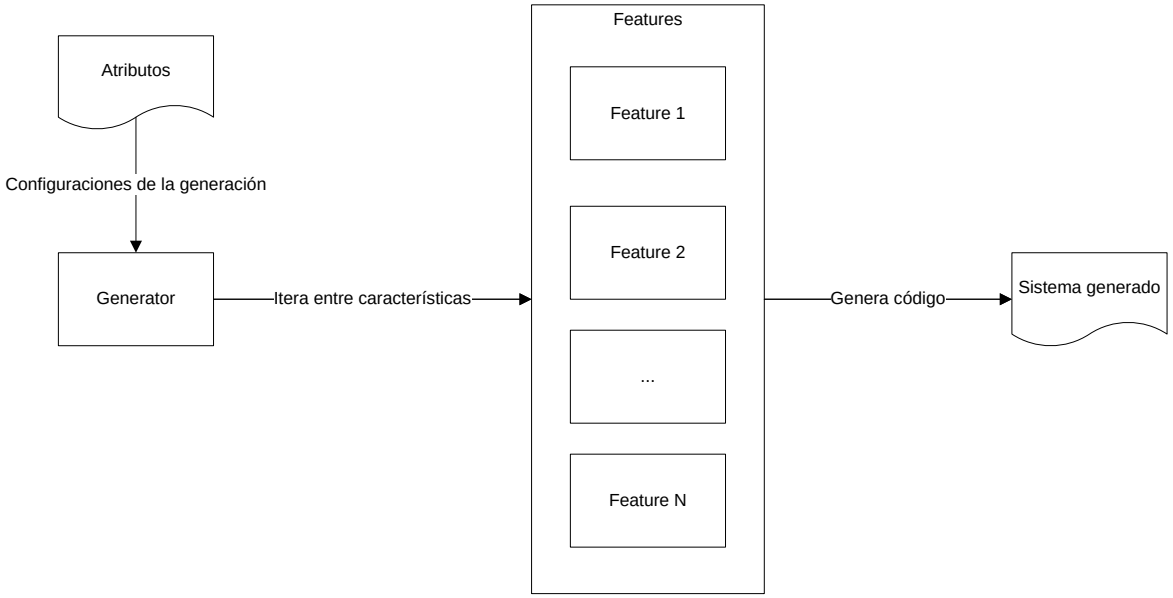


Ilustración 2. Diagrama de funcionamiento de CapicuaGen.

CapicuaGen es, a la fecha, un proyecto personal construido durante un periodo de seis meses, que contiene una serie de generadores de características sencillas expuestas a nivel didáctico, pero plenamente funcionales. Debido a su naturaleza de índole abierta, y su escalabilidad, es de desear que pronto pueda ampliarse con nuevas e interesantes características, aumentando cada vez más su potencia y utilidad.

Las capacidades generadoras que posee en la actualidad son las siguientes:

- Generación de proyectos para Visual Studio 2015.
    - Creación de proyectos de escritorio.
    - Creación de proyectos para exposición de servicios Web.
- Generación de proyectos para Android Studio 2.0.

No hay ninguna restricción en cuanto a la tecnología o lenguajes para los que es posible generar código fuente, siempre que exista un generador adecuado para tal efecto. En los ejemplos anteriores generamos elementos para Windows, Web, Android, y en C#, Java o XML.

Igualmente y tal como se verá en la descripción específica de la solución desarrollada, CapicuaGen está pensado para ser parte del ciclo de trabajo del desarrollo de un sistema, permitiendo ampliar su funcionalidad y reciclándola para nuevos proyectos según se muestra en el siguiente diagrama:

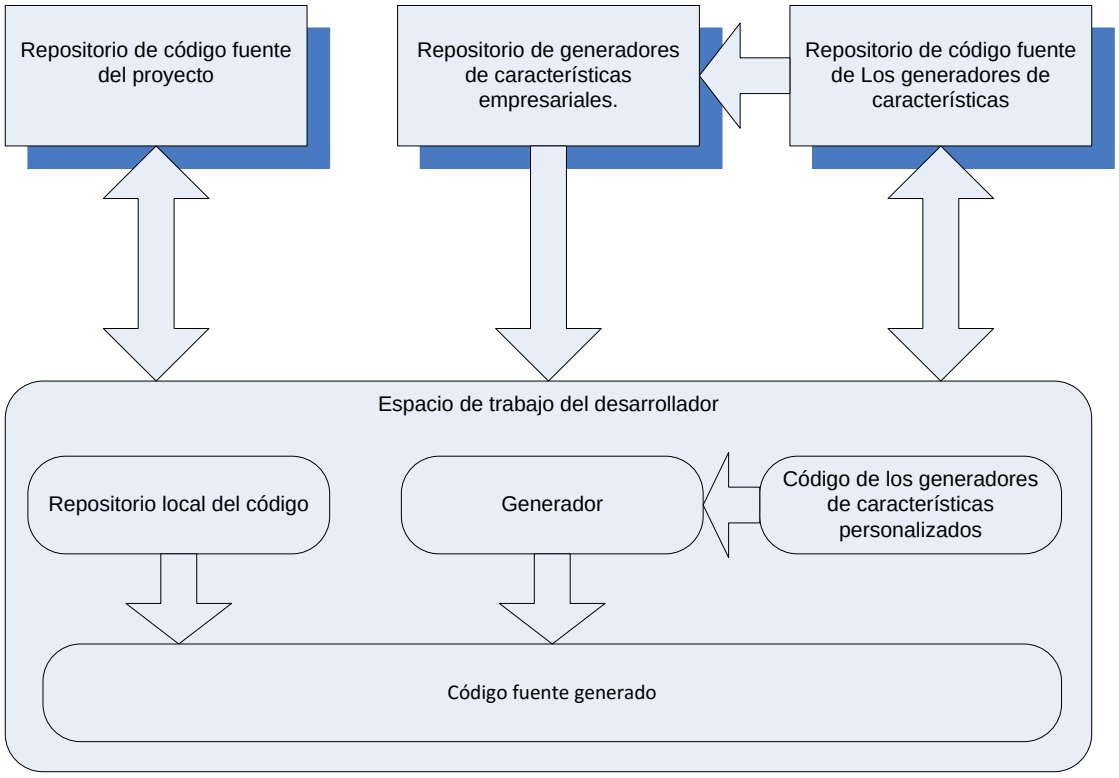


Ilustración 3. Diagrama de flujo de trabajo de CapicuaGen.

1.3 Objetivos de CapicuaGen

Los objetivos a conseguir son los siguientes:

- Agilizar la construcción de un sistema a partir del uso de generadores de código que creen de manera adecuada las características comunes ensamblándolas apropiadamente.

- Reducir la programación manual de un sistema a los elementos particulares que así lo requieran, esto es, a aquellas partes que son exclusivas de dicho sistema y por lo tanto no están incluidas en otros.

Con el cumplimiento de estos dos objetivos se espera satisfacer las siguientes necesidades:

- Aumento de la calidad del software al generarse en parte de forma automática. Los fallos detectados se podrían corregir en los elementos generadores y aplicar, por ende, al código de los sistemas.

- Los codificadores no perderían tiempo realizando tareas repetitivas y se concentrarían en las que realmente dan valor al sistema y no en aquellas que se duplican desarrollo tras desarrollo.

- Reducción de tiempo de desarrollo, con base en la generación de código automático.

Capítulo II. **ESTADO DEL ARTE**

El contexto en el cual se da la creación de CapicuaGen, es el de la generación automática de código, dentro de las necesidades empresariales para la creación de sistemas.

Si analizamos la evolución entre los diferentes lenguajes y tecnologías presentes en la actualidad para la construcción de software, podremos observar los siguientes aspectos:

- Lenguajes de programación cada vez más abstractos y enfocados a la manera de pensar de los programadores y no a la manera de operar de las máquinas.[1]
- Sistemas que tienden a la reutilización del software, creándose a través del ensamblado de componentes previamente desarrollados, situación mencionada por Malveau y Mowbray [4].
- Sistemas que delegan en componentes y servicios de diversas arquitecturas, de los que no se tiene conocimiento de cómo realizan su tarea, sino de la funcionalidad que realizan (se sabe el **"qué"** no él **"cómo"**), tal como afirman Garbajosa y Soriano [5].

Dentro de este enfoque se diseñó CapicuaGen, con la intención de poder reutilizar la mayor parte posible de los sistemas que construimos, incluidas las fases de "análisis" y "diseño", pero generando un código diferente para cada necesidad a suplir. Con base en eso integraremos CapicuaGen dentro de una metodología de **"fábrica"**, con una **"línea de producción de software"**.

[1] Podemos ver esta tendencia reflejada en los lenguajes punteros de la actualidad, como Ruby, Java o C# en comparación con otros como ensamblador o C. Un análisis de esta tendencia se puede apreciar en el libro de Pratt y Zelkowitz "Lenguajes de programación, Diseño e implementación" [18].

Para construir correctamente nuestra fábrica de software, debemos poseer una serie de elementos que es necesario combinar adecuadamente, para conseguir un sistema estable y escalable a futuro.

Dichos elementos son:

- Metodologías adecuadas.
- Patrones de software.
- Consumo de framework de terceros.
- Desarrollo de framework empresariales propios.
- Control de versiones.
- Generador de código.

Pueden verse modelos semejantes en los analizados por Velthius y Garzás [2] y por Greenfield y Short [6]. Igualmente pero desde un enfoque más técnico, exponen esta situación Richardson, Avondolio y otros en su libro "Profesional Java JDK 6" [7].

2.1 Arquitectura para un sistema empresarial prototipo

Con respecto a los sistemas empresariales, las arquitecturas pueden ser muy diversas y es necesario saber usar la idónea en cada momento (como explican Garlan y Shaw [8]), sin embargo el siguiente diagrama de capas representa un prototipo general común en este tipo de sistemas[2]. Se debe analizar cada capa, y ubicar qué elementos pueden ser creados automáticamente por nuestro generador.[3]

[2] El modelo está fuertemente basado en el propuesto por Microsoft en su guía de arquitectura [20].

3 Otra fuente de inspiración para la división en aspectos y características de la arquitectura presentada es el software "Microsoft Enterprise Library" descripto en el libro "Developer's Guide to Microsoft Enterprise Library, 2nd Edition" [22] y de forma más directa en su página web [23].

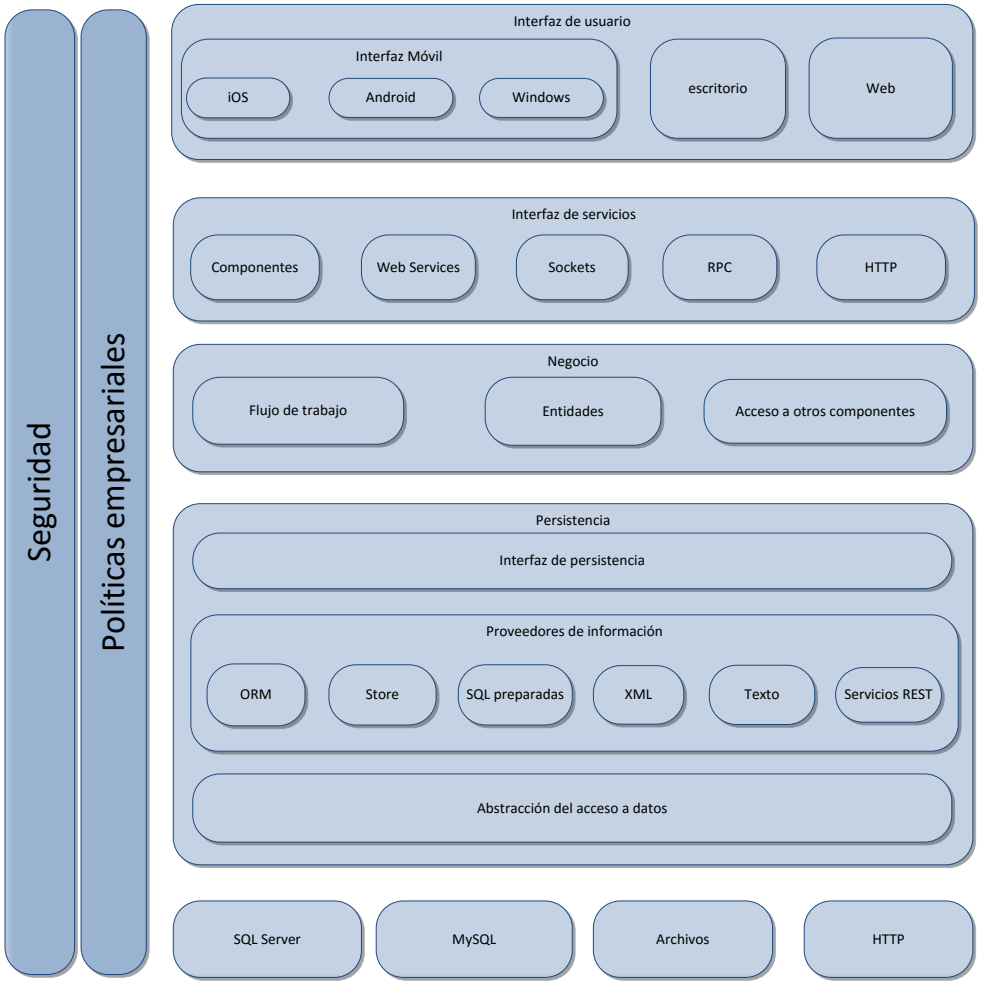


Ilustración 4. Arquitectura empresarial prototipo.

Interfaz de usuario

En esta capa se define la apariencia y la interacción con nuestros usuarios. Debe ser lo suficientemente sencilla para que la usen con facilidad, y lo suficientemente completa para que sea útil.

La interfaz representa lo que puede "hacer y ver" nuestro usuario, y esto tendría que ser común para todas las plataformas, el generador debiera ser capaz de crear dicha funcionalidad, y una vez que cambie el negocio, rehacer el código automáticamente para que se adapten todas

las interfaces a dichos cambios. Sería necesario contar con una característica generadora para cada plataforma deseada.

Interfaz de servicios

Representa la puerta de acceso a la funcionalidad del negocio. Es una fachada que expone a dicha funcionalidad de forma que se pueda consumirse apropiadamente por la interfaz gráfica. De esta manera aislamos el negocio, de la forma en la que este es expuesto.

Debido a que es un mismo negocio el que se debe exponer y que sólo cambia la forma de presentarse, el generador puede crear automáticamente cada fachada necesaria con base en dicho negocio.

Negocio

En esta capa está el problema que queremos resolver en nuestro sistema. Representa, en forma de software, la solución a una necesidad a suplir por la empresa con respecto al mercado.

Esta parte es muy dependiente de cómo diseñemos el modelo del dominio de nuestra aplicación, acorde a esto podemos crear elementos como:

- Entidades de negocio basadas en otros elementos como UML, o una base de datos.
- Agregar funcionalidad con base en diversas configuraciones.
- Agregar funcionalidad y comportamiento común de la empresa.

Persistencia

Esta capa se encarga de gestionar el acceso a diversas fuentes de persistencia desde dónde obtener y guardar la información de nuestro sistema.

Es posible crear las interfaces de los servicios de persistencia siguiendo algún patrón de software o alguna política empresarial sobre cómo deben persistir las entidades de negocio.

Seguridad

Es una capa que afecta a todas las demás independientemente de lo profundas o externas que sean[4].

El generador puede crear distintos tipos de seguridad y agregarlos en los puntos requeridos ya sea por programación orientada a aspectos, o cualquier otro mecanismo.

Políticas empresariales

Implica todo aspecto empresarial que se pueda aplicar de forma vertical en cualquier punto de nuestro sistema.

Dependiendo de qué tipo sea la política empresarial, el generador de código puede ayudarnos a introducirla en los lugares adecuados de nuestro software y encargase de "replicarla" en caso de que esta cambie.

2.2 Comparación con otras tecnologías generadoras

En esta sección vamos a revisar alternativas a la generación de código presentada por CapicuaGen, comparando beneficios y desventajas. Es importante señalar que debemos usar la herramienta adecuada para el problema que queremos resolver, y que posiblemente sea una combinación de diversas de estas, lo que necesitaremos. El enfocarnos en usar una herramienta exclusiva porque funcionó en otros proyectos, o porqué sea la estándar para nuestra empresa, sólo conseguirá dificultar un desarrollo, si no es que, directamente fracasar en su resolución

[4] Principio "Defense in Deep", unas de las buenas practicas mencionadas por Merkow y Raghavan [13].

(tanto en tiempo como en forma), como menciona Laplante, Hoffman y Klein, en su artículo sobre anti-patrones de software [9]. [5]

Code snippets

Los code snippets son fragmentos de código que se insertan automáticamente mediante una combinación de teclas establecida.

Plantillas integradas de IDE

Casi todos los IDE tienen plantillas integradas que nos permiten no comenzar de cero un nuevo desarrollo. Dichas plantillas se pueden instanciar al momento de crear un nuevo proyecto o al agregar un nuevo elemento.

Text Template Transformation Toolkit

Microsoft Visual Studio usa ampliamente los generadores y transformadores de código para facilitar el desarrollo de software automatizando una gran cantidad de tareas. Pero también nos da la posibilidad de generar y usar generadores de código como plantillas dinámicas a través de la tecnología Text Template Transformation Toolkit (abreviado como T4). Los archivos T4 permiten definir plantillas que serán procesadas al momento de ser guardadas y crearán el código fuente automáticamente.

MDA, Model-Driven Architecture

Como exponen Mellor, Scott y otros [10] MDA es una herramienta que nos permite definir un sistema a partir del diseño del modelo de la aplicación. Se comienza diseñando el negocio, generalmente en UML, y mediante una serie de transformaciones de modelos se genera el código fuente.

Es posible ver un análisis más en profundidad de MDA, en el texto del autor "MDA, Promesa y realidad" [11]

En la siguiente tabla se comparan las ventajas y desventajas de las tecnologías:

[5] El anti-patrón es conocido como el "martillo de oro". Existe un trabajo previo del autor que profundiza mas en dicho anti-patrón y en sus consecuencias [21].

Tabla 1. Comparativa de tecnologías alternas.

Tecnología	Sencillo de usar	Sencillo de cambiar	Cercano a la lógica del negocio	Cercano al código fuente	Fácil de reusar en otros proyectos	Tecnología usada ampliamente	Solución empresarial
Code Snippets	Sí	No	No	Sí	No	Sí	No
Plantillas integradas de IDE	Sí	No	No	Sí	Sí	Sí	No
Text Template Transformation Toolkit	Dificultad media	No	No	Sí	No	No	No
MDA, Model-Driven Architecture	No	Sí	Sí	Depende de la implementación de la herramienta.	Sí	No	Sí
CapicuaGen	Sí	Sí	Depende de la implementación del generador.	Depende de si estamos consumiendo o creando un generador.	Sí	No	Sí

Tecnología	Ventajas generales	Desventajas generales
Code Snippets	Sencillo de usar e incluido en la mayoría de los IDE.	Su ámbito es muy limitado (sólo son unas pocas líneas de código) y una vez insertado es como cualquier código escrito manualmente y debe ser mantenido como tal.
Plantillas integradas de IDE	Sencillo de usar e incluido en la mayoría de los IDE.	Son elementos estáticos que crean el código al momento de usarlas y no están pensadas en su mayoría para ser elementos actualizables a través del tiempo. Sólo son "atajos" para crear nuestro código más rápido.
Text Template Transformation Toolkit	Bastante poderoso en sus capacidades de generación.	La desventaja es que las plantillas pertenecen al proyecto en concreto y no es posible que sean compartidas entre varios proyectos empresariales, ni que sean actualizadas por un evento global externo.
MDA, Model-Driven Architecture	Trabajamos en conceptos más cercanos al negocio de una manera, presumiblemente sencilla. Las herramientas MDA, generan un código que se adapta a las necesidades técnicas vigentes o plataformas actuales.	La desventaja es que pese a las aparentes virtudes de MDA, y a que lleva un tiempo en el mercado, no hay una solución práctica que resuelva los problemas actuales de las aplicaciones empresariales
CapicuaGen	Sencillo de usar, escalable y adaptable a las necesidades empresariales actuales y futuras	Es una tecnología nueva, que requiere ser ampliada y difundida

Capítulo III. **SOLUCIÓN DESARROLLADA**

CapicuaGen es un sistema que ayuda a la creación automática de software empresarial a través de la definición y ensamblado de diversos generadores de características.

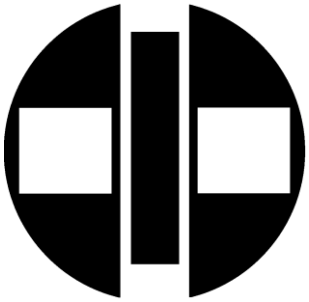


Ilustración 5. Logotipo de CapicuaGen.

Los generadores de características son herramientas especializadas en crear un aspecto en particular de un sistema destino. Un cambio de generador de características implicaría modificar un aspecto del sistema objetivo a crear, o generar un nuevo módulo dentro de nuestro sistema. En cualquier caso debiera minimizar el tiempo de construcción del software.

Entiéndase como características de un sistema, cada uno de los aspectos (que a su vez pueden dividirse en aspectos más pequeños) que deben implementar nuestros sistemas. Unas características pueden excluir a otras (como por ejemplo el motor de base de datos en el que vayamos a guardar la información, si usamos únicamente un repositorio) o pueden agregarse para trabajar como módulos independientes de nuestro sistema, por ejemplo podemos tener varias interfaces gráficas, como una interfaz para Windows, Mac, o dispositivos móviles (Android, iOS,…), siendo cada interfaz creada por un generador de características diferente.

CapicuaGen es una solución desarrollada en Ruby, cuya instalación es fácilmente realizable desde internet. Se puede encontrar más información en el anexo I **"Cómo obtener e instalar CapicuaGen"**.

3.1 Arquitectura de CapicuaGen

CapicuaGen es un framework, programado en Ruby, que se compone de un elemento central llamado **"generador**" (*generator*) en el que se configuran las **"características**" a generar (*features*) a través de **"objetivos"** (*targets*) a cumplir. Estos elementos se combinan mediante programación para generar un sistema destino.

Generator

Es el núcleo de CapicuaGen sobre el que se configuran todas las características a generar de un sistema, las cuales se crearán si son objetivos de la generación actual.

El generador poseerá entonces:

- Características que contendrá (o podrá contener) el sistema a generar.
- Objetivos a generar con base en las características agregadas.
- Atributos de la generación: Son configuraciones particulares y/o globales para una determinada generación. Las características pueden sobrescribir estas configuraciones para hacerlas particulares de ellas sin afectar la configuración global.

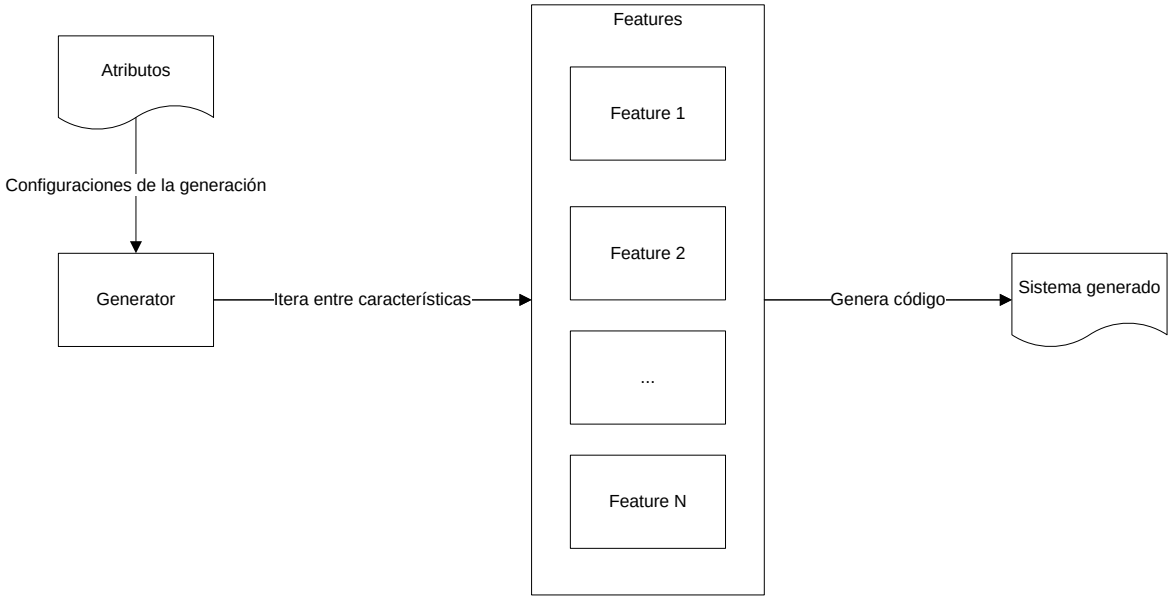


Ilustración 6. Flujo de generación de CapicuaGen.

Feature

Los generadores de características se encargan de crear cada uno de los aspectos de los sistemas destino cuando son llamados por el generador.

No hay ninguna política establecida sobre qué deben hacer las características una vez invocadas.

Algunos ejemplos son:

- Copiar archivos a rutas específicas.
- Modificar archivos creados previamente.
- Generar archivos en base a plantillas y a los atributos de generación.
- Proporcionar servicios que serán consumidos por otras características, como por ejemplo devolver qué entidades deben ser creadas.

Debido a lo común de la generación de archivos por plantillas se creó una característica para trabajar más fácilmente en dicho escenario.

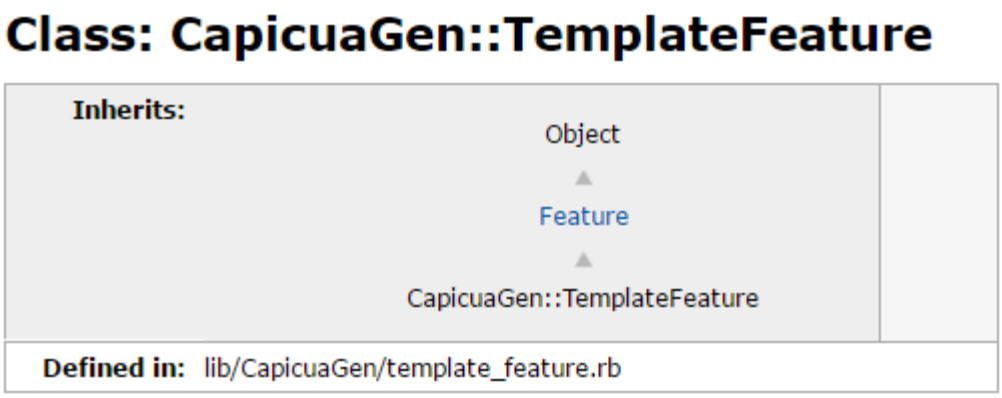


Ilustración 7. Clase CapicuaGen::TemplateFeature.

La clase que se encarga de generar archivos con base en plantillas es `CapicuaGen::TemplateFeature`, la cual además hereda las facultades de generación de la clase `Feature`. La clase es abstracta y no debe instanciarse directamente, sino heredarse para crear generadores de características basados en plantillas, los que tendrán siempre dos elementos:

- **Templates:** Son las plantillas que se usarán en el generador de la característica. Están en formato ERB (Embedded Ruby), pero también pueden ser archivos que simplemente serán copiados (sin ningún tipo de interpretación).
- **Targets:** Son los objetivos concretos a generar, o dicho de otra forma son los archivos a generar.

Un template podría estar asociado a uno o más objetivos, por ejemplo podemos tener un template para generar las entidades de negocio de nuestro modelo común para todas las entidades, sin embargo tendremos tantos targets a generar como entidades.

Módulos

Los generadores de características se agrupan en módulos por su semejanza. Por simplicidad un módulo equivale a una gema en Ruby; por ejemplo, tenemos el módulo

`CapicuaGenBalthazar` que agrupa características relacionadas con la generación de software en Android.

Se recomienda que los módulos comiencen con la palabra **CapicuaGen** y el nombre deseado. También se sugiere que el *namespace* (*module* en Ruby) se llame igual que el módulo y no usar ningún otro namespace anidado para hacerlo más simple al momento de usarlo.

La estructura general de un módulo es la siguiente:

- **Nombre del módulo:** Por ejemplo **CapicuaGenBalthazar**
  - **Lib**
    - **Modulo.rb:** Archivo que se llama igual que el módulo, pero sin CapicuaGen. Hace referencia a todas las características de la gema, de tal forma que con importar este archivo tengamos acceso a todas ellas.
    - **Nombre del módulo**
      - **Agrupación lógica de las características:** Se agrupan las características por su tipo o por la funcionalidad que están destinadas a hacer.
        - **Nombre de la característica**
          - **Source:** Archivo de la característica (acabado en feature).
          - **Templates:** Archivos *ERB* usados para generar la característica.
      - **Mixins:** Cualquier archivo que vaya a usarse como un mixin.
      - **Tools:** Cualquier herramienta o clase ayudante para nuestra librería.

Ejemplo 1. Organización de módulo de CapicuaGen.

Veamos el módulo CapicuaGenBalthazar

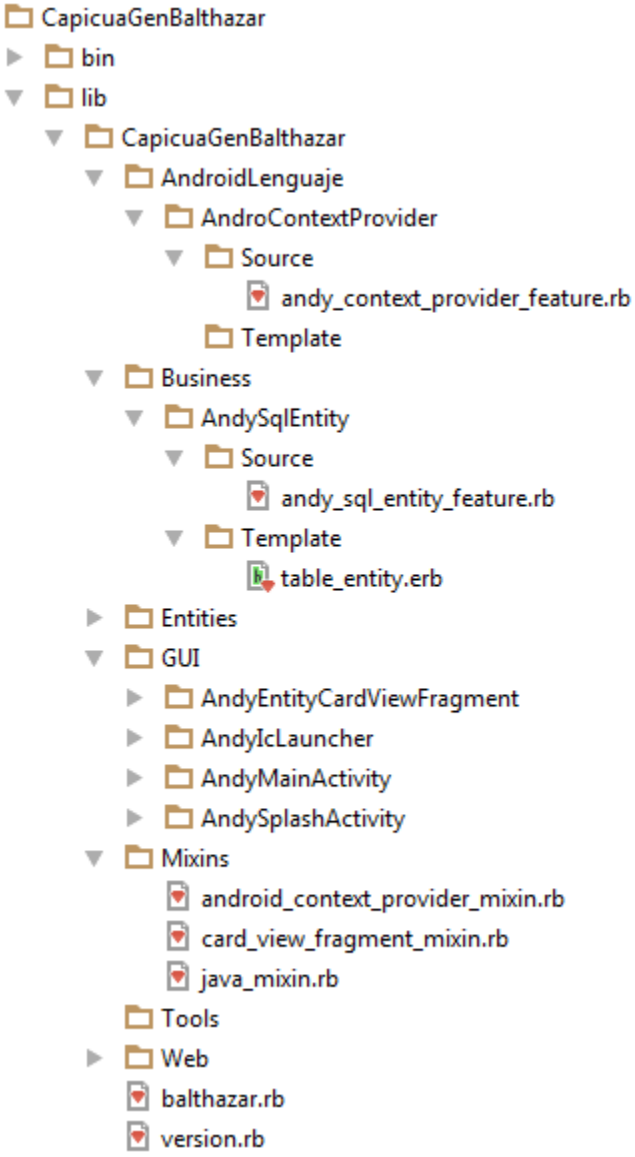
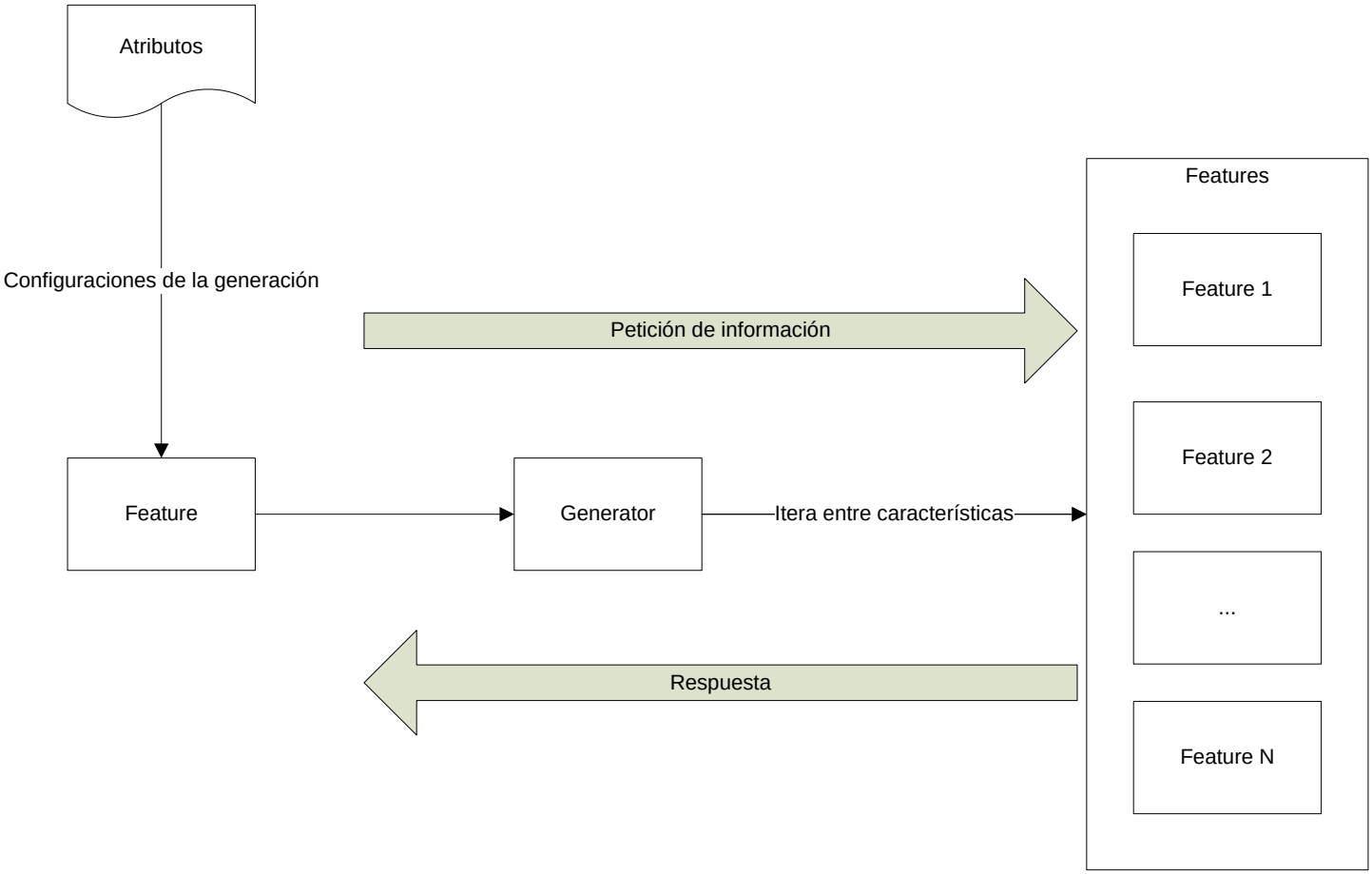


Ilustración 8. Organización CapicuaGenBalthazar

3.2 Interrelación de generadores de características

Los generadores de características son completamente independientes entre ellos, aunque pueden exponer funcionalidad como servicios y comunicarse a través de ellos.

Cada generador de características puede requerir información que proporcionan otros generadores. Se lanza una petición al generador para que investigue quién puede resolver dicha solicitud y una vez encontrado, se obtiene la información y esta es procesada por la característica solicitante. En el proceso, un generador de características obtiene información de otro (a través del generador central), sin que ni el origen o el destino se conozcan de alguna forma.



Ejemplo 2. Comunicación entre características.

Debido a que Ruby no tiene interfaces (elemento muy apropiado para esta necesidad en concreto), para saber qué servicios puede responder cada generador de características, simplemente se le asignará uno o varios tipos, o lo que es lo mismo una característica es de uno

o varios tipos. Según los tipos de los que sea la característica puede devolver determinada información o más bien responder a ciertos métodos que la regresen. De esta forma se emula el comportamiento de las interfaces de otros lenguajes.

Por ejemplo los generadores que poseen el tipo `:entity_sql_table`, pueden devolver entidades relacionadas a una tabla SQL (sin que importe de dónde obtuvieron la información de las tablas, que bien puede ser desde un script o directamente desde una base de datos).

Veamos el siguiente diagrama:

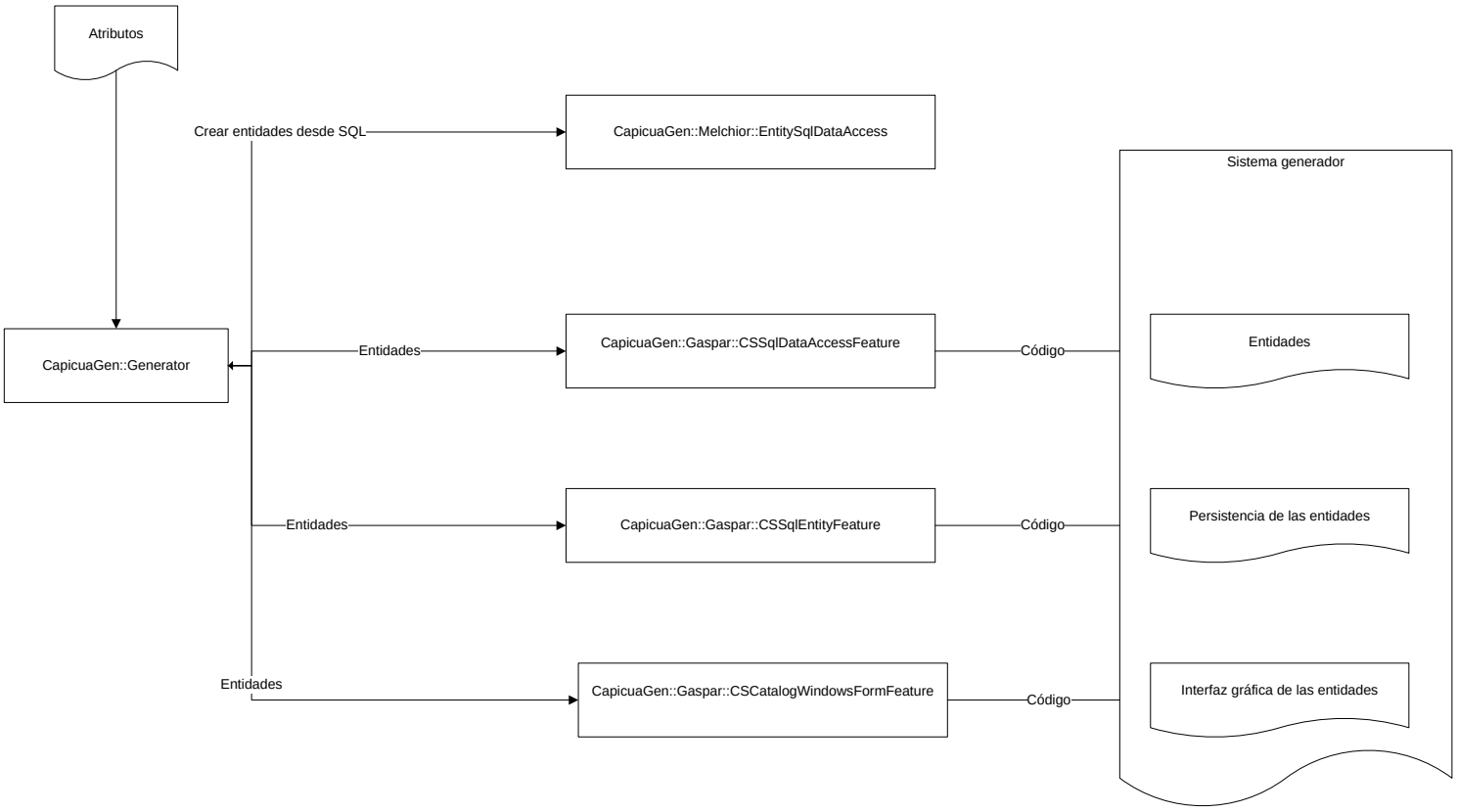


Ilustración 9. Ejemplo detallado de generación.

Tenemos los siguientes elementos:

- **EntitySqlDataAccess:** Analiza un archivo SQL y convierte sus definiciones de tablas en entidades abstractas tratables por el resto de características. Por ejemplo:

```
CREATE TABLE [dbo].[Territories]
    ([TerritoryID] [nvarchar] (20) NOT NULL ,
    [TerritoryDescription] [nchar] (50) NOT NULL ,
 [RegionID] [int] NOT NULL
) ON [PRIMARY]
GO
```

Se creará la siguiente entidad:

```
#<CapicuaGen::Melchior::EntitySchema:0x2f9f310
 @fields=[
       #<CapicuaGen::Melchior::EntityFieldSchema:0x2f9eff8
       @allow_null=false,
       @default_value=nil,
       @identity=false,
       @name="TerritoryID",
       @primary_key=false,
       @size="20",
       @sql_name="TerritoryID",
       @sql_type="nvarchar",
       @type="nvarchar">,

       #<CapicuaGen::Melchior::EntityFieldSchema:0x2f9ea10
       @allow_null=false,
       @default_value=nil,
       @identity=false,
       @name="TerritoryDescription",
       @primary_key=false,
       @size="50",
       @sql_name="TerritoryDescription",
       @sql_type="nchar",
       @type="nchar">,

       #<CapicuaGen::Melchior::EntityFieldSchema:0x2f9e638
       @allow_null=false,
       @default_value=nil,
       @identity=false,
       @name="RegionID",
       @primary_key=false,
       @size=nil,
       @sql_name="RegionID",
       @sql_type="int",
       @type="int">
],
 @name="Territories",
```

```
@sql_name="[dbo].[Territories]">
```

- **Generator:** Es el encargado de sincronizar las llamadas entre los distintos generadores de características. Los siguientes elementos requerirán las entidades creadas por el generador anterior, pero no accederán a él directamente, sino que a través de este generador central se pasará la información requerida. De esta forma, todos son independientes y sustituibles entre sí.

- **CSSqlDataAccessFeature**: Crea el código fuente necesario para la persistencia.

- **CSSQLEntityFeature:** Crea entidades en un código fuente de `C#` a partir de las entidades abstractas.

```
/*
* ---------------------------------------------------------------------
* Archivo Territories.cs
* ---------------------------------------------------------------------
* Generado automáticamente con CapicuaGen,
*
* CapicuaGen es un software que ayuda a la creación automática de
* sistemas empresariales a través de la definición y ensamblado de
* diversos generadores de características.
*
* El proyecto fue iniciado por José Luis Bautista Martín, el 6 de enero
* de 2016.
*
* Puede cambiar este encabezado haciendo un checkout de los templates de
* la caracteristica, y modificando el template 'header.erb'.
*
*          capicuagen.rb checkout CapicuaGenGaspar/CSHeaderFooter/template
*
*/

using Capicua.NorthWindServiceExample.Interface;

namespace Capicua.NorthWindServiceExample.Entities
{
 /// <summary>
 /// Entidad de Territories
 /// </summary>
 public partial class Territories : ITerritories
 {
 /// <summary>
 ///        Nombre: TerritoryID
 ///        Tipo: nvarchar
 ///        Permite Nulos: false
 ///        Valor por Defecto:
```

```
 ///        Llave Primaria:        false
 /// </summary>
 public string TerritoryID { get; set; }

 /// <summary>
 ///        Nombre: TerritoryDescription
 ///        Tipo: nchar
 ///        Permite Nulos: false
 ///        Valor por Defecto:
 ///        Llave Primaria:        false
 /// </summary>
 public string TerritoryDescription { get; set; }

 /// <summary>
 ///        Nombre: RegionID
 ///        Tipo: int
 ///        Permite Nulos: false
 ///        Valor por Defecto:
 ///        Llave Primaria:        false
 /// </summary>
 public int RegionID { get; set; }
 }
}

/*
* ---------------------------------------------------------------------
* Archivo Territories.cs
* ---------------------------------------------------------------------
* Generado automáticamente con CapicuaGen,
*
* Puede cambiar este pie de página haciendo un checkout de los templates de
* la característica, y modificando el template 'footer.erb'.
*
*          capicuagen.rb checkout CapicuaGenGaspar/CSHeaderFooter/template
*
*/
```

- **CSCatalogWindowsFormFeature:** Crea a través y para las entidades un catálogo de administración gráfica (altas, bajas, consultas y modificaciones).

3.3 Esquema de trabajo empresarial con CapicuaGen

El siguiente diagrama representa los elementos involucrados dentro del desarrollo de un producto software a través de CapicuaGen.

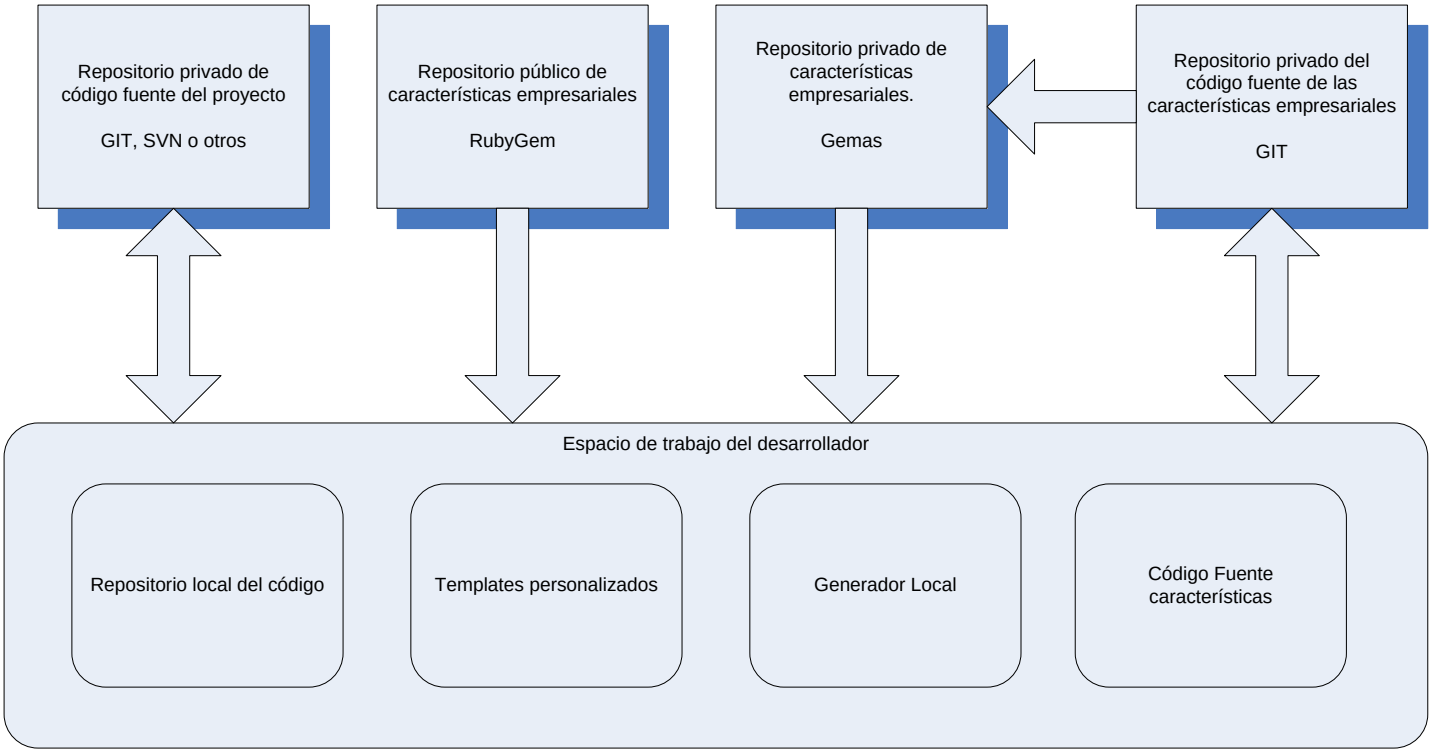


Ilustración 10. Esquema de trabajo empresarial con CapicuaGen.

Repositorio privado de código fuente del proyecto

Es el código fuente del proyecto almacenado en el repositorio de control de versiones de la empresa. El proyecto comienza con la creación de este repositorio y la posterior descarga local del código para poder trabajar con él. Lo más común es que sea un repositorio en Subversion (svn) o en Git.

Repositorio público de características empresariales

Es un repositorio público, ajeno a nuestra empresa, que ofrece generadores de características para CapicuaGen. Cualquiera podrá usar las características en sus proyectos (empresariales, privados o públicos). Los generadores están empaquetados dentro de gemas de Ruby y generalmente podrán descargase de RubyGems o de GitHub.

Repositorio privado de características empresariales

Semejante al repositorio anterior con la diferencia que almacena gemas que sólo podrán descargar e implementar los desarrolladores de una determinada compañía. Son por lo tanto generadores de índole privada, perteneciendo exclusivamente a dicha empresa y siendo esta la encargada de mantenerlos.

Repositorio privado del código fuente de las características empresariales

Es un repositorio de código fuente ajeno al repositorio del código del sistema que estamos construyendo. Su objetivo es almacenar el código fuente de los generadores de características que pertenecen a nuestra empresa y no son públicos. Es conveniente que este repositorio sea de Git.

Espacio de trabajo del desarrollador

Es el ambiente individual de desarrollo del programador, el cual está ubicado en su máquina de trabajo. Dicho ambiente debe estar en consonancia con los ambientes de otros desarrolladores y poder integrar y propagar los cambios del sistema en que se esté trabajando en ese momento.

Repositorio local del código

Es una versión del código fuente del proyecto actual integrado dentro del control de versiones. A través de dicha integración el programador puede subir cambios en el código y descargar los de otros programadores.

Templates personalizados

El programador puede exportar las plantillas (templates) de las gemas instaladas, y modificarlas para que se adapten a sus necesidades específicas. Si los cambios son integrados al control de versiones es posible realizar un "merge" (una mezcla) sobre futuras exportaciones de nuevas versiones de los templates.

Generador Local

Es un programa en Ruby que contiene el generador que creará el sistema y donde están configuradas todas las características que este usará. Cualquier personificación de la generación irá en este archivo.

Código Fuente características

Si no existiera un generador de características idóneo para nuestro problema, podríamos crear otro a nuestra medida e integrarlo con nuestro repositorio de código fuente privado. Al estar incluido en Git, los otros programadores pueden instalar directamente el nuevo generador (inclusive estando en construcción) como si de una gema se tratara (a través de una característica del gestor de dependencias `bundler`), de esta forma, parte del equipo podría estar creando las características generadoras y otra parte descargándolas y usándolas. Al final del desarrollo el código se convierte en una gema y se integra en nuestro repositorio de gemas empresariales.

3.4 Acerca de la configuración de generadores de CapicuaGen

CapicuaGen usa el esquema de definir qué es lo que debe hacerse mediante programación, en lugar de otras formas comunes como XML o un lenguaje del dominio específico (DSL). Se escogió esta opción porque se considera la más poderosa para poder indicar de la manera más clara y sencilla posible todas las opciones de CapicuaGen, además de permitir crecer al sistema de forma adecuada.

Se eligió Ruby como lenguaje para el desarrollo y la configuración, por los siguientes motivos:

- Es un lenguaje muy sencillo. Una persona con conocimientos en otros lenguajes de programación podría entenderlo fácilmente.
- También es poderoso, a la vez que tiene a su disposición multitud de APIs desarrolladas por terceros.
- El lenguaje es bastante reflexivo, pudiéndolo modificar para que se adapte a nuestras necesidades.

Veamos un ejemplo de programación de un generador, particularmente el del ejemplo creado automáticamente para Android.

```
# Creo el generador para android
generator_android = CapicuaGen::Generator.new do |g|

	# Generación de características
	feature_beans_entity = CapicuaGen::Balthazar::AndySqlEntityFeature.new(:name => 'feature_beans_entity')
	feature_json = CapicuaGen::Balthazar::AndyWebRequestFeature.new(:name => 'feature_json')
	feature_fragment = CapicuaGen::Balthazar::AndyEntityCardViewFragmentFeature.new(:name => 'feature_fragment')
	feature_splash = CapicuaGen::Balthazar::AndySplashActivityFeature.new(:name => 'feature_splash')
	feature_main = CapicuaGen::Balthazar::AndyMainActivityFeature.new(:name => 'feature_main')
	feature_android_context = CapicuaGen::Balthazar::AndyContextProvider.new(:name => 'feature_android_context')
	feature_ic_launcher = CapicuaGen::Balthazar::AndyIcLauncherFeature.new(:name => 'feature_ic_launcher')

	# Agrego las características al generador
	g.add_feature_and_target feature_entity,
		feature_beans_entity,
		feature_json,
		feature_fragment,
		feature_splash,
		feature_main,
		feature_header,
		feature_android_context,
		feature_ic_launcher

	 g.generation_attributes.add :out_dir => File.join(OUT_DIR, "generated/NorthWindAndroidExample/app/src/main"),
		:package => "com.capicua.northWindAndroidExample",
		:web_timeout => 60000,
		:remote_url => 'http://192.168.1.15/EjemploRESTful /Services/Catalogs.svc',
		:app_title => 'Titulo del ejemplo',
		:app_description => 'Descripción del ejemplo',
		:app_copyright => 'Copyright del ejemplo',
		:app_version => '1.0.0.0'


	 g.generation_attributes[:remote_url]= 'http://10.0.2.2/EjemploRESTful /Services/Catalogs.svc'


end
```

Ejemplo 3. Ejemplo de generación de Android.

Lo primero que hacemos es crear el generador con la sentencia:

```
generator_android = CapicuaGen::Generator.new
```

Al constructor se le puede pasar un bloque de ejecución que sirve para configurar los atributos del objeto creado, emulando de esta forma características de otros lenguajes que poseen el bloque `with..end with` o `fluet programing`. Esto es un método que se implementó en todos los constructores de CapicuaGen

```
generator_android = CapicuaGen::Generator.new do |g|
 …
 g.<Atributos del generador>
end
```

Ejemplo 5. Bloque constructor del generador.

También es posible crear el objeto necesario en una sola línea, pasando los atributos como parámetros del constructor de esta forma `:attributo => valor.`

```
 # Generación de características
 feature_beans_entity = CapicuaGen::Balthazar::AndySqlEntityFeature.new(:name =>
'feature_beans_entity')
```

Y por supuesto también puede asignarse simplemente estableciendo posteriormente el valor:

```
 feature_beans_entity.name = 'feature_beans_entity'
```

Para agregar valores a una lista de objetos, se pueden hacer uno a uno, o varios a la vez. En el ejemplo siguiente estamos agregando características (y objetivos) al generador:

```
 # Agrego las características al generador
 g.add_feature_and_target feature_entity,
```

```
    feature_beans_entity,
    feature_json,
    feature_fragment,
    feature_splash,
    feature_main,
    feature_header,
    feature_android_context,
    feature_ic_launcher
```

Ejemplo 4. Características y atributos agregados.

Del mismo modo podemos agregar los atributos globales de generación:

```
 g.generation_attributes.add :out_dir => File.join(OUT_DIR,
"generated/NorthWindAndroidExample/app/src/main"),
     :package => "com.capicua.northWindAndroidExample",
     :web_timeout => 60000,
     :remote_url => 'http://192.168.1.15/EjemploRESTful /Services/Catalogs.svc',
     :app_title => 'Titulo del ejemplo',
     :app_description => 'Descripción del ejemplo',
     :app_copyright => 'Copyright del ejemplo',
     :app_version => '1.0.0.0'
```

Ejemplo 5. Atributos agregados al generador.

O podemos configurarlos posteriormente como si fueran un hash:

```
 g.generation_attributes[:remote_url]= 'http://10.0.2.2/EjemploRESTful /Services/Catalogs.svc'
```

El aspecto general de la configuración de un generador es el siguiente:

```
generator_nombre = CapicuaGen::Generator.new do |g|
            <Creo las caracteristicas, pueden crearse fuera de este bloque>
            <Agrego las caracteristica en al generador>
            <Agrego los atributos>
end
```

```
<Ejecuto el generador>
```

Ejemplo 6. Esquema de creación de un generador.

De este modo tenemos una sencilla manera de crear nuevas configuraciones de generadores que se pueden leer de una forma clara, incluso más que un XML, pero que poseen toda la potencia de Ruby y al que podemos ampliar con características, APIs, y gemas de este lenguaje.

Capítulo IV. **VALIDACIÓN EXPERIMENTAL**

Para la validación experimental de CapicuaGen se generaron una serie de sistemas de muestra en diversas arquitecturas y lenguajes de programación. Todos ellos parten de la base de datos Northwind[6] (o más bien de su script de creación) y crean diversas entidades e interfaces gráficas relativas a sus tablas. Los ejemplos son:

Tabla 2. Tabla de ejemplos generados automáticamente.

Ejemplo	Tipo de Aplicación	Tecnología	Sistema Operativo	Lenguaje	IDE
Capicua.NorthWindWindowsExample	Escritorio	.NET	Windows	C#	Microsoft Visual Studio 2015
Capicua.NorthWindServiceExample	Servicio Web RESTful	.NET	Windows	C#	Microsoft Visual Studio 2015
NorthWindAndroidExample	Aplicación Móvil	Android	Android	Java	Android Studio 2.0

4.1 Ejemplo de aplicaciones generadas automáticamente

Ejemplo generado Capicua.NorthWindWindowsExample

Uno de los ejemplos generados es una aplicación Windows que accede a la base de datos Northwind y permite editar las tablas desde catálogos gráficos. Veamos unas ventanas de ejemplo (todas generadas automáticamente).

- **Ventana principal**

[6] Northwind es una popular base de datos de ejemplo ofrecida por Microsoft que simula ser una base de datos de una compañía real llamada de la misma forma. Para más información consultar https://northwinddatabase.codeplex.com/.

Ilustración 11. Pantalla principal de Capicua.NorthWindWindowsExample.

- **Menú de catálogos**

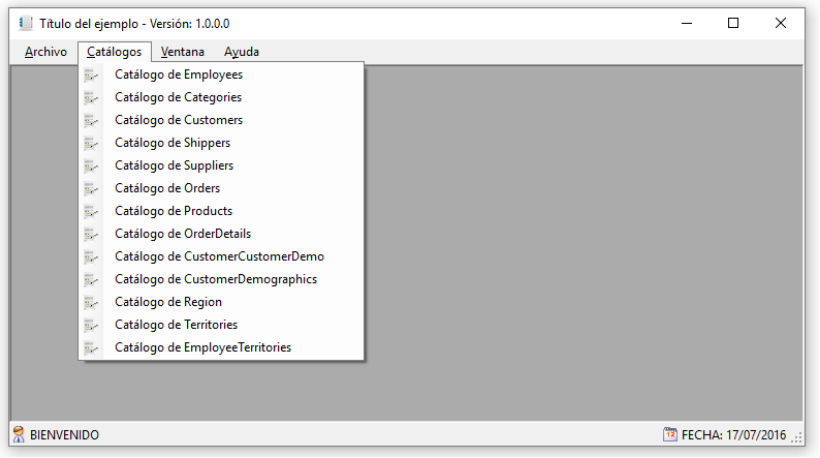


Ilustración 12. Menú de catálogos.

- **Catálogo de categorías**

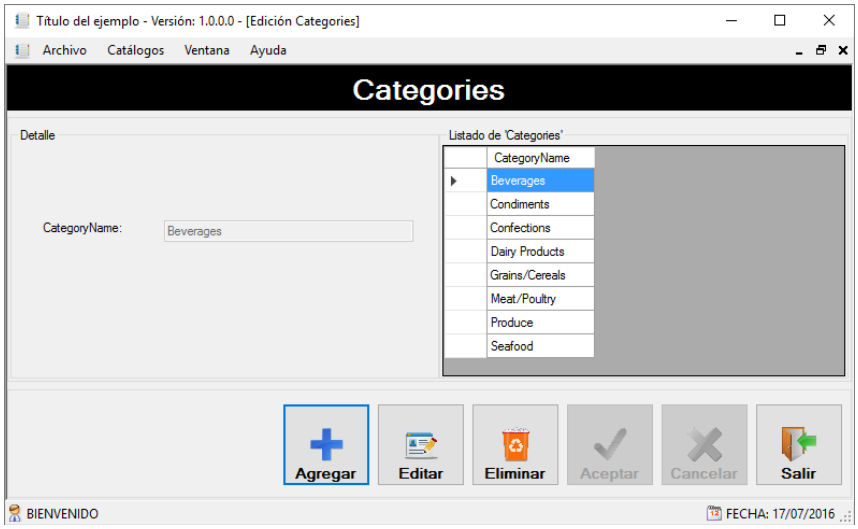


Ilustración 13. Catálogo de categorías.

Diagrama UML de la aplicación

A continuación se muestra el diagrama UML, que relaciona la entidad `Categorie` con su catálogo de mantenimiento:

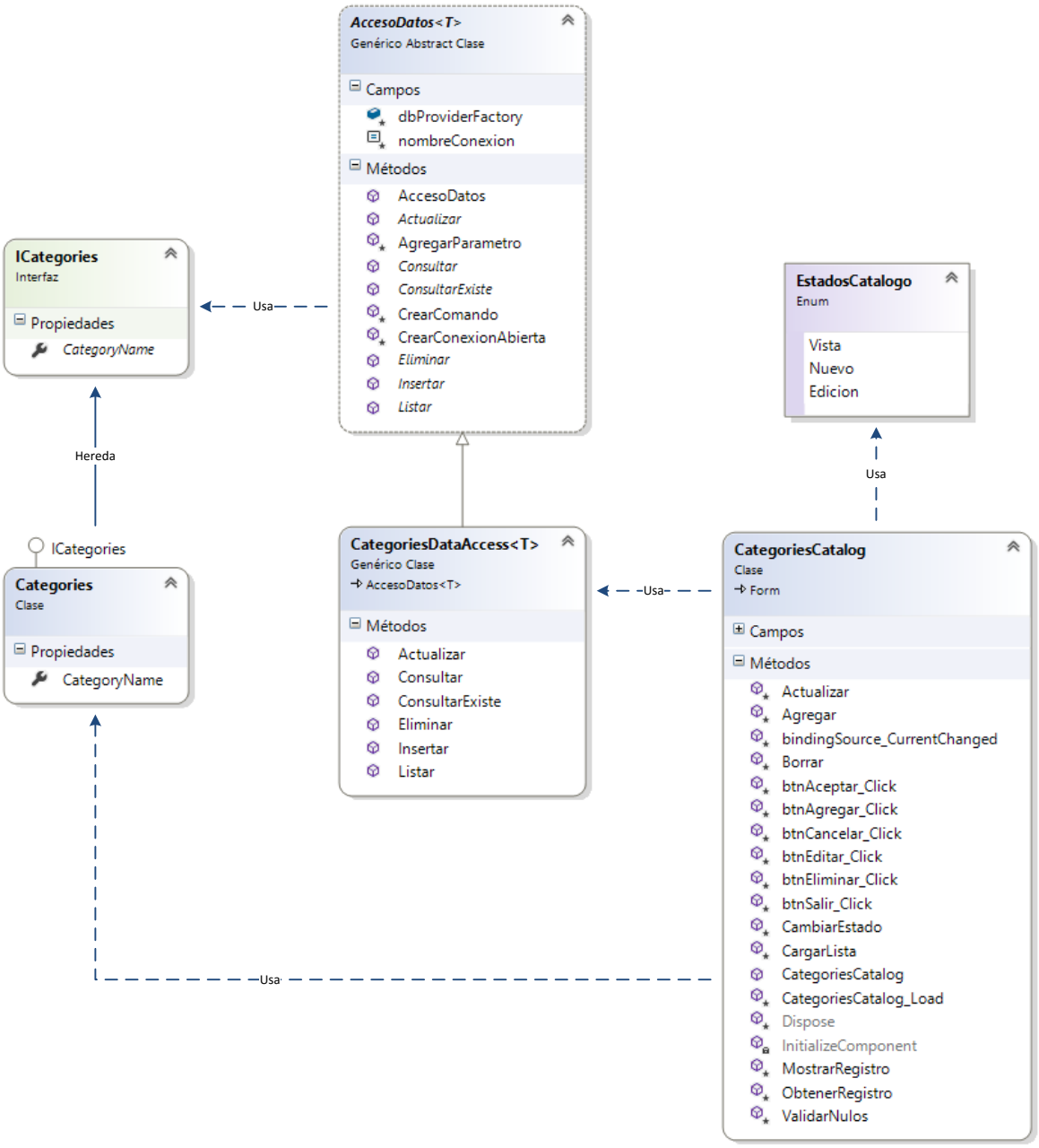


Ilustración 14. Diagrama UML de la entidad Categories en Windows Form.

Ejemplo de RESTful

Este ejemplo es una aplicación Web, que accede a la base de datos Northwind y expone las tablas a través de RESTful, devolviendo la información como JSON.

Diagrama UML de la aplicación

A continuación se muestra el diagrama UML que relaciona la entidad Category con la página web que muestra su contenido.

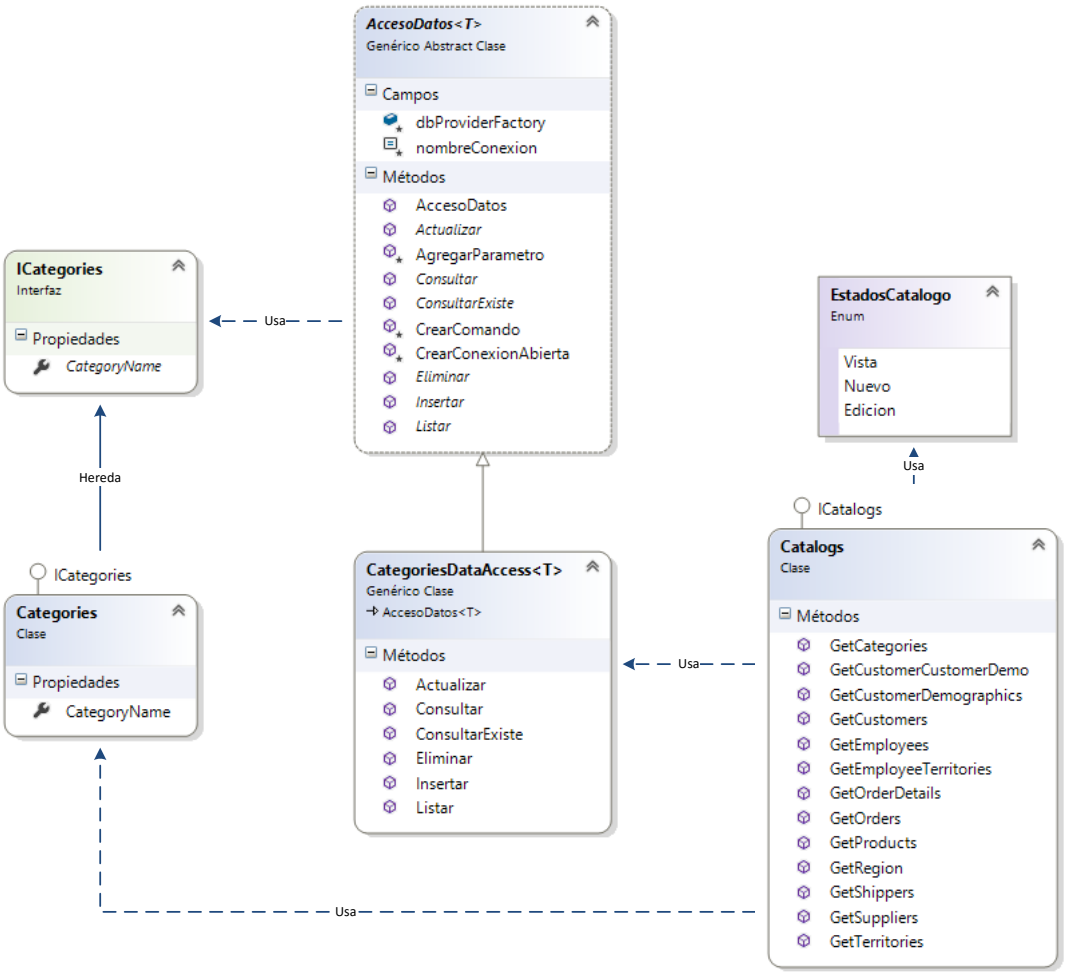


Ilustración 15. Diagrama UML de la entidad Categories para RESTFUL.

Ejemplo Android

El ejemplo para Android es una pequeña aplicación que se conecta por RESTful, a los catálogos expuestos en el ejemplo anterior.

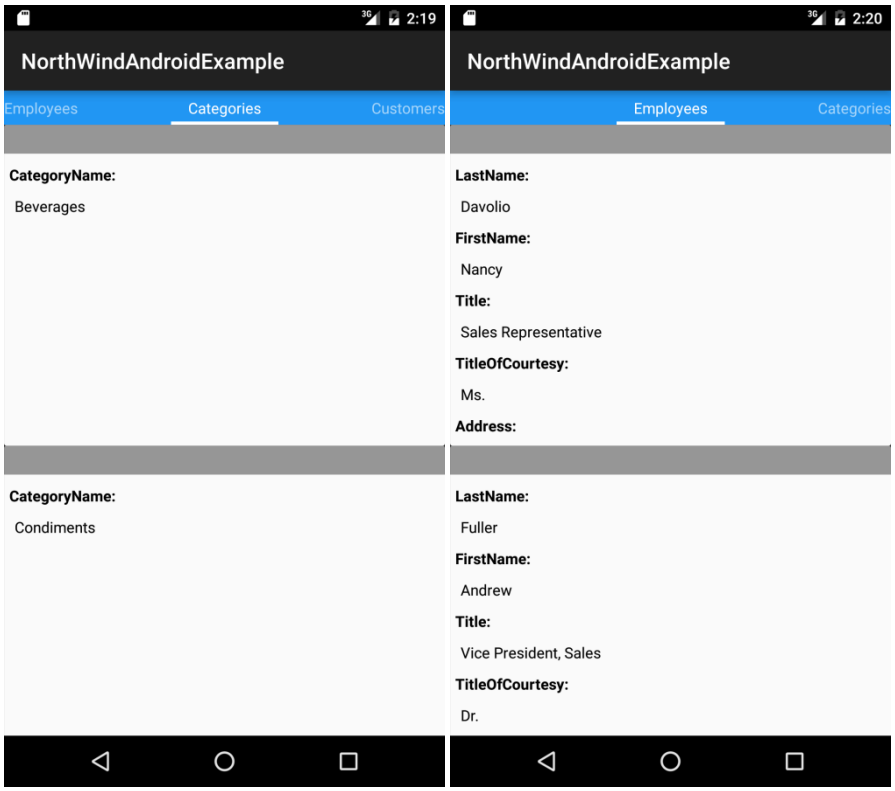


Ilustración 16. Aplicación Android.

Diagrama UML de la aplicación

A continuación se muestra el diagrama UML, que relaciona la entidad Categories con la actividad de Android que muestra el contenido.

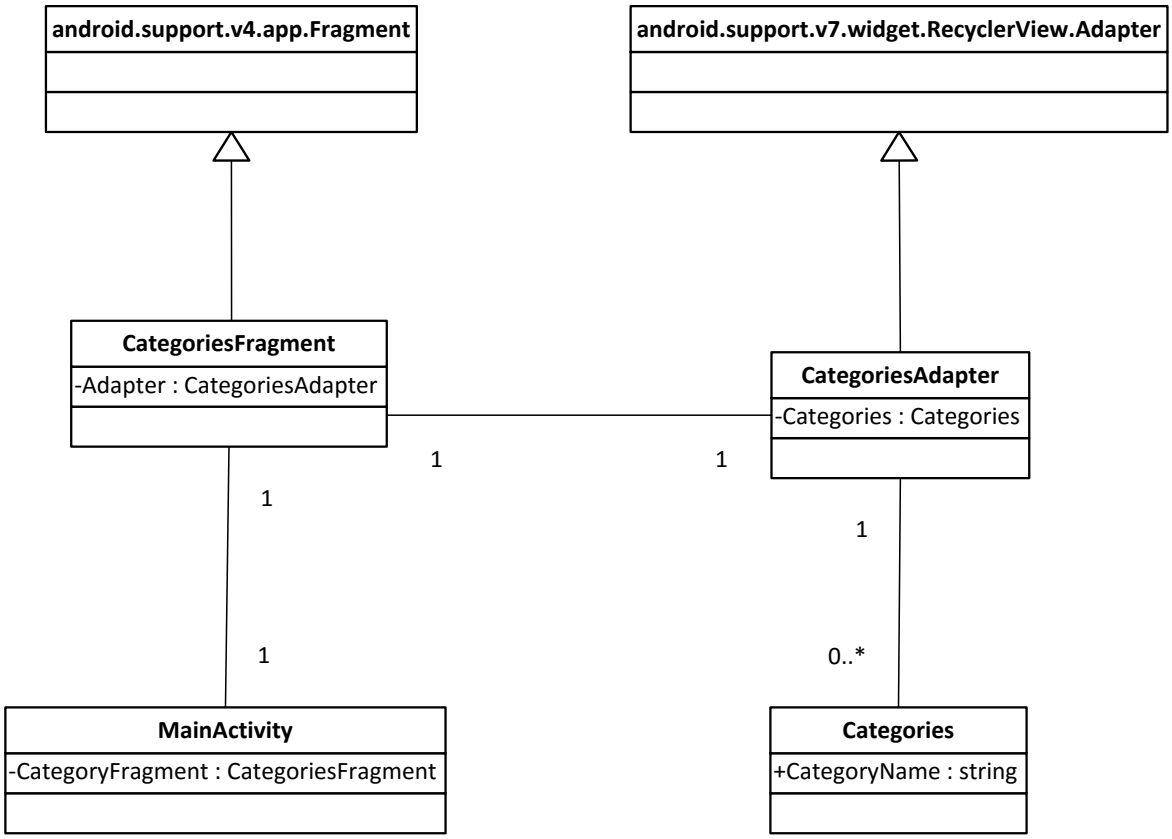


Ilustración 17. Diagrama UML de la case Categories para Android.

4.2 Generación de los ejemplos a través de CapicuaGen

CapicuaGen permite comenzar a trabajar desde el mismo momento en que es instalado. Para obtener los ejemplos mencionados previamente[7] simplemente ejecutamos el comando `CapicuaGen` con el parámetro `example`:

[7] Los ejemplos se incluyen dentro del mismo sistema de CapicuaGen para fines didácticos. Una vez creados automáticamente los futuros usuarios del generador pueden tomarlo como base para sus propios proyectos de generación. De esta forma es muy sencillo acercase a la herramienta por primera vez.

```
capicuagen example


Procesando característica: 'feature_example -> CapicuaGen::ExampleFeature'
 + generator.rb -> './generator.rb': Creado
 + GemFile -> './GemFile': Creado
 + instnwnd.sql -> './scripts/instnwnd.sql': Creado

Finalizado, tiempo total: 0.054999 segundos.
```

Ejemplo 7. Generación de ejemplo

Se crearán los siguientes archivos:

- **generator.rb:** Ejemplo de un generador de código.
- **GemFile:** Archivo de configuración de dependencias para Bundler.
- **instnwnd.sql**: Ejemplo de base de datos Northwind, para Microsoft SQL Server.

Generando el ejemplo

El generador de código de ejemplo se ejecuta como cualquier otro generador, con lo que podemos ver sus opciones ejecutando `--help`

```
>generator.rb --help


Uso: <Script generador> [comandos] [opciones]
Comandos:
 generate : Genera las características configuradas.
 clean : Limpia los archivos generados.
 cleanAndGenerate : Limpia los archivos generados y luego los vuelve a crear.
 example : Genera un ejemplo
 template : Lista o permite reemplazar plantillas por defecto.


Ejecute 'opt.rb COMMAND --help' para obtener más ayuda.
```

Resultado de pantalla 1. Ayuda a través del archivo "generator.rb".

Para crear el código en sí, bastará con ejecutarlo con la opción `generate`

```
>generator.rb generate



Este es un ejemplo de CapicuaGen

CapicuaGen es un software que ayuda a la creación automática de
sistemas empresariales a través de la definición y ensamblado de
diversos generadores de características.

Este es un ejemplo generado automáticamente para comprender el uso
de CapicuaGen, incluye el uso de generadores de los siguientes tipos:

CapicuaGenEssential agrega referencia a los generadores de características

* Melchior: Analizador de script SQL.
* Gaspar: Generador de código en C#
* Balthazar: Generador de código en Android

¿Quieres ver el código para generar tus propias características? Revisa
los siguientes repositorios:


CapicuaGen -> https:// Github.com/jbautistamartin/CapicuaGen
CapicuaGenMelchior -> https:// Github.com/jbautistamartin/CapicuaGenMelchior
CapicuaGenGaspar -> https:// Github.com/jbautistamartin/CapicuaGenGaspar
CapicuaGenBalthazar -> https:// Github.com/jbautistamartin/CapicuaGenBalthazar


Se incluye deshabilitada la característica de formatear apropiadamente código fuente C#,
la característica es 'CodeMaidCleanerFeature', que aunque muy interesante, es MUY
lenta. Puede activarse esta característica cambiando el código de este archivo:

ENABLE_CODE_MAID_FEATURE = true


Pulse enter para continuar.

Procesando característica: 'proyect_web -> CapicuaGen::Gaspar::CSProyectRESTful Feature'
 + Soluction.erb -> './Capicua.NorthWindServiceExample/../Capicua.NorthWindServiceExample.sln':
Creado
 + Proyect.erb -> './Capicua.NorthWindServiceExample/Capicua.NorthWindServiceExample.csproj': Creado
 + Web.erb -> './Capicua.NorthWindServiceExample/Web.Config': Creado
 + AssemblyInfo.erb -> './Capicua.NorthWindServiceExample/Properties/AssemblyInfo.cs': Creado
 + ExcepcionControlada.erb -> './Capicua.NorthWindServiceExample/ExcepcionControlada.cs': Creado

Procesando característica: 'feature_catalog_service -> CapicuaGen::Gaspar::CSRESTFUL CatalogFeature'
 + CatalogState.erb -> './Capicua.NorthWindServiceExample/Services/CatalogState.cs': Creado
 + Catalogs.erb -> './Capicua.NorthWindServiceExample/Services/Catalogs.svc.cs': Creado
 + Catalogs.svc.erb -> './Capicua.NorthWindServiceExample/Services/Catalogs.svc': Creado
 + ICatalogs.erb -> './Capicua.NorthWindServiceExample/Services/ICatalogs.cs': Creado

Procesando característica: 'feature_business_entity -> CapicuaGen::Gaspar::CSSqlEntityFeature'
 + table_entity.erb -> './Capicua.NorthWindServiceExample/Entities/Employees.cs': Creado
 + table_entity.erb -> './Capicua.NorthWindServiceExample/Entities/Categories.cs': Creado
 + table_entity.erb -> './Capicua.NorthWindServiceExample/Entities/Customers.cs': Creado
 + table_entity.erb -> './Capicua.NorthWindServiceExample/Entities/Shippers.cs': Creado
 + table_entity.erb -> './Capicua.NorthWindServiceExample/Entities/Suppliers.cs': Creado
```

```
 + table_entity.erb -> './Capicua.NorthWindServiceExample/Entities/Orders.cs': Creado
 + table_entity.erb -> './Capicua.NorthWindServiceExample/Entities/Products.cs': Creado
 + table_entity.erb -> './Capicua.NorthWindServiceExample/Entities/OrderDetails.cs': Creado
 + table_entity.erb -> './Capicua.NorthWindServiceExample/Entities/CustomerCustomerDemo.cs': Creado
 + table_entity.erb -> './Capicua.NorthWindServiceExample/Entities/CustomerDemographics.cs': Creado
 + table_entity.erb -> './Capicua.NorthWindServiceExample/Entities/Region.cs': Creado
 + table_entity.erb -> './Capicua.NorthWindServiceExample/Entities/Territories.cs': Creado
 + table_entity.erb -> './Capicua.NorthWindServiceExample/Entities/EmployeeTerritories.cs': Creado

Procesando característica: 'feature_header -> CapicuaGen::Gaspar::CSHeaderFooterFeature'
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Data.Access\DataAccessException.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Data.Access\DataAccess.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Data.Access\EmployeesDataAccess.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Data.Access\CategoriesDataAccess.cs':
Sobreescrit
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Data.Access\CustomersDataAccess.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Data.Access\ShippersDataAccess.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Data.Access\SuppliersDataAccess.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Data.Access\OrdersDataAccess.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Data.Access\ProductsDataAccess.cs':
Sobreescrito
 * header.erb, footer.erb ->
'Capicua.NorthWindServiceExample\Data.Access\OrderDetailsDataAccess.cs': Sobreescr
 * header.erb, footer.erb ->
'Capicua.NorthWindServiceExample\Data.Access\CustomerCustomerDemoDataAccess.cs': S
 * header.erb, footer.erb ->
'Capicua.NorthWindServiceExample\Data.Access\CustomerDemographicsDataAccess.cs': S
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Data.Access\RegionDataAccess.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Data.Access\TerritoriesDataAccess.cs':
Sobreescri
 * header.erb, footer.erb ->
'Capicua.NorthWindServiceExample\Data.Access\EmployeeTerritoriesDataAccess.cs': So
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Interface\IEmployees.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Interface\ICategories.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Interface\ICustomers.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Interface\IShippers.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Interface\ISuppliers.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Interface\IOrders.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Interface\IProducts.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Interface\IOrderDetails.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Interface\ICustomerCustomerDemo.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Interface\ICustomerDemographics.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Interface\IRegion.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Interface\ITerritories.cs':
Sobreescrito
```

```
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Interface\IEmployeeTerritories.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Properties\AssemblyInfo.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\ExcepcionControlada.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Services\CatalogState.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Services\Catalogs.svc.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Services\ICatalogs.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Entities\Employees.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Entities\Categories.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Entities\Customers.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Entities\Shippers.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Entities\Suppliers.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Entities\Orders.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Entities\Products.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Entities\OrderDetails.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Entities\CustomerCustomerDemo.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Entities\CustomerDemographics.cs':
Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Entities\Region.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Entities\Territories.cs': Sobreescrito
 * header.erb, footer.erb -> 'Capicua.NorthWindServiceExample\Entities\EmployeeTerritories.cs':
Sobreescrito


Finalizado, tiempo total: 8.275827 segundos.
```

Resultado de pantalla 2. Ejecución del generador de ejemplo.

Personalización de los ejemplos generados

Los ejemplos están generados a través de plantillas, las cuales es posible modificar para personalizar el resultado final. El proceso para modificar plantillas es el siguiente:

1. Opcionalmente listar las plantillas instaladas:

```
>capicuagen template –l

 gem: 'CapicuaGen', type: 'Examples', feature: 'Example'
 gem: 'CapicuaGenBalthazar', type: 'Business', feature: 'AndySqlEntity'
 gem: 'CapicuaGenBalthazar', type: 'GUI', feature: 'AndyEntityCardViewFragment'
 gem: 'CapicuaGenBalthazar', type: 'GUI', feature: 'AndyIcLauncher'
 gem: 'CapicuaGenBalthazar', type: 'GUI', feature: 'AndyMainActivity'
 gem: 'CapicuaGenBalthazar', type: 'GUI', feature: 'AndySplashActivity'
 gem: 'CapicuaGenBalthazar', type: 'Web', feature: 'AndyWebRequest'
 gem: 'CapicuaGenGaspar', type: 'Business', feature: 'CSSqlEntitityInterface'
 gem: 'CapicuaGenGaspar', type: 'Business', feature: 'CSSqlEntity'
```

```
 gem: 'CapicuaGenGaspar', type: 'CodeTransformer', feature: 'CSHeaderFooter'
 gem: 'CapicuaGenGaspar', type: 'DataAccess', feature: 'CSSqlDataAccess'
 gem: 'CapicuaGenGaspar', type: 'GUI', feature: 'CSAboutWindowsForm'
 gem: 'CapicuaGenGaspar', type: 'GUI', feature: 'CSCatalogWindowsForm'
 gem: 'CapicuaGenGaspar', type: 'GUI', feature: 'CSMDIWindowsForm'
 gem: 'CapicuaGenGaspar', type: 'GUI', feature: 'CSSplashWindowsForm'
 gem: 'CapicuaGenGaspar', type: 'Proyect', feature: 'CSProyectRESTful '
 gem: 'CapicuaGenGaspar', type: 'Proyect', feature: 'CSProyectWindowsForm'
 gem: 'CapicuaGenGaspar', type: 'WCFService', feature: 'CSRESTful Catalog'
```

Resultado de pantalla 3. Listado de plantillas instaladas.

2. Exportar las plantillas que se desean modificar:

   - Cabeceras:

```
>capicuagen template -g CapicuaGenGaspar -t CodeTransformer -f CSHeaderFooter

+ CapicuaGenGaspar, CodeTransformer, CSHeaderFooter ->
'capicua/CapicuaGenGaspar/CodeTransformer/CSHeaderFooter/footer.erb': Creado
+ CapicuaGenGaspar, CodeTransformer, CSHeaderFooter ->
'capicua/CapicuaGenGaspar/CodeTransformer/CSHeaderFooter/header.erb': Creado
```

Resultado de pantalla 4. Exportación de plantillas de cabeceras.

   - Pantalla de inicio:

```
>capicuagen template -g CapicuaGenGaspar -t GUI -f CSSplashWindowsForm

+ CapicuaGenGaspar, GUI, CSSplashWindowsForm ->
'capicua/CapicuaGenGaspar/GUI/CSSplashWindowsForm/SplashForm.designer.erb': Creado
+ CapicuaGenGaspar, GUI, CSSplashWindowsForm ->
'capicua/CapicuaGenGaspar/GUI/CSSplashWindowsForm/SplashForm.erb': Creado
+ CapicuaGenGaspar, GUI, CSSplashWindowsForm ->
'capicua/CapicuaGenGaspar/GUI/CSSplashWindowsForm/SplashForm.resx.erb': Creado
```

Resultado de pantalla 5. Exportación de plantillas de pantalla de inicio.

3. Eliminar los archivos que no queremos.

Cuando se exporta una plantilla se exportan todas las plantillas de la característica seleccionada. Es conveniente eliminar las que no van a modificarse para que se usen las que están dentro de CapicuaGen y no las exportadas en el directorio local.

4. Modificar las plantillas

Una vez exportadas las plantillas exactas, las modificaremos para que se adapten a nuestras necesidades. Por ejemplo podemos modificar las plantillas de cabeceras y pies de página:

- **Plantilla:**
  `capicua\CapicuaGenGaspar\CodeTransformer\CSHeaderFooter\header.erb`
- **Código original:**

```
<%
#encoding: UTF-8

=begin

CapicuaGen

CapicuaGen es un software que ayuda a la creación automática de
sistemas empresariales a través de la definición y ensamblado de
diversos generadores de características.

El proyecto fue iniciado por José Luis Bautista Martín, el 6 de enero
de 2016.

Puede modificar y distribuir este software, según le plazca, y usarlo
para cualquier fin ya sea comercial, personal, educativo, o de cualquier
índole, siempre y cuando incluya este mensaje, y se permita acceso al
código fuente.

Este software es código libre, y se licencia bajo LGPL.

Para más información consultar http://www.gnu.org/licenses/lgpl.html
=end
%>/*
* ------------------------------------------------------------------------
* Archivo <%=file_name%>
* ------------------------------------------------------------------------
* Generado automáticamente con CapicuaGen,
*
* CapicuaGen es un software que ayuda a la creación automática de
```

```
* sistemas empresariales a través de la definición y ensamblado de
* diversos generadores de características.
*
* El proyecto fue iniciado por José Luis Bautista Martín, el 6 de enero
* de 2016.
*
* Puede cambiar este encabezado haciendo un checkout de los templates de
* la característica, y modificando el template 'header.erb'.
*
*          capicuagen template -g CapicuaGenGaspar -t CodeTransformer -f CSHeaderFooter
*
*/
```

Ejemplo 8. Plantilla original cabecera.

- **Plantilla modificada:**

```
<%
#encoding: UTF-8

=begin

CapicuaGen

CapicuaGen es un software que ayuda a la creación automática de
sistemas empresariales a través de la definición y ensamblado de
diversos generadores de características.

El proyecto fue iniciado por José Luis Bautista Martín, el 6 de enero
de 2016.

Puede modificar y distribuir este software, según le plazca, y usarlo
para cualquier fin ya sea comercial, personal, educativo, o de cualquier
índole, siempre y cuando incluya este mensaje, y se permita acceso al
código fuente.

Este software es código libre, y se licencia bajo LGPL.

Para más información consultar http://www.gnu.org/licenses/lgpl.html
=end
%>/*
* ------------------------------------------------------------------------
* Archivo <%=file_name%>
* ------------------------------------------------------------------------
* Generado automáticamente con CapicuaGen,
*
* Ejemplo NorthWind
*
* Autor: José Luis Bautista Martín
*
*/
```

Ejemplo 9. Plantilla modificada cabecera.

Igualmente podemos modificar la ventana de inicio de la aplicación generada para, por ejemplo, cambiar la imagen por una corporativa.

- **Plantilla:**
  `\capicua\CapicuaGenGaspar\GUI\CSSplashWindowsForm\SplashForm.resx.erb`

```
<%
#encoding: UTF-8

=begin

CapicuaGen

CapicuaGen es un software que ayuda a la creación automática de
sistemas empresariales a través de la definición y ensamblado de
diversos generadores de características.

El proyecto fue iniciado por José Luis Bautista Martín, el 6 de enero
de 2016.

Puede modificar y distribuir este software, según le plazca, y usarlo
para cualquier fin ya sea comercial, personal, educativo, o de cualquier
índole, siempre y cuando incluya este mensaje, y se permita acceso al
código fuente.

Este software es código libre, y se licencia bajo LGPL.

Para más información consultar http://www.gnu.org/licenses/lgpl.html
=end
%><?xml version="1.0" encoding="utf-8"?>
<root>
...
 <data name="$this.BackgroundImage" type="System.Drawing.Bitmap, System.Drawing"
mimetype="application/x-microsoft.net.object.bytearray.base64">
 <value>a
 iVBORw0KGgoAAAANSUhEUgAAA+gAAAPoCAIAAADCwUOzAAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8
 YQUAAAAJcEhZcwAADsIAAA7CARUoSoAAAFo0SURBVHhe7dbbltw6DmXR/v+f7u5TXmeUs+y080KEKGrO
 ...
</value>
 </data>
</root>
```

Ejemplo 10. Plantilla original de la ventana de bienvenida.

Para modificarlo seleccionamos una imagen y la convertimos en Base64 sustituyéndola en el archivo anterior (podemos usar por ejemplo esta página https://www.base64-image.de/)

Si quisiéramos cambiar la imagen de la aplicación de Android de prueba ejecutaríamos:

```
>capicuagen template -g CapicuaGenBalthazar -t GUI -f AndySplashActivity

+ CapicuaGenBalthazar, GUI, AndySplashActivity ->
'capicua/CapicuaGenBalthazar/GUI/AndySplashActivity/logo.png': Creado
+ CapicuaGenBalthazar, GUI, AndySplashActivity ->
'capicua/CapicuaGenBalthazar/GUI/AndySplashActivity/splash.erb': Creado
+ CapicuaGenBalthazar, GUI, AndySplashActivity ->
'capicua/CapicuaGenBalthazar/GUI/AndySplashActivity/splash_activity.erb': Creado
```

Ejemplo 11. Exportación de plantillas de Android.

Modificaríamos el archivo `logo.png` con el logo que quisiéramos en nuestra aplicación.

5. Una vez configuradas las plantillas, podemos volver a generar todo el código con el comando, "limpiar y generar".

```
>generator.rb cleanAndGenerate
```

Ejemplo 12. Comando cleanAndGenerate.

Capítulo V. **CONCLUSIONES Y TRABAJO FUTURO**

5.1 Conclusiones

Habiendo desempeñado gran parte de mi vida laboral como arquitecto de software, mis intereses profesionales van enfocados a crear sistemas homogéneos, escalables, preparados para el cambio y que además tengan una correcta interacción entre ellos y entre otros de naturaleza heterogénea y externa.

Debido a esto dediqué mi interés a los generadores automáticos de código, viendo la cantidad de elementos y fases repetibles existentes en el desarrollo de un sistema. El problema inherente de esto era la falta de homogenización entre distintos generadores "pequeños" cuyo enfoque era resolver un problema en concreto, pero que a la vez dependía tanto de dicho problema que un cambio en este representaría un cambio profundo del generador.

CapicuaGen intenta ser un agrupador lógico, reutilizable y eficaz de "generadores de características" completamente independientes entre ellos y enfocados a resolver diversos aspectos de un sistema, estando además preparados para integrarse de forma adecuada y desde un inicio, al ciclo de trabajo de un desarrollo empresarial.

Con CapicuaGen se espera eliminar la necesidad de realizar manualmente multitud de tareas repetitivas, además de implementar los cambios y variaciones de un sistema de forma automática. Con esto se intenta conseguir que los desarrolladores puedan invertir su talento y creatividad en las tareas específicas que lo requieran, en lugar de gastarlo en las que no.

Es todavía pronto para poder dar datos estadísticos certeros de cuánto tiempo podemos ahorrar en la construcción de un software. Sin embargo, debido a que, para la realización de este trabajo se creó una serie de sistemas tanto manualmente como automáticamente, se compararon los tiempos invertidos en ambos procesos para poder aproximar la ganancia dentro de un escenario real.

Se mostrará el tiempo requerido para crear las entidades necesarias para los sistemas generados, así como cada una de las capas que componen dichas entidades (se tendrá en

cuenta además, que sucesivas creaciones de las entidades tanto manuales, como automáticas, requieren menos tiempo que la primera).

Tabla 3. Comparativa de diversos tiempos de generación

		Tiempo invertido en horas										
Entidades	1	Análisis	Diseño	Elementos SQL	Acceso Datos	Objetos Simples	Entidades Negocio	Servicios	Interfaces			Totales
Recursos	1								Escritorio	Web	Móvil	
Generación manual		01:00	01:00	01:00	03:00	02:00	05:00	02:00	05:00	05:00	10:00	35:00
Generación automática		01:00	01:00	01:00	00:10	00:05	00:05	00:05	01:00	01:00	02:00	07:25
											Diferencia	27:35
											Ganancia	79%

		Tiempo invertido en horas										
Entidades	5	Análisis	Diseño	Elementos SQL	Acceso Datos	Objetos Simples	Entidades Negocio	Servicios	Interfaces			Totales
Recursos	1								Escritorio	Web	Móvil	
Generación manual		03:00	03:00	03:00	12:00	06:00	15:00	10:00	15:00	15:00	30:00	112:00
Generación automática		03:00	03:00	03:00	00:14	00:07	00:07	00:07	01:24	01:24	02:48	15:11
											Diferencia	96:49
											Ganancia	86%

		Tiempo invertido en horas										
Entidades	100	Análisis	Diseño	Elementos SQL	Acceso Datos	Objetos Simples	Entidades Negocio	Servicios	Interfaces			Totales
Recursos	1								Escritorio	Web	Móvil	
Generación manual		50:30	50:30	50:30	225:45	101:00	252:30	200:00	252:30	252:30	505:00	1940:45
Generación automática		50:30	50:30	50:30	01:49	00:54	00:54	00:54	10:54	10:54	21:48	199:38
											Diferencia	1741:06
											Ganancia	90%

Si visualizamos esto gráficamente en la construcción de aproximadamente 100 entidades, obtendremos una información muy relevante sobre la ganancia real del uso de CapicuaGen:

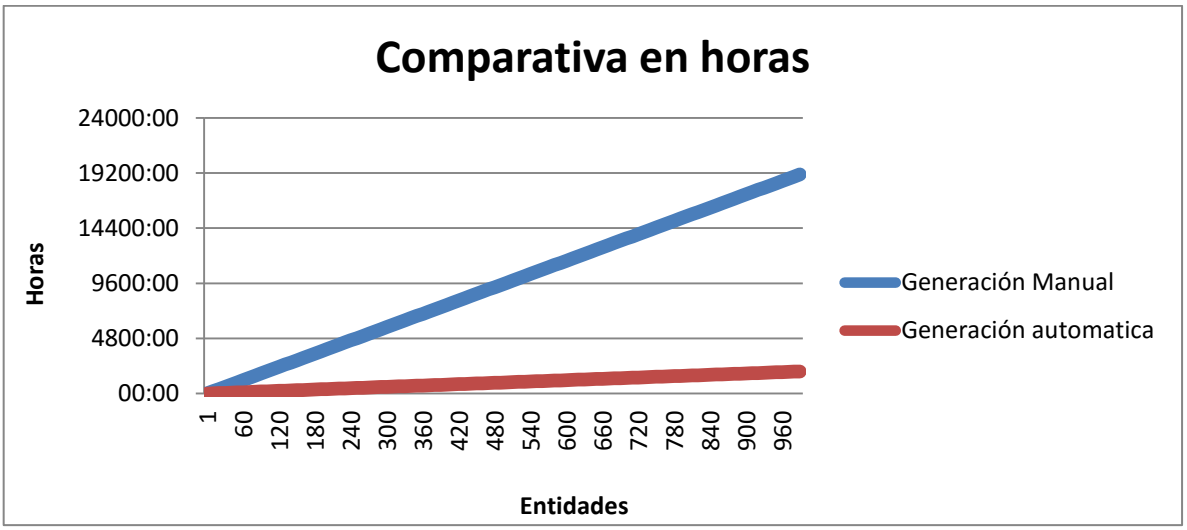


Ilustración 18. Comparativa de tiempos de generación.

5.2 Trabajo futuro y posible impacto en el mercado empresarial

Es difícil introducir generadores de código en las organizaciones porque crean cierta desconfianza infundada, como bien menciona Jack Herrington [12].Es necesario vencer dicha desconfianza y proponer a la empresa un escenario positivo y de mejora en sus procesos. El mejor argumento para las compañías, es que con los mismos recursos (económicos, físicos y humanos) van a poder realizar más software, en menos tiempo y con más calidad.

En cuanto a los motivos intrínsecos que tiene CapicuaGen para triunfar están; que es un software libre y fácilmente accesible por medios conocidos como GitHub y RubyGems, es estructural, modular y diseñado para que sea fácilmente comprensible y ampliable con nuevas características. En la actualidad y como primera versión, puede resolver un número muy limitado de necesidades casi educativas.

El trabajo futuro de esta herramienta es básicamente aumentar el número de aspectos que puede resolver y compartir, en el repositorio público de la misma, los generadores creados

para que cada vez más personas y en circunstancias más diversas puedan resolver sus necesidades de generación de código. Para esto, se requiere la colaboración de equipos de personas, que estén convencidos de la potencia de la herramienta y quieran colaborar en ella. El carácter libre y público de CapicuaGen se presta fácilmente para tal efecto.

BIBLIOGRAFÍA CONSULTADA.

A continuación se muestra un listado de todas las fuentes que se consultaron para la creación de la solución de CapicuaGen y la generación de esta memoria.

[1] Rajiv D. Banker and Sandra A. Slaughter, "A Field Study of Scale Economies in Software Maintenance", *Management Science*, vol. 43, no. 12, pp. 1709-1725, 1997.

[2] Mario Piattini Velthuis and Javier Garzás Parra, *Fábricas de software: Experiencias, tecnologías y organización*. España: Ra-Ma Editorial, 2010.

[3] Krzysztof Czarneck, *Generative Programming Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models*. Ilmenau, Alemania: Department of Computer Science and Automation, Technical University of Ilmenau, 1998.

[4] Thomas J. Mowbray and Raphael C. Malveau, *Software Architect Bootcamp*. Estados Unidos de América: Prentice Hall, 2000.

[5] Juan Garbajosa Sopeña and Francisco Javier Soriano Camino, "Tenologías software orientadas a servicios," Madrid, vigilancia tecnológica 2008.

[6] Jack Greenfield and Keith Short, "Software factories: assembling applications with patterns, models, frameworks and tools," in *Object-Oriented Programming, Systems, Languages & Applications*, 2003, pp. 16-27.

[7] W. Clay Richardson, Donald Avondolio, Scot Schrager, Mark W. Mitchell, and Jeff Scanlon, *Professional Java JDK 6 Edition*. Estados Unidos de América: Wrox, 2007.

[8] David Garlan and Mary Shaw, "An Introduction to Software Architecture", *Advances in Software Engineering*, vol. I, 1993.

[9] Phil Laplante, Robert R. Hoffman, and Gary Klein, "Antipatterns in the Creation of Intelligent Systems", *Intelligent Systems*, vol. 22, no. 1, pp. 91-95, 2007.

[10] Stephen J. Mellor, Kendall Scott, Axel Uhl, and Dirk Weise, *MDA Distilled*. Estados Unidos de América: Addison-Wesley Professional, 2004.

[11] José Luis Bautista Martín, *MDA, promesa y realidad*. Guadalajara, Mexico, 2014.

[12] Jack Herrington, *Code Generation in Action*. Estados Unidos de América: Manning Publications Co., 2003.

[13] Mark S. Merkow and Lakshmikanth Raghavan, *Secure and Resilient Software*.: CRC Press, 2010.

[14] Sebastian Priolo, *Ruby, manual del programador*. Argentina: Users, 2008.

[15] José Luis Bautista Martín. (2014, septiembre) Regla N°5 de la Ingeniería de Software: No te repitas. [Online]. http://desdelashorasextras.blogspot.mx/2014/09/regla-n-5-no-te-repitas.html

[16] José Luis Bautista Martín. (2014, septiembre) Regla N°3 de la Ingeniería de Software: Se va a mantener. [Online]. http://desdelashorasextras.blogspot.mx/2014/09/regla-n3-de-la-ingeniera-de-software-se.html

[17] José Luis Bautista Martín. (2014, Agosto) Regla N°1 de la Ingeniería de Software: Va a cambiar. [Online]. http://desdelashorasextras.blogspot.mx/2014/08/regla-n1-de-la-ingeniera-de-software-va.html

[18] Terrence W. Pratt and Marvin V. Zelkowitz, *Programming Languages: Design and Implementation*. Estados Unidos de América: prentice hall, 1998.

[19] Rubén Heradio Gil, *Metodología de desarrollo de software basada en el paradigma generativo. Realización mediante la transformación de ejemplares*. Madrid, España: Departamento de Ingeniería de Software y Sistemas Informáticos de la UNED, 2007.

[20] César de la Torre, Unai Zorrilla, Ramos Miguel Angel, and Calvarro Javier, *Guía de Arquitectura N-Capas orientada al Dominio con.NET 4.0*. España: Krasis Consulting S.L., 2011.

[21] José Luis Bautista Martín. (2014, septiembre) El problema del martillo de oro. [Online]. http://desdelashorasextras.blogspot.mx/2014/09/el-problema-del-martillo-de-oro.html

[22] Dominic Betts et al., *Developer's Guide to Microsoft Enterprise Library*, 2nd ed. Estados Unidos de América: Microsoft Corporation, 2013.

[23] Microsoft Corporation. (2006, junio) Arquitectura de aplicaciones de.NET: Diseño de aplicaciones y servicios. [Online]. https://msdn.microsoft.com/es-es/library/ms954595.aspx

[24] Giovanni Scerra. (2014) 10 Golden Rules Of Good OOP. [Online]. http://www.codeproject.com/Articles/768052/Golden-Rules-Of-Good-OOP

GLOSARIO DE TÉRMINOS

Se incluye un glosario de términos, siglas, abreviaturas y acrónimos usados en este documento junto con sus respectivos significados.

.NET	.NET o Microsoft.NET hace referencia a una colección de tecnologías de desarrollo de software creadas por Microsoft. Abarca varios lenguajes de programación, un IDE, un conjunto de clases, y un runtime de ejecución. Además permite desarrollar tanto para aplicaciones Windows, como Web y otras.
Activity (Android)	En Android una actividad es, salvo excepciones, cada una de las pantallas de las que consta nuestra aplicación móvil. Una característica es que la lógica de negocio y la presentación están claramente separadas.
Administrador de configuraciones (Visual Studio)	Menú de Visual Studio que nos permite elegir el tipo de configuración del compilado (`Release` o `Debug`) y la arquitectura (x86, x64 o `AnyCPU`).
Android	Sistema operativo de Google para dispositivos móviles, ampliamente usado en el mercado por hardware de heterodoxas características.
Android Studio	IDE de Google para el desarrollo de aplicaciones para Android.
AndroidManifiest	Archivo de configuración de una aplicación de Android. Define elementos como las actividades o los permisos.
Bundler	Versátil gestor de paquetes y dependencias para Ruby.
C#	Es uno de los lenguajes de programación de .NET, el principal realmente. Está basado en C++, con gran parecido a Java (del cual se puede considerar su competidor).
Característica (de un sistema)	Entiéndase como característica de un sistema, cada uno de los aspectos (que a su vez pueden dividirse en otros más pequeños), que debe implementar nuestro software. Algunas características pueden excluir a otras (como por ejemplo el motor de base de

	datos en el que vamos a guardar la información, en el caso que sólo debamos tener uno) o pueden agregarse para trabajar como módulos independientes de nuestro sistema. Por ejemplo podemos tener varias interfaces gráficas, como una interfaz para Windows, Mac, o dispositivos móviles (Android, iOS,…).
Code Snippets	Son pequeñas partes reusables de código fuente. Algunos IDE permiten pequeñas personalizaciones de dichas partes.
Control de versiones	Los software controladores de versiones, nos permiten gestionar el código fuente de un proyecto, recuperar versiones anteriores del código, comprobar cambios y editar el mismo código de forma adecuada dentro de un equipo de trabajo. Populares software de este tipo son Subversion (svn) y Git.
Dominio (del problema)	Conjunto de reglas, elementos participantes, y la relación entre ellos, de un problema en concreto.
Empresa	Organización que persigue unos objetivos y en la que generalmente hay una transformación de un tipo de bien en otro, con el cual obtener un beneficio.
Framework	Conjunto de librerías, prácticas y conceptos para resolver un problema en particular en el mundo del software.
Generación de código automática	Actividades que pretenden automatizar la creación de la máxima cantidad posible de código fuente de un sistema.
Git	Software de control de versión reciente, bastante poderoso y práctico, además de descentralizado; más complicado de manejar que Subversion pero mucho más versátil y potente.
GitHub	Almacén en la nube de repositorios de Git muy popular, con opciones gratuitas de almacenaje, además de diversas opciones de pago.
GUI	Interfaz gráfica de usuario, es la parte con la que interactúa el usuario de un sistema.
Java	Lenguaje de programación multiplataforma, bastante popular actualmente, creado por Sun y ahora propiedad de Oracle.
JSON	JavaScript Object Notation. Es un formato de intercambio de información, mucho más ligero que XML, pero conservando una gran facilidad de lectura e interpretación tanto para las computadoras como para los seres humanos.

LGPL	*GNU Lesser General Public License*. Licencia de código libre menos restrictiva que la licencia GPL tradicional, al permitir convivir con software propietario.
Mercado	Colección de circunstancias externas en la que tiene que desenvolverse nuestra empresa, incluyendo interacción con otras empresas competidoras o que le influyen de alguna forma. Incluyen también la relación con sus clientes.
Mixin (Ruby)	Característica de Ruby que permite agregar funcionalidad de diversas fuentes a una clase, sin incurrir por ello en herencia múltiple.
Model-driven architecture (MDA)	Propuesta de diseño de software, enfocada en la construcción y transformación de modelos.
Modelo	Representación del dominio de un problema, que es lo suficientemente abstracta para poder ser manejada por un sistema, al igual que lo sufrientemente concisa para resultar útil.
Negocio	Colección de reglas que debe cumplir nuestro software, con respecto al problema a resolver en el ámbito de la empresa.
Northwind	Base de datos de Microsoft que emula a la de una empresa de nombre 'Northwind', incluyendo conceptos empresariales comunes como empleados, ventas, y territorios.
Open Source	Son sistemas software en los que el código fuente está disponible para su modificación y distribución libremente.
RESTful	Representational State Transfer. Es un mecanismo para consumir servicios a través de verbos HTTP (GET, POST, PUT, DELETE), de una forma mucho más sencilla y ligera que con XML y SOAP.
Ruby	Lenguaje de programación interpretado, reflexivo y orientado a objetos, creado a mediados de los 90. Es bastante fácil de aprender y poderoso en sus funciones; puede ampliarse a través de un enorme repositorio de módulos, que reciben el nombre de "gemas".
RubyGems	Repositorio público y amplio con multitud de gemas de Ruby. Cualquier desarrollador puede subir sus gemas en este repositorio y posibilitar que otros desarrolladores las utilicen de forma sencilla.
Servicio Web	Exposición de cierta funcionalidad de un sistema a través

	de un servidor web y unos protocolos establecidos.
Sistema	Conjunto de elementos heterodoxos, incluyendo los de tipo software, que trabajan juntos para resolver una serie de tareas.
Software	Elementos desarrollados en algún lenguaje de programación que resuelve necesidades dentro de una empresa, interactuando con otros componentes (software o no), para construir sistemas completos encargados de determinadas tareas dentro de la institución.
SQL	Structured Query Language. Es un lenguaje declarativo para realizar consultas a base de datos. Al ser un estándar (en teoría), éste es independiente del motor que lo implemente y del sistema que lo consuma.
SQL Server	Popular motor de base de datos ofrecido por Microsoft, con varias versiones, algunas gratuitas.
Text Template Transformation Toolkit	Plantillas de Visual Studio con la capacidad de generar código automáticamente y directamente dentro del IDE, sin que sea necesario ningún proceso adicional.
Windows.Forms	Hace referencia a la API de .NET para generar programa de escritorio de Windows.
XML	Esquema sencillo para intercambiar información entre sistemas (entre otros usos) independiente de la arquitectura de los elementos participantes.

Anexo I. CÓMO OBTENER E INSTALAR CAPICUAGEN

CapicuaGen está liberado como open source[8] y puede obtenerse desde GitHub o desde RubyGems. El motivo de este tipo de licencia es que su código pueda ser analizado y modificado según las necesidades de cada empresa en particular, ayudando a los participantes de la creación de un sistema a realizar su trabajo de una forma más efectiva y sencilla.

Si bien CapicuaGen es software libre, no se restringe el tipo de licencia a aplicar para el software generado por él. Igualmente si se crean nuevos generadores de características, se pueden licenciar de la forma que más convenga a la empresa o al desarrollo en particular.

Si se modifica el código de CapicuaGen en sí, deben licenciarse los cambios como LGPL, aunque convendría notificar los cambios y considerar un *merge* con la línea principal de desarrollo de CapicuaGen.

Los generadores de CapicuaGen esta distribuidos en módulos y pueden obtenerse desde GitHub en:

https:// Github.com/jbautistamartin

Y como gemas de Ruby en:

https://rubygems.org/profiles/jbautista

CapicuaGen es una colección de gemas desarrolladas en **Ruby, versión 2.2** (exactamente en *"2.2.3p173 (2015-08-18 revision 51636) [i386-mingw32]"*) por lo que necesitaremos este entorno o uno superior para poder ejecutar los generadores de código.

[8] Usa la licencia LGPL, "GNU Lesser General Public License", menos restrictiva que la licencia GPL

Es importante señalar que CapicuaGen (como la mayoría de gemas de Ruby) es multiplataforma, y puede crear código fuente para cualquier lenguaje al se le haya implementado una característica.

Gemas

Se pueden instalar las gemas de Ruby a través de RubyGems o mediante Bundler, a través de GitHub. Las gemas en cuestión son las siguientes:

- **CapicuaGen**: Núcleo del generador de código.

  https://rubygems.org/gems/CapicuaGen

  https:// Github.com/jbautistamartin/CapicuaGen

- **CapicuaGenMelchior**: Características comunes del generador.

  https://rubygems.org/gems/CapicuaGenMelchior

  https:// Github.com/jbautistamartin/CapicuaGenMelchior

- **CapicuaGenGaspar**: Características para C#.

  https://rubygems.org/gems/CapicuaGenGaspar

  https:// Github.com/jbautistamartin/CapicuaGenGaspar

- **CapicuaGenBalthazar**: Características para Android.

  https://rubygems.org/gems/CapicuaGenBalthazar

  https:// Github.com/jbautistamartin/CapicuaGenBalthazar

- **CapicuaGenEssential**: Clase base que referencia a las anteriores para facilitar la instalación de un entorno funcional.

  https://rubygems.org/gems/CapicuaGenEssential

  https:// Github.com/jbautistamartin/CapicuaGenEssential

Dependencias

Gemas

CapicuaGen requiere adicionalmente la instalación (automática con RubyGems) de las siguientes gemas:

- activesupport >= 2.3.18, ~> 2.3
- nokogiri >= 1.6.7, ~> 1.6
- uuidtools >= 2.1.5, ~> 2.1
- colorize ~> 0.7.7

Otro tipo de dependencias

La característica `CodeMaidCleaner` (para embellecer y formatear código en C#) delega en RunCodeMaidCleaner.exe (incluido en la gema) que requiere tener instalado:

- Microsoft Visual Studio 2010 o superior
- Microsoft Framework .NET 4.0.3 o superior

Instalación de Ruby

Para instalar Ruby, simplemente descargamos el instalador adecuado de la siguiente página y lo ejecutamos:

http://rubyinstaller.org/downloads/

Se recomienda usar una versión de la rama **2.2.X**, y que además sea de **32 bits** porque no todas las gemas funcionan correctamente en la versión de 64 bits.

Instalación de CapicuaGen a través de RubyGems

Una vez instalado Ruby, podemos instalar CapicuaGen, mediante RubyGems, con el siguiente comando:

```
gem install CapicuaGenEssential
```

El resultado de la instalación será el siguiente:

```
>gem install CapicuaGenEssential

Fetching: activesupport-2.3.18.gem (100%)
Successfully installed activesupport-2.3.18
Fetching: mini_portile2-2.0.0.gem (100%)
Successfully installed mini_portile2-2.0.0
Fetching: nokogiri-1.6.7.2-x86-mingw32.gem (100%)
Nokogiri is built with the packaged libraries: libxml2-2.9.2, libxslt-1.1.28, zlib-1.2.8, libiconv-
1.14.
Successfully installed nokogiri-1.6.7.2-x86-mingw32
Fetching: uuidtools-2.1.5.gem (100%)
Successfully installed uuidtools-2.1.5
Fetching: CapicuaGen-0.0.3.gem (100%)
Successfully installed CapicuaGen-0.0.3
Fetching: CapicuaGenMelchior-0.0.3.gem (100%)
Successfully installed CapicuaGenMelchior-0.0.3
Fetching: CapicuaGenBalthazar-0.0.3.gem (100%)
Successfully installed CapicuaGenBalthazar-0.0.3
Fetching: CapicuaGenGaspar-0.0.3.gem (100%)
Successfully installed CapicuaGenGaspar-0.0.3
Fetching: CapicuaGenEssential-0.0.3.gem (100%)
Successfully installed CapicuaGenEssential-0.0.3
Parsing documentation for activesupport-2.3.18
Installing ri documentation for activesupport-2.3.18
Parsing documentation for mini_portile2-2.0.0
Installing ri documentation for mini_portile2-2.0.0
Parsing documentation for nokogiri-1.6.7.2-x86-mingw32
Installing ri documentation for nokogiri-1.6.7.2-x86-mingw32
Parsing documentation for uuidtools-2.1.5
Installing ri documentation for uuidtools-2.1.5
Parsing documentation for CapicuaGen-0.0.3
Installing ri documentation for CapicuaGen-0.0.3
Parsing documentation for CapicuaGenMelchior-0.0.3
Installing ri documentation for CapicuaGenMelchior-0.0.3
Parsing documentation for CapicuaGenBalthazar-0.0.3
Installing ri documentation for CapicuaGenBalthazar-0.0.3
Parsing documentation for CapicuaGenGaspar-0.0.3
Installing ri documentation for CapicuaGenGaspar-0.0.3
Parsing documentation for CapicuaGenEssential-0.0.3
Installing ri documentation for CapicuaGenEssential-0.0.3
Done installing documentation for activesupport, mini_portile2, nokogiri, uuidtools, CapicuaGen,
CapicuaGenMelchior, CapicuaGenBalthazar, CapicuaGenGaspar, CapicuaGenEssential after 57 seconds
```

```
9 gems installed
```

Resultado de pantalla 6. Instalación de CapicuaGenEssential.

Esto nos permitirá tener un ambiente funcional de CapicuaGen. Para validarlo simplemente ejecute CapicuaGen. Se mostrará la ayuda asociada al programa:

```
capicuagen
```

Nos mostrará la siguiente información:

```
Uso: <Script generador> [comandos] [opciones]


Comandos:
 generate : Genera las características configuradas.
 clean : Limpia los archivos generados.
 cleanAndGenerate : Limpia los archivos generados y luego los vuelve a crear.
 example : Genera un ejemplo.
 template : Lista o permite reemplazar plantillas por defecto.


Ejecute '<generator.rb> COMMAND --help' para obtener más ayuda.

 -h, --help Muestra este mensaje.
 --version Muestra la versión.
```

Resultado de pantalla 7. Ayuda de CapicuaGen.

Anexo II. PREPARACIÓN DE AMBIENTE PARA LOS EJEMPLOS

En el presente anexo se revisarán los prerrequisitos necesarios para la correcta ejecución de los ejemplos generados.

Preparación del ambiente para el ejemplo

Para poder ejecutar el ambiente es necesario tener configurado el siguiente software:

- Microsoft Windows, con IIS configurado apropiadamente.
- Microsoft Visual Studio 2010 o superior.
- Android Studio 2.

¿Dónde obtener el software?

- Puede obtener Microsoft SQL Express 2012 en el siguiente enlace:

  https://www.microsoft.com/es-es/download/details.aspx?id=29062

- Si no tiene una versión comercial de Visual Studio puede descargar la versiones *express* desde:

  https://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx

- En el caso de Android Studio puede descargar el software de:

  http://developer.android.com/intl/es/sdk/index.html

- Northwind es un ejemplo de base de datos de Microsoft que puede obtenerlo en :

  https://northwinddatabase.codeplex.com/

2.1 Preparación de la base de datos Northwind

Debe ejecutarse el script Northwind.sql dentro de una instancia SQL. En nuestro caso fue SQL Express2012:

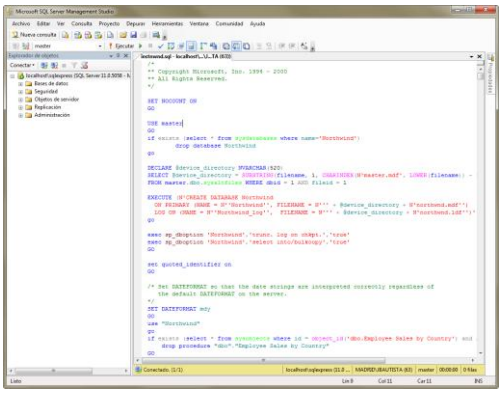


Ilustración 19. Ejecución de Northwind.sql.

Se crearán las bases de datos, tanto la estructura como el contenido:

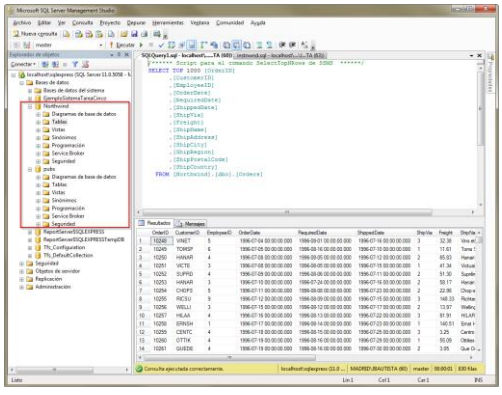


Ilustración 20. Base de datos Northwind.

2.2 Preparación del ejemplo de Android

Para ejecutar el ejemplo de Android, es necesario crear una aplicación en Android Studio 2.0 que lo contenga en interior. Para ello creamos un proyecto nuevo con las siguientes características:

- **Application Name:** NorthWindAndroidExample
- **Company Domain:** capicua.com
- **Proyect location:** <DIR>\NorthWindAndroidExample
- **Minimum SDK:** API15 Android 4.0 (IceCreamSandwich)
- **Package Name**: com.capicua.northWindAndroidExample (Es necesario editarlo y respetar las mayúsculas y minúsculas según están establecidas)
- Seleccionaremos `Empty Activity`.

Una vez creada la aplicación hay que agregar las dependencias necesarias a nuestro proyecto, para lo cual modificamos la sección de dependencias del archivo `build.gradle` (dentro de la carpeta `app`):

```
dependencies {
 compile fileTree(dir: 'libs', include: ['*.jar'])
 testCompile 'junit:junit:4.12'
 compile 'com.android.support:appcompat-v7:22.2.1'
 compile 'com.android.support:recyclerview-v7:22.2.1'
 compile 'com.android.support:cardview-v7:22.2.1'
 compile 'com.google.code.gson:gson:2.6.2'
 compile 'com.google.android.gms:play-services-appindexing:8.1.0'
}
```

Ejemplo 13. Dependencias en archivo build.gradle.

También hay que establecer los siguientes valores para que se usen las versiones de compilación adecuadas (es necesario instalar el SDK versión 22 porque en versiones posteriores el componente `CardView` genera efectos inesperados)

```
compileSdkVersion 22
buildToolsVersion "22.0.1"
```

```
minSdkVersion 14
targetSdkVersion 22
```

Ejemplo 14. Dependencias en archivo build.gradle.

El archivo completo es el siguiente:

```
apply plugin: 'com.android.application'

android {
 compileSdkVersion 22
 buildToolsVersion "22.0.1"

 defaultConfig {
 applicationId "com.capicua.northWindAndroidExample"
 minSdkVersion 14
 targetSdkVersion 22
 versionCode 1
 versionName "1.0"
 }
 buildTypes {
 release {
 minifyEnabled false
 proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
 }
 }
}

dependencies {
 compile fileTree(dir: 'libs', include: ['*.jar'])
 testCompile 'junit:junit:4.12'
 compile 'com.android.support:appcompat-v7:22.2.1'
 compile 'com.android.support:recyclerview-v7:22.2.1'
 compile 'com.android.support:cardview-v7:22.2.1'
 compile 'com.google.code.gson:gson:2.6.2'
 compile 'com.google.android.gms:play-services-appindexing:8.1.0'
}
```

Ejemplo 15. Archivo build.gradle completo.

A continuación es necesario eliminar el archivo `MainActivity.java` (ya que lo crea el ejemplo generador en otro directorio).

También hay que eliminar del código del archivo `AndroidManifiest.xml`, las referencias a la actividad `MainActivity`. Es el siguiente código:

```
<activity android:name=".MainActivity">
 <intent-filter>
 <action android:name="android.intent.action.MAIN"/>
 <category android:name="android.intent.category.LAUNCHER"/>
 </intent-filter>
</activity>
```

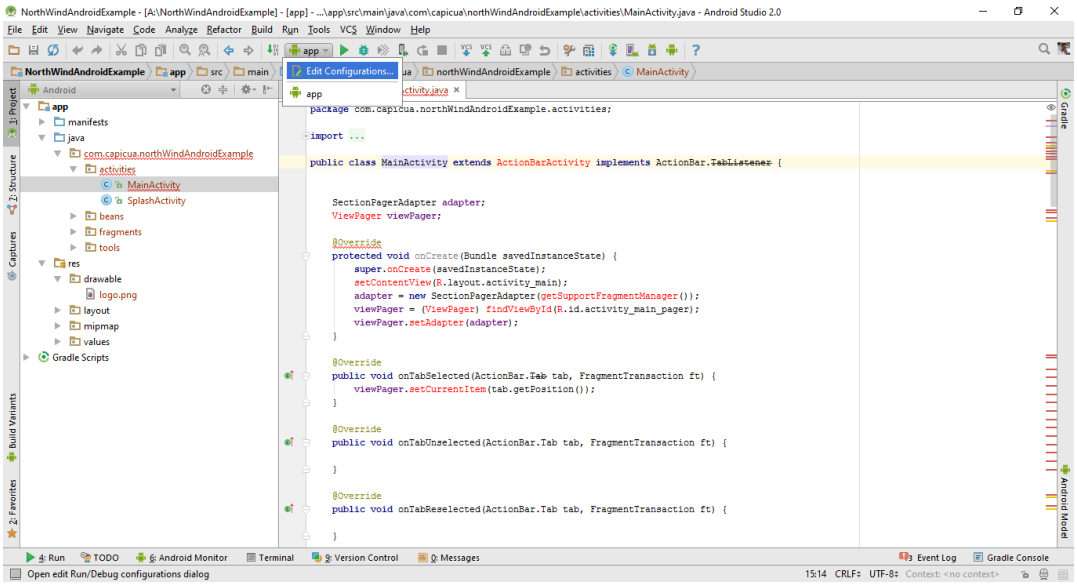
Ejemplo 16. Actividad MAIN in AndroidManifiest.xml.

El archivo manifiest debería quedar así:

```
<?xml version='1.0' encoding='UTF-8'?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.capicua.northWindAndroidExample">
 <application android:allowBackup="true" android:icon="@mipmap/ic_launcher"
android:label="@string/app_name" android:supportsRtl="true" android:theme="@style/AppTheme">
 </application>
</manifest>
```

Ejemplo 17. Archivo AndroidManifiest.xml sin actividades.

Una vez que el código ha sido generado, para que funcione correctamente, es necesario establecer la actividad de inicio dentro de Android:



Seleccionaremos SplashActivity, como actividad de inicio:

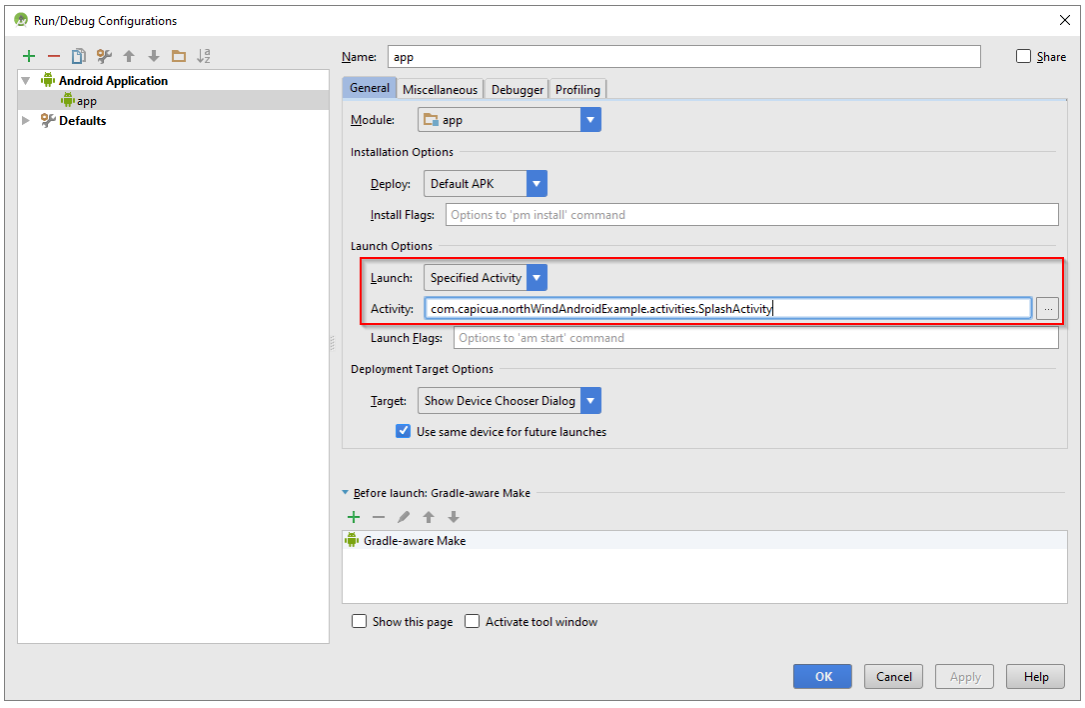


Ilustración 21. Actividad de inicio de Android.

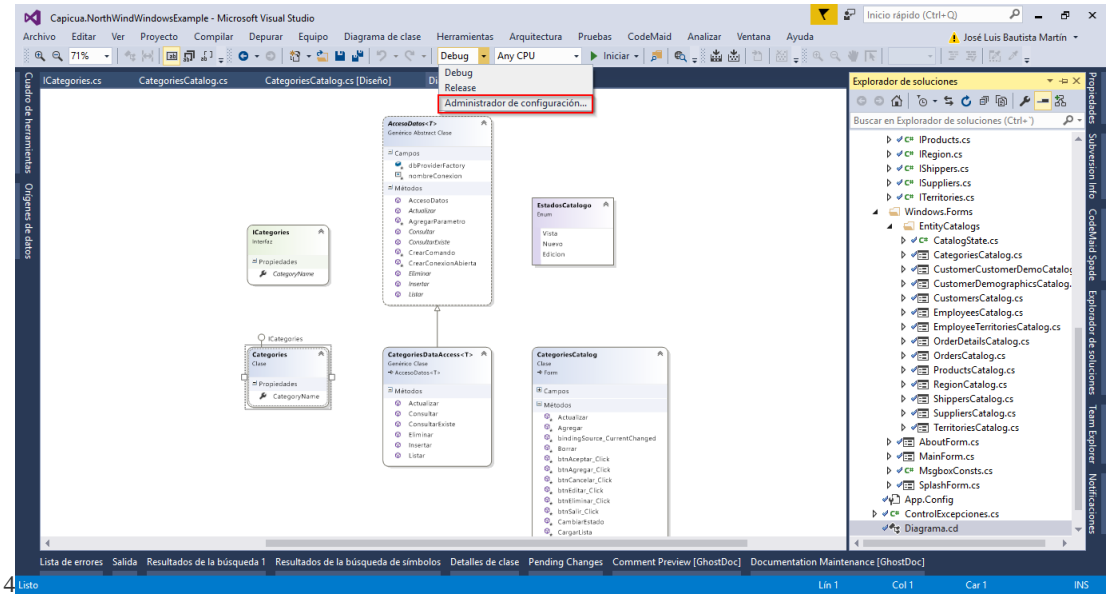
2.3 Preparación del ejemplo para Visual Studio

Es necesario instalar la extensión `CodeMaid` en Visual Studio, la cual nos ayudará a embellecer el código generado, entre otras funcionalidades. Se puede instalar la extensión desde:

http://www.codemaid.net/

Compilación del ejemplo para escritorio

Para compilar el ejemplo necesitamos indicar en el gestor de configuraciones, que compile el proyecto generado (por defecto está demarcada la casilla) y que lo cree como "Any CPU".



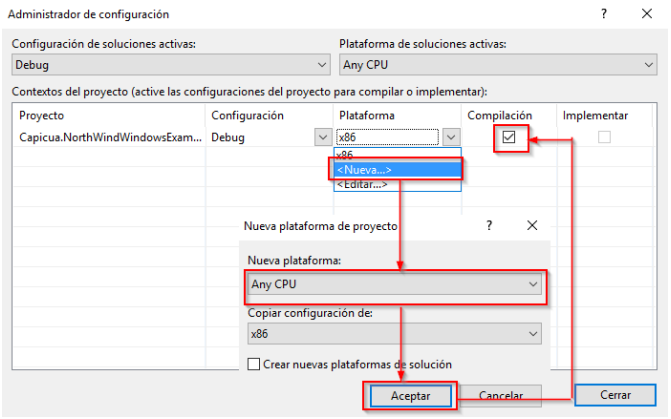


Ilustración 22. Administrador de configuración para Windows.Forms.

Compilación del ejemplo RESTful

Para compilar el ejemplo necesitamos indicar en el administrador de configuraciones que compile el proyecto generado (por defecto está demarcada la casilla), y que lo cree como "Any CPU".

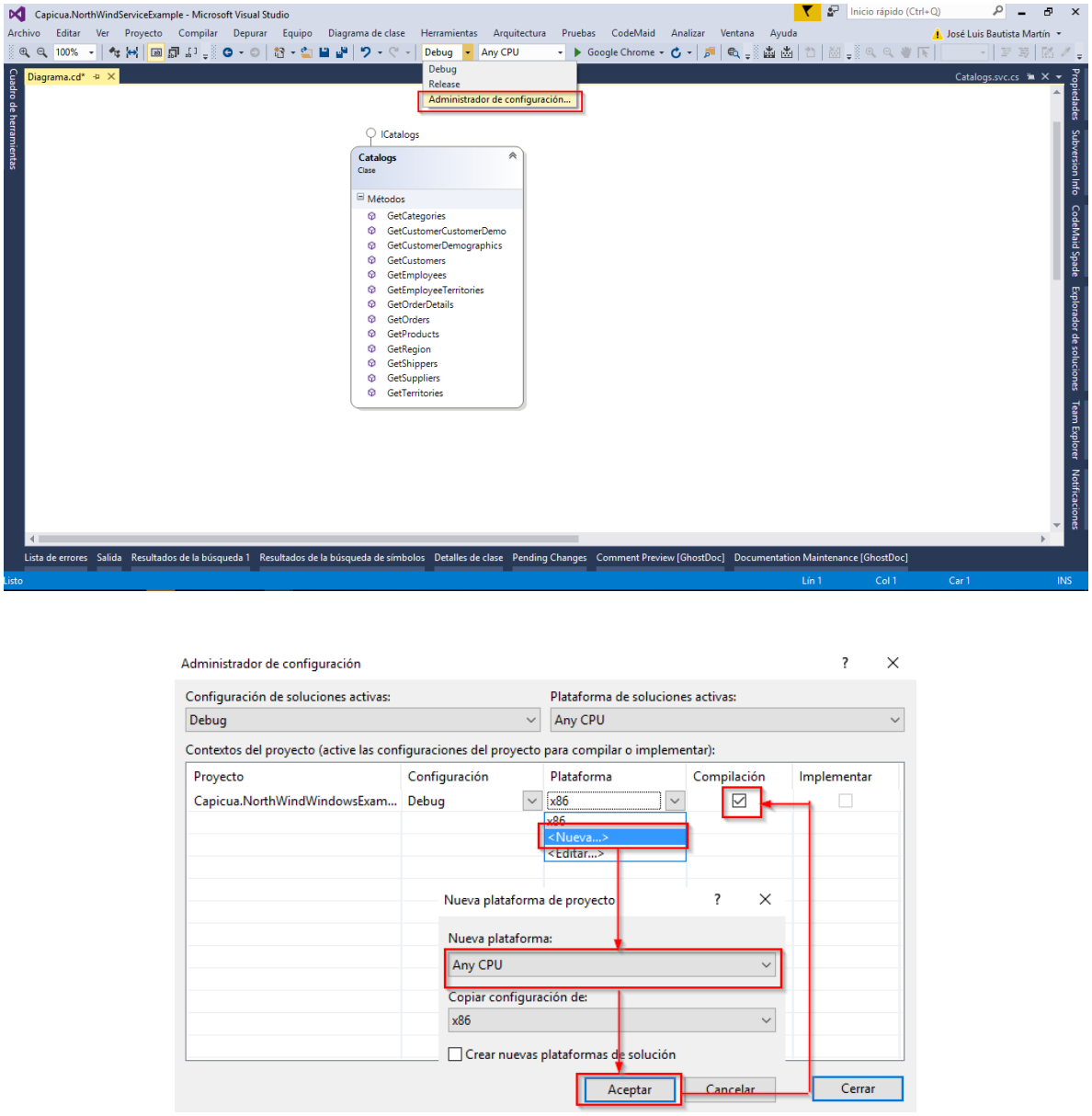


Ilustración 23. Administrador de configuración para RESTful.

Adicionalmente hay que configurar apropiadamente la aplicación web:

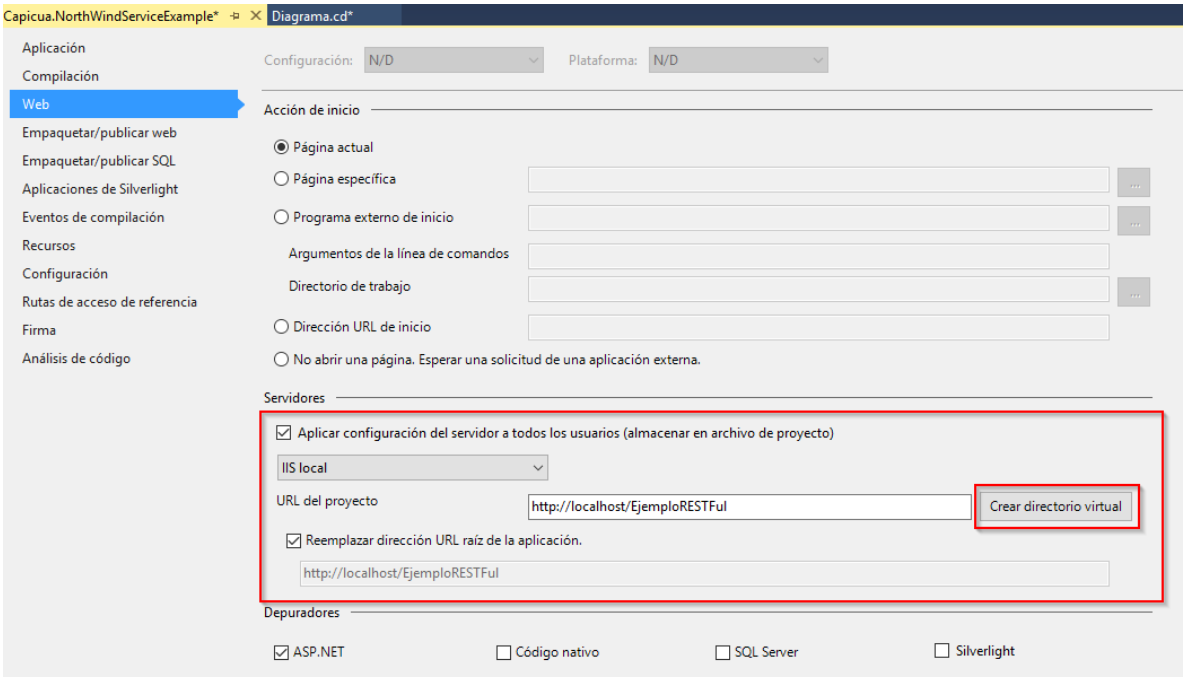


Ilustración 24. Configuración web para el ejemplo RESTful.

Si quisiéramos ver algún catálogo en particular ejecutaríamos:

http://localhost/EjemploRESTful/Services/Catalogs.svc/Categories

Obtendríamos el siguiente resultado:


```
[
 {
 "CategoryName": "Beverages"
 },
 {
 "CategoryName": "Condiments"
 },
 {
 "CategoryName": "Confections"
 },
 {
 "CategoryName": "Dairy Products"
 },
 {
 "CategoryName": "Grains/Cereals"
 },
```

```
 {
 "CategoryName": "Meat/Poultry"
 },
 {
 "CategoryName": "Produce"
 },
 {
 "CategoryName": "Seafood"
 }
]
```


Resultado de pantalla 8. Ejemplo JSON de Categories.

Anexo III. CARACTERÍSTICAS PREEXISTENTES DE CAPICUAGEN

Si bien CapicuaGen es un framework para que las empresas creen sus propios generadores de características y módulos que se ajusten a sus necesidades, se proporcionan algunos generadores previamente establecidos para que sirvan tanto de base, como de ejemplo para futuros proyectos.

A continuación se describen los módulos y características principales incluidas.

3.1 Módulo CapicuaGen

Este módulo contiene el núcleo de CapicuaGen. Todas las clases centrales que se usan en todos los generadores de características se encuentran en esta gema.

- **capicua_gen.rb:** Contiene referencias a las clases principales de CapicuaGen. Si se desea usar CapicuaGen se debe importar este archivo y no cada archivo individual.

- **Feature:** Define la clase base de todos los generadores de características.

- **Generator:** Es el generador central que aglutina todos los generadores de características así como los objetivos y atributos a crear.

- **Target:** Son los objetivos a generar.

- **TemplateFeature:** Clase base (que hereda de Feature) para crear generadores de características basados en templates.

- **Template:** Cada una de las plantillas a usar en los generadores de tipo `TemplateFeature`.

- **TemplateTarget:** Cada uno de los objetivos ligados a un Template.

- **MessageHelper:** Contiene cada uno los mensajes informativos a mostrar dentro de una generación.

- **TemplateHelper**: Clase ayudante para convertir los archivos ERB a los archivos objetivos.

3.2 Modulo CapicuaGenMelchior

Este módulo almacena generadores de características comunes para otros módulos.

- **melchior.rb:** Contiene referencias a las clases principales de CapicuaGenMelchior. Si se desea usar CapicuaGenMelchior se debe importar este archivo y no cada archivo individual.

DataAccess

Contiene generadores relacionados con el acceso a datos.

- **EntitySqlDataAccessFeature:** Analiza un archivo de SQL Server para la creación de una base de datos, buscando las sentencias `Create Table` y expone las tablas como objetos `EntitySchema`, para diversas finalidades.

- **EntitySchema:** Define una entidad con la forma de una estructura con definiciones de campos y diversos atributos.

- **EntityFieldSchema**: Define un campo de una entidad. Tiene atributos relativos al tipo de campo, si es requerido o no, además de su longitud.

- **Mixins:** Permite acceder a la funcionalidad expuesta de las características.

5.4 Módulo CapicuaGenGaspar

CapicuaGenGaspar es un módulo que contiene características generadoras para software en .NET, particularmente en C#.

- **gaspar.rb:** Contiene referencias a las clases principales de CapicuaGenGaspar. Si se desea usar CapicuaGenGaspar se debe importar este archivo y no cada archivo individual.

Business

Contiene los generadores de características relacionados al negocio.

- **CSSqlEntitityInterfaceFeature:** Genera interfaces a entidades de negocio basándose en tablas SQL.

- **CSSqlEntityFeature:** Genera entidades de negocio basándose en tablas SQL.

CodeTransformer

Genera transformaciones de código para archivos previamente existentes, a diferencia de los generadores basados en plantillas que crean el archivo en sí.

- **CodeMaidCleanerFeature**: "Embellece" el código XML y C# generado, usando la extensión de Visual Studio "CodeMaid" (la cual debe estar instalada previamente). Hace el código final más legible.

- **CSHeaderFooterFeature**: Agrega a los archivos generados de C#, una cabecera y pie de página, en donde se pueden agregar comentarios tales como el objetivo del sistema, el modo de uso, la licencia o cualquier información pertinente.

Es posible "exportar" todos los templates y reemplazarlos por los que nosotros necesitemos. En el caso de los `headers` y los `footers` podemos exportarlos con el siguiente comando:

```
>capicuagen template -g CapicuaGenGaspar -t CodeTransformer -f CSHeaderFooter

+ CapicuaGenGaspar, CodeTransformer, CSHeaderFooter ->
'capicua/CapicuaGenGaspar/CodeTransformer/CSHeaderFooter/footer.erb': Creado
+ CapicuaGenGaspar, CodeTransformer, CSHeaderFooter ->
'capicua/CapicuaGenGaspar/CodeTransformer/CSHeaderFooter/header.erb': Creado
```

Ejemplo 18. Exportación de templates.

Si modificamos los archivos exportados con nuestros propios templates, serán reemplazados las cabeceras y los pies de página en nuestros archivos generados.

DataAccess

Contiene generadores de características relativas al acceso a base de datos.

- **CSConnectionProviderFeature:** Proporciona mecanismos para obtener cadenas de conexión a base de datos además los drivers específicos de conexión.

- **CSSqlDataAccessFeature:** Generador para crear las clases de persistencia de las entidades.

GUI

Aglutina las características relacionadas con la generación de interfaces gráficas.

- **CSAboutWindowsFormFeature:** Genera una ventana de Windows.Forms "Acerca de…"

- **CSCatalogWindowsFormFeature:** Genera ventanas de Windows.Forms para administrar las entidades a través de catálogos (consultas, altas, bajas, modificaciones).

- **CSMDIWindowsFormFeature:** Genera una ventana principal de Windows.Forms de estilo MDI.

- **CSSplashWindowsFormFeature:** Genera una ventana de bienvenida.

- **Mixins:** Permite acceder a la funcionalidad expuesta de las características.

Proyect

Contiene características que se encargan de generar archivos de proyectos.

- **CSProyectFeature:** Genera un proyecto y una solución de C#.

- **CSProyectRESTfulFeature:** Genera un proyecto y una solución de C# para exponer servicios con RESTful.

- **CSProyectWindowsForm:** Genera un proyecto y una solución de C# para Windows, agregando las ventanas indicadas por otros generadores de características.

WCFService

Contiene características relativas a servicios que usan la tecnología WCF (Windows Communication Fundation):

- **CSRESTfulCatalogFeature:** Crea un servicio que expone las entidades a través de RESTful.

5.5 Módulo CapicuaGenBalthazar

Contiene características generadoras para dispositivos móviles basados en Android.

- **balthazar.rb:** Contiene referencias a las clases principales de CapicuaGenBalthazar. Si se desea usar CapicuaGenBalthazar se debe importar este archivo y no cada archivo individual.

AndroidLanguage

Contiene características relativas al lenguaje y contexto de programación (Java sobre Android).

- **AndroContextProviderFeature**: Devuelve información del contexto de Android, como el package de la clase R.

Business

Contiene los generadores relacionados al negocio.

- **CSSqlEntitityInterfaceFeature:** Genera interfaces a entidades de negocio basándose en tablas SQL.

- **AndySqlEntityFeature:** Genera entidades de negocio basándose en tablas SQL.

GUI

Aglutina las características relacionadas con la generación de interfaces gráficas.

- **AndyEntityCardViewFragmentFeature:** Generador de clases de tipo `CardView`, pensadas para mostrar entidades en la pantalla del móvil.

- **AndyIcLauncherFeature:** Generador para crear los iconos asociados a la aplicación de Android.

- **AndyMainActivityFeature:** Genera la pantalla principal de la aplicación de Android.

- **AndySplashActivityFeature:** Genera una ventana de bienvenida.

Mixins

Permite acceder a la funcionalidad expuesta de las características.

Web

Características relativas a comunicaciones a través de HTTP y semejantes.

- **AndyWebRequestFeature:** Genera una clase para peticiones HTTP.

5.6 Módulo CapicuaGenEssential

Este módulo contiene referencia a los módulos anteriores y su objetivo es facilitar la instalación de un ambiente funcional de CapicuaGen. Instalando este módulo se instalan automáticamente los anteriores.