Máster Universitario En Investigación En Ingeniería De Software Y Sistemas Informáticos

Desarrollo De Líneas De Producto Software Mediante Un Enfoque Generativo Trabajo de Investigación

Autor
César Hugo Bárzano Cruz

—

2019/2020

APIGENA: Generación de APIs como linea de productos software

César Hugo Bárzano Cruz[1]

UNED,
hugobarzano@gmail.com

Resumen En el ámbito de la web 2.0 y las soluciones cloud, el uso y consumo de servicios o productos software basados en el protocolo HTTP se ha incrementado notablemente. A raíz de la gran demanda de este tipo de productos software, esta publicación propone la implementación de una linea de productos software mediante un enfoque generativo, concretamete de aplicaciones API-REST con el objetivo de generalizar este tipo de aplicaciones para alcanzar una producción en escala. En esta publicación se muestra una implementación capaz de dado un modelo de especifico de dominio definido en formato JSON producir el código fuente y configuración necesarias para realizar operaciones CRUD sobre este tipo de modelos mediante una interfaz API-REST. La implementación que aquí se muestra es capaz de producir código fuente en los siguientes lenguajes: Go, Python, Javascript. Las APIs generadas incluyen todo lo necesario para ser auto-contenidas, es decir, contienen todas las librerías o dependencias necesarias para funcionar correctamente. De manera adicional, esta publicación muestra como el uso de Docker como entorno virtual, facilita el desarrollo y la portabilidad del software aquí propuesto.

Keywords: Lineas de Productos Software, Programación Generativa, API-Rest, Cloud, Golang, Docker

1. Introducción

1.1. Problema

El problema que aquí se propone consiste en ¿ Como generalizar y automatizar el proceso de especificación, documentación y desarrollo de servicios web de naturaleza API-REST? ¿ Es posible abordar el problema para producir APIs en distintos lenguajes de programación?

1.2. Estado del Arte

Actualmente existen herramientas capaces de facilitar y automatizar el proceso de desarrollo de APIs mediante un enfoque generativo. Las herramientas que se proponen a continuación tiene en común el uso de una especificación

OpenAPI[1] que permite describir APIs para ser usadas posteriormente a la hora de producir, consumir y visualizar servicios web. Por otro lado tambien se proponen algunas herramientas las cuales son capaces de dada una especificación de API generar documentación e interfaz de usuario asociada. Todas las herramientas propuestas son de naturaleza Open-Source. Existen otros tipos de especificación que son igual de validos, como por ejemplo:

- API Blueprint
- WSDL
- RAML
- Swagger 1.0
- Swagger 2.0

OpenAPI Generator Permite la generación de bibliotecas de cliente API (SDK), stubs de servidor, documentación y configuración dada una especificación OpenAPI. Para obtener mas información acerca de este software, puede consultarse el repositorio del proyecto openapi-generator.[2]

Swagger Codegen De igual forma que OpenApi Generator, Swagger Codegen permite la generación de clientes, servidores y documentación de API dada una especificación OpenApi, para obtener mas información sobre esta herramienta puede consultarse el repositorio del proyecto.[3]

LucyBot DocGen Permite la generación estática de documentación así como una consola de pruebas para las operaciones especificadas para la API. También permite configurar el dominio y el estilo con el que se produce la salida. Pueden consultarse ejemplos de uso en la documentación de la herramienta Lucybot[4]

DapperDox Permite generar documentación relacionada con la API especificada así como explorador para navegador web permitiendo una mayor comprensión de las operaciones generadas. Esta herramienta tiene la particularidad de poder consumir diversas especificaciones de diversas APIs lo que facilita trabajar con distintos micro-servicios.

Widdershins Genera documentación estática de la API especificada así como vistas HTML para probar métodos, parámetros de los métodos y respuestas de los métodos. La peculiaridad de Widdershins[5] reside en que es capaz de producir código fuente de ejemplo en distintos lenguajes como por ejemplo:

- Ruby
- Python
- JavaScript
- Java

1.3. Solución

Tras el análisis realizado sobre las actuales herramientas capaces de satisfacer en mayor o menor medida el problema propuesto, se aprecia una dependencia directa en todas ellas a la especificación de la API, sea en un formato o en otro, por ello, en este generador se pretende ir un paso mas allá y dado un modelo especificó de dominio en formato JSON:

```
{
    "type":"Apple",
    "name": "Golden",
    "price":1.5,
    "inStock":true
}
```

producir dicha especificación en formato OpenApi o Swagger 2.0 para posteriormente ser aprovechada por el código fuente generado para satisfacer operaciones CRUD sobre el modelo de entrada.

El generador aquí propuesto es capaz de producir código fuente en los siguientes lenguajes de programación:

- Go
- Python
- JavaScript

Debido a que el generador esta implementado en GO, esto facilitar la generación de modelos de datos complejos de manera nativa. En el caso de Python y JavaScript, el código fuente generado generaliza el tratamiento de datos y no define un modelo concreto si no que utiliza la entidad definida en la especificación. De igual forma, el código fuente generado para Python y JavaScript no define rutas del tipo /api o /api/object si no que las rutas de la API son inferidas de la especificación generada. Este enfoque generativo proporciona de manera adicional una interfaz de usuario donde consultar la documentación del API generada, así como una consola para ejecutar operaciones de creación, consulta, borrado y actualización, incluyendo ejemplos de las peticiones a realizar y de las posibles respuestas obtenidas. Las APIs generadas son de carácter auto-contenido, es decir junto a la especificación y al código fuente se produce la configuración necesaria para poder instalar las librerías o dependencias que cada tecnología necesita para ejecutar su APIs.

Como parte adicional, el generador incluye una imagen Docker para facilitar la potabilidad, ejecución y pruebas de las APIs generadas. En las siguientes secciones se muestran detalles de implementación, un caso de uso a modo de ejemplo y los posibles trabajos futuros que pueden resultar del aquí propuesto.

2. Implementación

El generador que aquí se plantea ha sido desarrollado utilizando GO como lenguaje de programación. De este lenguaje se aprovecha el uso de interfaces y su motor de plantillas que facilita en gran medida la generación del código fuente.

2.1. Generator Interface

En esta sección se presenta la interfaz definida para el generador de APIs.

```
// Generator interface definition
type Generator interface {
    Init() Generator
    WithName(name string) Generator
    WithPort(port int) Generator
    WithInputSpec(spec interface{}) Generator
    WithOutputPath(path string) Generator
    Generate()
}
```

La estrategia a seguir es que cada uno de los lenguajes de programación soportados por el generador implementa todas las funciones definidas en la interfaz, caracterizando y adecuando cada caso según las propiedades de dicha tecnología. De manera general, el comportamiento esperado a implementar por cada una de las funciones que describen la interfaz es:

- **Init() Generator** ⇒ Inicializa las estructuras de datos internas del generador.
- **WithName(name string) Generator** Establece el nombre de la API. Este valor es utilizado para nombrar las entidades, los modelos de datos y las rutas de la API.
- **WithPort(port int) Generator** ⇒ Establece el puerto de servicio donde la API será expuesta.
- **WithInputSpec(spec interface) Generator** ⇒ Establece la especificación del modelo de negocio usado como entrada para definir la API. Como se indicó anteriormente, este modelo de negocio especifico de dominio se presenta en formato JSON. En la definición de este método se ha optado por definir utilizando la primitiva interface para soportar a futuro distintos tipos de especificación o DSL. Esta función extrae y transforma la información del modelo de datos para su posterior generación.
- **WithOutputPath(path string) Generator** ⇒ Establece la ruta de salida y crea la jerarquía de directorios donde es producido el código fuente. Dicha jerarquía es variable en función de la tecnológia que se quiere producir como salida.
- **Generate()** ⇒ Produce el código fuente mediante llamadas al motor de plantillas correspondiente a la tecnología que se esté implementando.

Las diversas implementaciones de la interfaz para cada una de las tecnologías soportadas son las siguientes:

- Golang Generator
- Python Generator
- JavaScript Generator

De esta forma se pone de manifiesto que la integración en el generador de nuevas tecnologías es trivial, es decir, una nueva tecnología supone una nueva implementación de la interfaz aquí propuesta.

2.2. Template Engine

Otra de las ventajas que aporta GO como lenguaje de programación a la hora de implementar software generativo es su motor de plantillas, ya que permite producir ficheros desde distintos tipos de estructuras de datos e incluso añadir lógica adicional (como condicionales o iteradores) dentro de la propia plantilla. En este proyecto se usan los siguientes tipos de plantillas, ordenadas de menor a mayor complejidad:

- **Plantilla Constante** ⇒ Se caracteriza por ser el tipo base de plantilla en la que no se realiza ningún tipo de sustitución o transformación.
- **Plantilla Simple** ⇒ Se caracteriza por ser el tipo de plantilla básico en la que se realiza la sustitución de un tipo de dato básico (escalar), por ejemplo una cadena de texto o un entero.
- **Plantilla Intermedia** ⇒ Se caracteriza por usar estructuras de datos definidas en la librería estándar del lenguaje como pueden ser los mapas clave:valor o las listas de elementos.
- **Plantilla Compleja** ⇒ Se caracteriza por usar estructuras de datos complejas para producir resultados, por ejemplo:

```
1
2     tpl, err := template.New("Spec").Parse(SpecTemplate)
3     var output bytes.Buffer
4
5     input := ApiSpec{
6         Title:      getTitle(name),
7         ApiName:    getApiName(name),
8         ModelName: getModelName(name),
9         ApiPkg:     pkg,
10        ModelData: string(generateModelProperties(data)),
11    }
12
13    err = tpl.Execute(&output, input)
14    output.Bytes()
```

Este ejemplo muestra como la estructura de datos compleja **ApiSpec** es utilizada para ejecutar la plantilla **SpecTemplate** y expresar su resultado en forma de Array de bytes. La plantilla SpecTemplate puede ser consultada en el repositorio del proyecto.

3. Caso de Uso

A modo de ejemplos y con el objetivo de mostrar la capacidad generativa de este proyecto sobre las distintas tecnologías soportadas, se ha redactado un caso de uso que puede ser consultado en el README del repositorio del proyecto.

4. Trabajos Futuros

Como trabajos futuros o posibles extensiones que pueden alcanzarse tomando como referencia este proyecto, se destacan los siguientes puntos.

4.1. Tecnologías Soportadas

Se propone como mejora el soporte a mas tecnologías para la generación de APIs. Estas tecnologías podrían ser por ejemplo Ruby on Rails, PHP, Perl o Scala. La idea consiste en implementar la interfaz del generador para estas tecnologías y la integración de los módulos correspondientes en el motor de plantillas.

4.2. Especificaciones de API

Como se ha visto, este proyecto soportar como entrada modelos de negocio (específicos de dominio) en formato JSON y se propone como mejora la generalización de esta entrada para soportar distintos tipos de formatos como por ejemplo XML o YML.

Por otra parte, también se propone mejorar la generación de la especificación de API. Actualmente el tipo de especificación que produce este proyecto es del tipo Swagger 2.0 y por ello se propone generar especificaciones de otras naturalezas como API Blueprint, WSDL o RAML.

Referencias

1. S. Software, What is openapi?
   URL https://swagger.io/docs/specification/about/
2. S. Software, Openapitools/openapi-generator.
   URL https://github.com/openapitools/openapi-generator
3. S. Software, swagger-api/swagger-codegen.
   URL https://github.com/swagger-api/swagger-codegen
4. L. Inc, Lucybot docs.
   URL http://docs.lucybot.com/Introduction
5. Mermade, Mermade/widdershins.
   URL https://github.com/mermade/widdershins