Una Aproximación MDA para la Construcción de Componentes COTSgets en Aplicaciones Web

José A. Asensio, Nicolás Padilla, Javier Criado, and Luis Iribarne

Applied Computing Group, University of Almería, Spain
{jacortes, npadilla, javi.criado, luis.iribarne}@ual.es

Resumen – Actualmente, existe una tendencia al desarrollo de aplicaciones web. Muchas de estas aplicaciones se construyen en base a componentes reutilizables, lo que influye considerablemente en el tiempo de desarrollo. En este contexto se enmarca nuestra propuesta. El artículo presenta una solución basada en la ingeniería dirigida por modelos (MDE) para agilizar y facilitar a los desarrolladores la implementación de un tipo de componentes web (llamados COTSgets). Nuestra propuesta consiste en la generación automática de la implementación de estos componentes, en lo que a su estructura y funcionalidad básica se refiere, a partir de un modelo que describe su especificación y mediante la utilización de una transformación modelo-a-texto (M2T). Para dicha implementación se ha seleccionado la incipiente tecnología Polymer.

Palabras clave: Ingeniería Dirigida por Modelos (MDE), Transformación de Modelo a Texto (M2T), Componentes COTSgets, COScore, Polymer.

1. Introducción

Una de las principales técnicas de la ingeniería dirigida por modelos (MDE) consiste en la generación de código mediante transformaciones modelo-a-texto (M2T). Se utilizan, por ejemplo, para la generación de IUs basadas en Android [1]. En otros casos, se hace uso de un lenguaje específico de dominio (DSL) para la generación de aplicaciones web [2]. Esta generación automática de código permite obtener diversos beneficios relacionados con el coste y el tiempo de desarrollo, así como la reducción de la tasa de errores en la implementación.

Nuestra investigación se centra en el desarrollo de una infraestructura basada en servicios Web, denominada COScore [3], que da soporte a aplicaciones cliente multiplataforma. La construcción de dichas aplicaciones se realiza actualmente de forma manual a partir de componentes de granularidad gruesa llamados COTSgets. Teniendo en cuenta los beneficios indicados anteriormente, en este artículo se propone la generación automática de la implementación de estos componentes para el caso de aplicaciones web.

La Figura 1 muestra los elementos que participan en este proceso. El artefacto principal es una transformación M2T que toma como entradas el modelo que describe un componente así como información adicional que necesita dicho componente para ser desplegado en la infraestructura COScore y funcionar

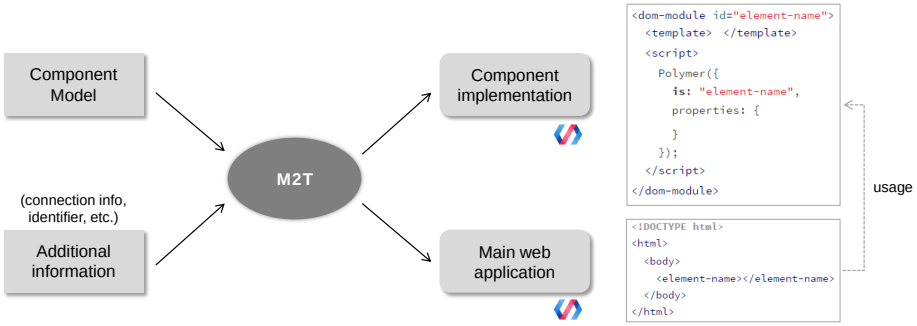

Figura 1. Propuesta para la construcción automática de COTSgets

correctamente. Como salidas, la transformación generará tanto el código correspondiente a la implementación del componente, como el código necesario para que la aplicación web principal pueda hacer uso de dicho componente.

2. Propuesta

El punto de partida de la propuesta es el metamodelo de un componente COTSget [3]. Su estructura está organizada en cuatro secciones: a) información de marketing, b) información de empaquetado, c) información extra-funcional, que permite definir propiedades no funcionales del componente, y d) una sección funcional, que incluye la lógica de negocio del componente (separando la funcionalidad visible al exterior de la funcionalidad interna), y la gestión de eventos de interacción y de comunicación con otros componentes. Para describir la comunicación con otros componentes, la sección funcional define dos conjuntos de interfaces: proporcionadas, para permitir a un componente la recepción de peticiones de servicio; y requeridas, para permitir el envío de estas peticiones a otros componentes. El lenguaje utilizado para la definición de ambos tipo de interfaces es WSDL 1.1 (Web Services Description Language).

A partir de la estructura de un componente COTSget descrita anteriormente, la propuesta consiste en generar automáticamente una implementación de estos componentes para un entorno web. Dicha implementación generará el código de su funcionalidad básica, como puede ser el que corresponde a su inicialización o la comunicación con otros componentes, así como el código que implementa la propia estructura del componente, mientras que la lógica de negocio es labor de los desarrolladores. Para poder alcanzar este objetivo se usarán técnicas de MDE para desarrollar una transformación M2T. La transformación se definirá usando Acceleo [4], una de las implementaciones más utilizada del estándar MOF Model to Text Language (MTL) de OMG, mientras que la implementación de los componentes web hará uso de Polymer (HTML5, CSS3 y JavaScript) [5].

La implementación resultante de la transformación se organizará en una serie de carpetas y archivos. El Listado 1 muestra el archivo principal de un componente (*index.html*). La línea 1 se utiliza para importar la librería Polymer. A partir

de la línea 2 (etiqueta `dom-module`), aparece la definición del componente. Lo primero que se define es la hoja de estilos propia del componente (línea 3) y el contenido que se mostrará al usuario, dentro de la etiqueta `template` (línea 4), si el componente dispone de interfaz de usuario. Las líneas 5–8, se utilizan para construir toda la estructura de archivos de un componente (`InteractionInterface`, `CoreContent`, `InteractionContent` y `ControllerInterface`), descritos debajo. En las líneas 10–22, se realiza la creación del elemento y su registro, con sus propiedades y las funciones que implementan su ciclo de vida. Por último, a partir de la línea 24, se hace referencia a otras librerías que usa el componente. En este caso, sólo se ha incluido la librería `Socket.io`, utilizada para la comunicación entre componentes mediante WebSockets.

En el Listado 2 se identifican los detalles más relevantes de los archivos asociados a un componente. La lógica de negocio de un componente se define en los archivos *content/interaction/interaction.js* y *content/core/core.js* mediante funciones que siguen la estructura del Listado 2a. Las funciones que definen la gestión de los eventos de interacción con el usuario (archivo *interface/interaction/interaction.js*) también siguen dicha estructura. Por otro lado, el archivo *interface/controller/input.js* contiene las funciones que implementan los puertos de entrada del componente (su estructura se corresponde con la mostrada en el Listado 2b). En la línea 1 de dicho archivo, junto al identificador del componente, se especifica el tipo de interfaz (`PI`). El archivo *interface/controller/output.js* contiene las funciones correspondientes a los puertos de salida que, como se puede ver en el Listado 2c, únicamente realizan la emisión de la información hacia el puerto de entrada de otro componente. Igual que con la entrada, el tipo de interfaz se especifica junto al identificador del componente. Finalmente, el archivo *interface/controller/connection.js* contiene las funciones que hacen posible la comunicación de los componentes y que son comunes a todos ellos. A modo ilustrativo, el Listado 2d muestra alguna de estas funciones, como `connect`, que permite establecer la conexión para el envío y recepción de información, o la función `emit` para el envío de información.

```
1  <link rel="import" href="../../polymer/polymer.html">
2  <dom-module id="[component-id]">
3    <!-- style references -->
4    <template> <!-- component interface --> </template>
5    <script src="./src/component.js"></script>
6    <script src="./src/content/core/core.js"></script>
7    <script src="./src/content/interaction/interaction.js"></script>
8    ...
9    <script>
10     Polymer({
11       is: '[component-id]',
12       properties: {
13         user_id: { type: String },
14         nodejs_url: { type: String },
15         component_name: { type: String },
16         component_alias: { type: String },
17         component_instance: { type: String } },
18       ...
19       ready: function() {
20         var _myComponent = this;
21         _myComponent.init(_myComponent, []);
22         _myComponent.connect(_myComponent, []); } });
23   </script>
24   <!-- script references -->
25   <script src="http://cdn.socket.io/socket.io-1.4.5.js"></script>
26 </dom-module>
```

Listado 1. Contenido del archivo *index.html* de un componente.

a)
```
[component-id].[operation-id] = function(_myComponent, _parameters) {
  // function parameters
  var [parameter-id] = _parameters[0]; ...
  // function implementation
}
```

b)
```
[component-id].PI.[input-id] = function(_myComponent, _parameters) {
  // function parameters
  var [parameter-id] = _parameters[0]; ...
  // function implementation
}
```

c)
```
[component-id].RI.[output-id] = function(_myComponent, _parameters) {
  // function parameters
  var [parameter-id] = _parameters[0]; ...
  // function implementation
  var portName = '[output-operation-id]';
  var data = [];  data[0] = [parameter-id]; ...
  var dataSchema = null;
  _myComponent.emit(_myComponent, [portName, data, dataSchema]);
}
```

d)
```
[component-id].connect = function(_myComponent, _parameters) {
  // function implementation
  if(_myComponent.nodejs_url != null) {
    var eventName = 'connect';
    var event = _myComponent.getEvent(_myComponent, [eventName]);
    _myComponent.dataStore.websocket = io.connect(_myComponent.nodejs_url);
    _myComponent.dataStore.websocket.on('connect', event);
    _myComponent.registerInputs(_myComponent, []);  }
}
[component-id].emit = function(_myComponent, _parameters) {
  // function parameters
  var portName = _params[0], data = _params[1], dataSchema = _params[2];
  // function implementation
  _myComponent.dataStore.websocket.emit('send', _myComponent.user_id,
   _myComponent.component_alias, _myComponent.component_instance, portName,
   data, dataSchema);
}
```

Listado 2. Estructura de algunas funciones que definen un componente

3. Conclusiones y Trabajo Futuro

En este artículo hemos presentado una propuesta para la generación automática de componentes COTSgets para la plataforma web utilizando una transformación M2T. Un objetivo futuro es la obtención del modelo de componente a partir de una transformación T2M (texto-a-modelo) para permitir la refactorización y facilitar el desarrollo de componentes. También, se pretende utilizar la transformación M2T para generar automáticamente la documentación del código.

Agradecimientos: Este trabajo ha sido financiado con fondos del Ministerio de Economía y Competitividad (MINECO) dentro del Proyecto TIN2013-41576-R.

Referencias

1. Lachgar, M., and Abdali, A.: Generating Android Graphical User Interfaces using an MDA Approach. Third IEEE International Colloquium in Information Science and Technology, CIST 2014, Tetouan, Morocco, October 20-22, pp. 80–85 (2014)
2. Morales, Z., et al.: A Baseline Domain Specific Language Proposal for Model-Driven Web Engineering Code Generation. Computational Science and Its Applications - ICCSA 2016, Part V, LNCS 9790, pp. 50—59 (2016)
3. Vallecillos, J., Criado, J., Padilla, N., and Iribarne, L.: A Cloud Service for COTS Component-based Architectures. Computer Standards & Interfaces (Elsevier), Vol. 48, pp. 198–216 (2016)
4. Acceleo, `http://www.acceleo.org`
5. Polymer Project, `https://www.polymer-project.org`