Orientando a Componentes la Web

Componentes Web y El Framework Polymer

Javier Vélez Reyes

@javiervelezreye
Javier.velez.reyes@gmail.com

Junio 2014

I. ¿Quién Soy?

Licenciado en informática por la Universidad Politécnica de Madrid (UPM) desde el año 2001 y doctor en informática por la Universidad Nacional de Educación a Distancia (UNED) desde el año 2009, Javier es investigador y está especializado en el diseño y análisis de la colaboración. Esta labor la compagina con actividades de evangelización, consultoría, mentoring y formación especializada para empresas dentro del sector IT. Inquieto, ávido lector y seguidor cercano de las innovaciones en tecnología.

javier.velez.reyes@gmail.com

@javiervelezreye

linkedin.com/in/javiervelezreyes

gplus.to/javiervelezreyes

jvelez77

javiervelezreyes

youtube.com/user/javiervelezreyes

II. ¿A Qué Me Dedico?

Desarrollado Front/Back

Evangelización Web

Arquitectura Software

Formación & Consultoría IT

E-learning

Diseño de Sistemas de Colaboración

Learning Analytics

Gamificación Colaborativa

Javier Vélez Reyes
@javiervelezreye
Javier.veler.reyes@gmail.com

1 Introducción

- El Camino hacia los Componentes Web
- Qué es un Componente Web
- La Web como Plataforma Orientada a Componentes
- Un nuevos modelo de roles

I. El Camino hacia los Componentes Web

En los últimos años se ha venido observando un claro movimiento hacia la modularización de la Web. Sin el ánimo de ser exhaustivos ni rigurosos podemos citar las principales aportaciones que han promovido este cambio y que, en suma, han contribuido a la aparición de un nuevo estándar para la construcción de sistemas Web basados en Componentes.

Librerías JavaScript

Se promueve el uso de librerías JavaScript que encapsulan funcionalidades básicas para problemas recurrentes en la Web.

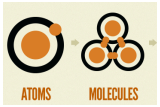
Frameworks MV*

Se construyen frameworks de front-end para asistir al proceso de desarrollo de aplicaciones Web basadas en arquitecturas MVC o MVVM.

Frameworks CSS

Desde el mundo del diseño se construyen frameworks presentacionales que fomentan el uso de prácticas de CSS orientado a objetos

Bootstrap



II. Qué es un Componente Web

Como consecuencia de los esfuerzos de modularización y reutilización anteriores surge, de la mano de la W3C, el **estándar de Componentes Web** que pretende recoger estas inquietudes y resolver algunos de los problemas que presentan las soluciones propuestas hasta la fecha. Aunque, como veremos más adelante son muchas las tecnologías que circundan esta especificación, un Componente Web puede definirse como sigue.

> *Un **Componente Web** proporciona un mecanismo para construir nuevos elementos DOM personalizados que incluyen semántica funcional nueva además de una estructura sintáctica potencialmente diferente.*

Principio I. Extensibilidad

Se pretende proporcionar capacidades a los desarrolladores para que creen sus propias etiquetas. Éstas se caracterizan, a diferencia de los elementos del estándar DOM, por presentar un modelo de comportamiento no inerte

Principio II. Racionalidad

La extensibilidad de la Web alcanzada por estos medios debe desarrollarse de manera que se asegure que ésta se mantenga en consonancia con la forma en que la Web opera actualmente

II. Qué es un Componente Web

Es conveniente destacar algunas características que ya sabemos acerca del funcionamiento de la Web en general y del conjunto de sus elementos constituyentes en particular. En virtud del principio de Racionalidad, los nuevos componentes Web también deberán ajustarse a estas características.

Correspondencia entre HTML y JS

Todos los elementos pueden ser instanciados a través del lenguaje de marcado o JavaScript.

HTML

```
<input type="text"/>
```

JS

```
var e = document.createElement ('input');
e instanceof HTMLInputElement === true;
```

Las etiquetas tienen estado

Las etiquetas tienen un estado caracterizado por el valor en curso de cada uno de sus atributos

JS

```
input.setAttribute ('value', 'foo');
 ...
input.value === 'foo';
```

Las etiquetas son instancias

Todas las etiquetas que aparecen en una página corresponden a instancias diferentes de una clase o prototipo de etiqueta.

JS

```
var e = document.createElement('input');
document.body.appendChild(e);
```

Los elementos tienen un ciclo de vida

Todos los elementos tienen su lógica de inicialización que desarrollan a medida que el navegador renderiza la página

HTML/JS

```
<input value="foo"/>;
...
<script>
  querySelector('input').value === 'foo';
</script>
```

II. Qué es un Componente Web

Es conveniente destacar algunas características que ya sabemos acerca del funcionamiento de la Web en general y del conjunto de sus elementos constituyentes en particular. En virtud del principio de Racionalidad, los nuevos componentes Web también deberán ajustarse a estas características.

Las etiquetas tienen capacidades

Las etiquetas tienen métodos JavaScript que pueden invocarse para operar sobre ellas

HTML

```
<form id="f"/>
```

JS

```
var f = document.querySelector('#f');
f.submit();
```

Las etiquetas pueden tener un modelo DOM complejo

Las etiquetas pueden tener una estructura de objetos interna potencialmente compleja pese a que sean etiquetas sencillas desde la perspectiva del diseñador web

JS

```
<input id="d" type="date" />
<script>
 var date = document.querySelector('#d');
 date.children.length === 0;
</script>
```

Las etiquetas pueden anidarse

Las etiquetas con estructura de bloque pueden presentarse en contextos que aniden a otras etiquetas hijas y a su vez sean anidadas por etiquetas padre

HTML

```
<select>
  <option>1</option>
  <option>2</option>
<select>
```

Los elementos tienen hooks de estilo

Los elementos pueden definir ganchos para definir comportamientos reactivos al cambio de estado desde las etiquetas de estilo

CSS

```
a:hover {
   ...
}
```

III. La Web como Plataforma Orientada a Componentes

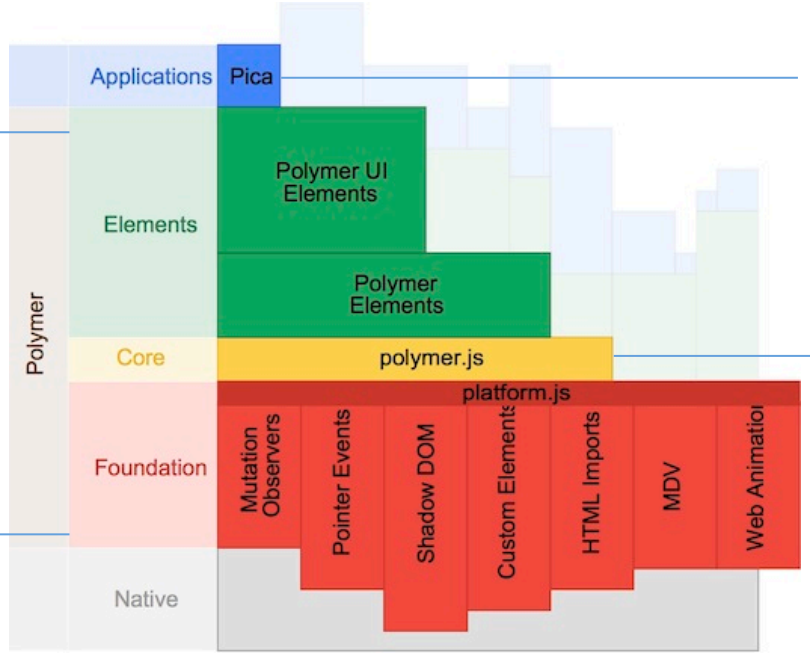
De esta manera asistimos a un proceso de transformación conceptual de la Web, en la que ésta se presenta como una plataforma de hospedaje para una colección de componentes con un comportamiento funcional subyacente que se comunican y coordinan entre si, de acuerdo a diferentes esquemas para ofrecer servicios de valor en el contexto de la página donde se inscriben.

Elementos

Componentes estándar dedicados a dar respuesta a necesidades recurrentes de control o UI que surgen en el desarrollo orientado a Componentes Web

Plataforma

Incluye todas las nuevas capacidades que extienden la Web como plataforma de orientación a componentes



Aplicaciones

Aplicaciones orientadas a componentes al servicio de los usuarios de la Web

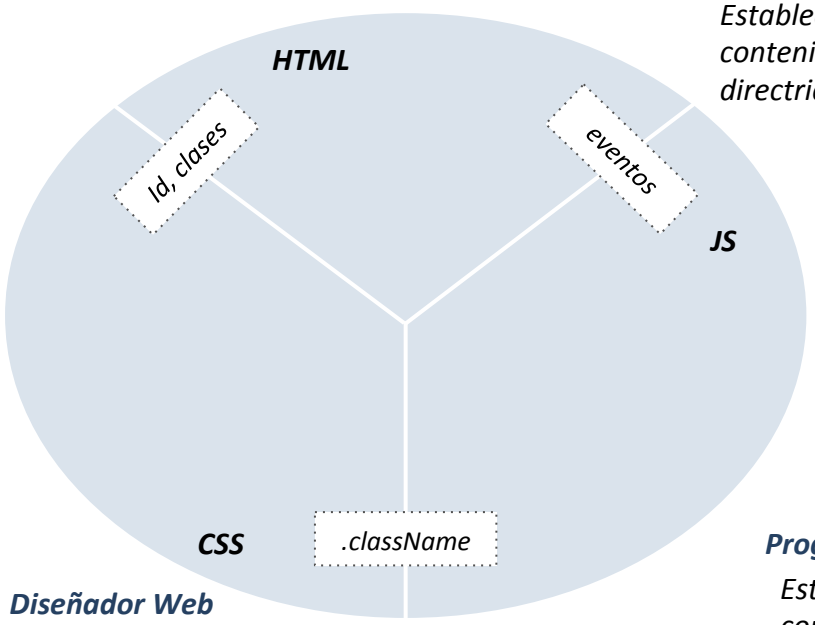
Core

Capacidades funcionales sobre la plataforma que simplifican el proceso de desarrollo de Componentes Web

IV. Un Nuevo Modelo de Roles

Conviene señalar que, en el marco de este nuevo proceso de construcción de soluciones para la Web basadas en componentes, se requieren nuevos roles de especialización técnica y flujos de trabajo establecidos. A continuación bosquejamos los mismos.

En la Web Clásica…



Maquetador Web

Establece la estructura de los contenidos de acuerdo a directrices semánticas

Programador Web

Establece el modelo de comportamiento en reacción a los eventos de interacción del usuario

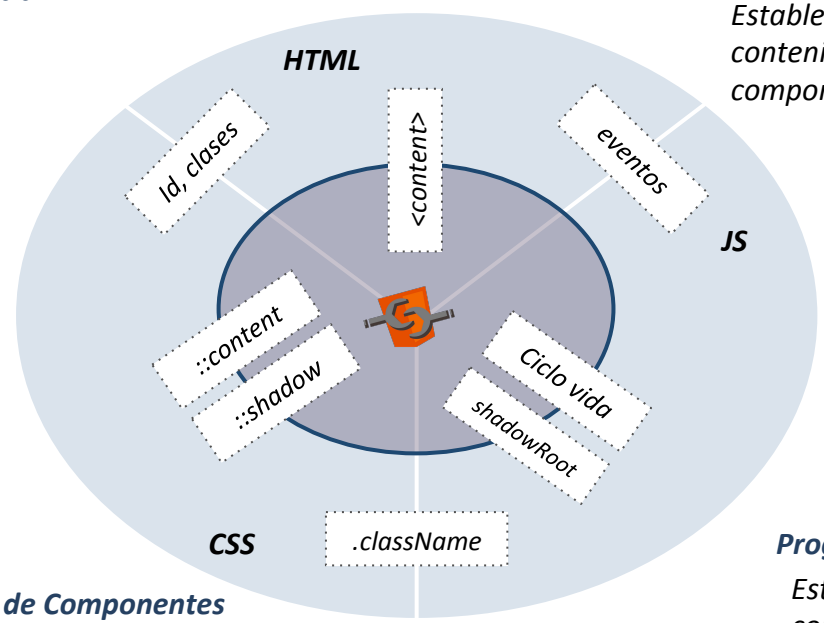
Diseñador Web

Estiliza los contenidos de acuerdo a directrices presentacionales

IV. Un Nuevo Modelo de Roles

Conviene señalar que, en el marco de este nuevo proceso de construcción de soluciones para la Web basadas en componentes, se requieren nuevos roles de especialización técnica y flujos de trabajo establecidos. A continuación bosquejamos los mismos.

En la Web Orientada a Componentes…



Maquetador de Componentes

Establece la estructura de los contenidos internos que presenta el componente cuando se renderiza

Diseñador de Componentes

Define el modelo de rendering del componente de acuerdo a directrices de estilo

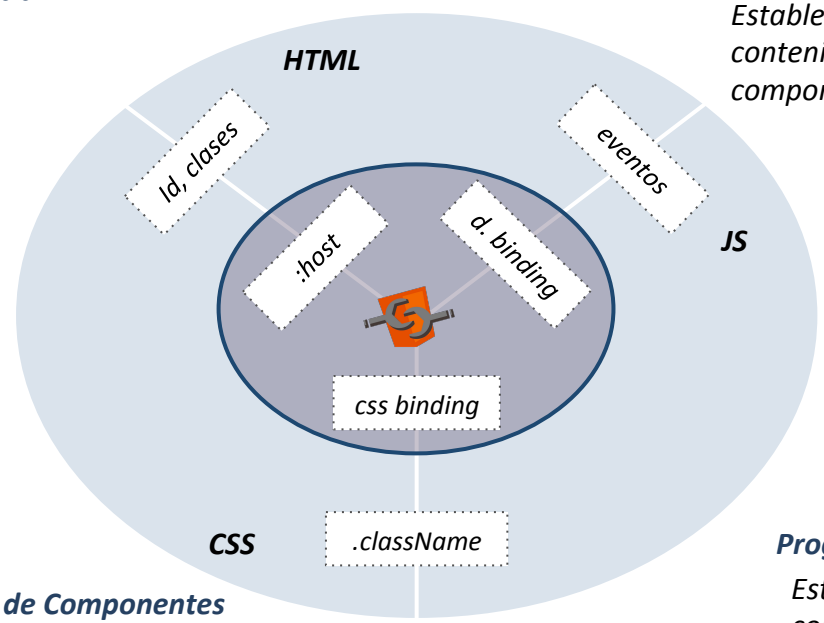
Programador de Componentes

Establece el modelo de comportamiento subyacente que se activa cuando se renderiza el componente

IV. Un Nuevo Modelo de Roles

Conviene señalar que, en el marco de este nuevo proceso de construcción de soluciones para la Web basadas en componentes, se requieren nuevos roles de especialización técnica y flujos de trabajo establecidos. A continuación bosquejamos los mismos.

En la Web Orientada a Componentes…



Maquetador de Componentes

Establece la estructura de los contenidos internos que presenta el componente cuando se renderiza

Diseñador de Componentes

Define el modelo de rendering del componente de acuerdo a directrices de estilo

Programador de Componentes

Establece el modelo de comportamiento subyacente que se activa cuando se renderiza el componente

Javier Vélez Reyes
@javiervelezreye
Javier.veler.reyes@gmail.com

2

La Plataforma Web Orientada a Componentes

- Modelo de Encapsulamiento. Shadow DOM
- Modelo de Renderizado. Templates
- Modelo de Extensión. Custom Elements
- Modelo de Modularización. HTML Imports

I. Introducción

El modelo de componentes para la Web definido por la W3C consiste en cuatro especificaciones tecnológicamente independientes que cubren distintos aspectos del proceso constructivo. A lo largo de este capítulo revisaremos los conceptos esenciales que se manejan en relación a tales especificaciones

Shadow DOM

Ofrece un modelo de encapsulamiento que permite aislar el contenido interno del componente de aquél en la página donde éste es renderizado

Templates

Ofrece un modelo de construcción basado en plantillas inertes de código HTML que sólo son activadas cuando se renderiza el componente

HTML Imports

Ofrece un modelo de modularización basado en la posibilidad de incluir ficheros de código HTML dentro de otros ficheros HTML

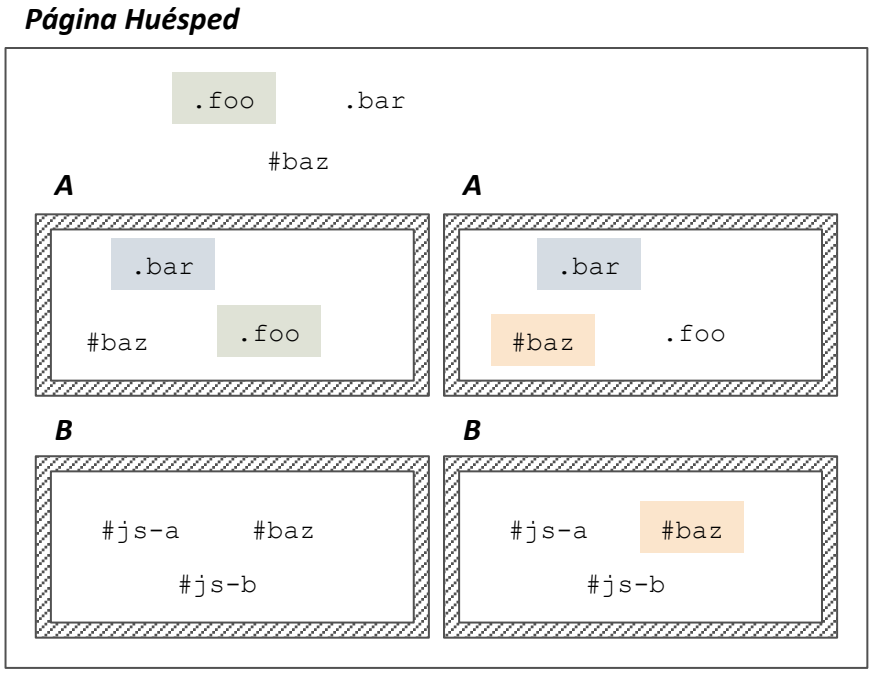
Custom Elements

Ofrece un modelo de extensibilidad que permite a los desarrolladores definir sus propias etiquetas o redefinir semánticamente las etiquetas del estándar DOM

II. Modelo de Encapsulamiento. Shadow DOM

Shadow Host, Shadow Root & Shadow DOM

Sin más infraestructura de soporte, los identificadores JS y CSS utilizados internamente dentro de cada componente podrían colisionar con los usados dentro de la página huésped o en otros componentes residentes. Es necesario proporcionar un mecanismo de encapsulamiento que aísle cada instancia de componente del resto de instancias y de la propia página.



`.foo` ***Aislamiento con la página huésped***

El modelo de encapsulamiento garantiza la ausencia de colisión con los elementos de la página incluso aunque ésta sea a su vez la de otro componente

`.bar` ***Aislamiento entre instancias***

Se debe garantizar que los identificadores dentro de una instancia de componente no entren en conflicto con aquéllos homónimos que, de forma natural, aparecerán en el resto de instancias de componentes del mismo tipo dentro la página huésped

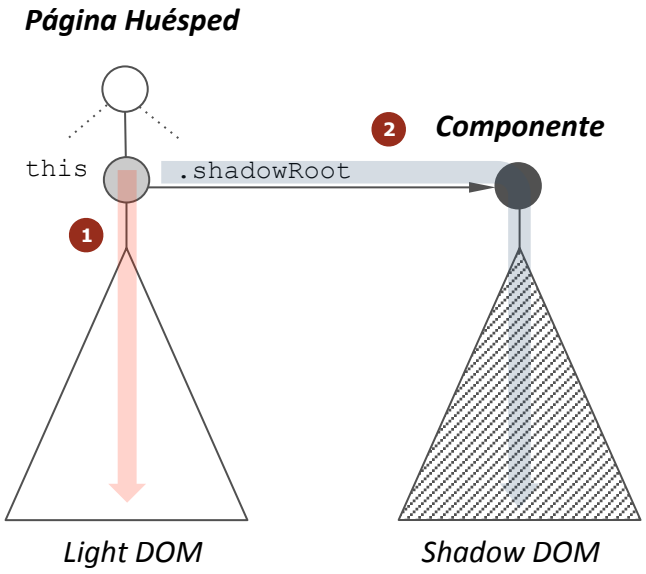
`#baz` ***Aislamiento entre componentes***

Debe asegurarse también el aislamiento entre identificadores homónimos usados entre componentes de distintos tipos

II. Modelo de Encapsulamiento. Shadow DOM

Shadow Host, Shadow Root & Shadow DOM

Definiremos una nueva variante de nodo DOM llamado **Shadow Host**. Un Shadow Host es un nodo DOM que contiene un atributo hacia un nuevo nodo DOM llamado **Shadow Root**. De él se puede hacer depender un subárbol DOM que transcurre en paralelo al documento llamado **Shadow DOM**. Esto establece una variante de contenidos que al ser paralela es ignorada por el algoritmo de rendering del navegador y establece así un espacio de nombres propio.



1 ***Renderizado normal***

En los navegadores sin soporte a componentes web se aplica el algoritmo de renderizado habitual. Como el shadow DOM no es accesible por este algoritmo, el componente no se renderiza y se continúa por el recorrido habitual

2 ***Renderizado shadow***

Cuando el soporte a componentes web está activo, el algoritmo de renderizado discrimina entre nodos normales y nodos shadow host. En éstos últimos, deriva el proceso de renderizado para mostrar el contenido del shadow DOM mientras el Light DOM no se muestra

Nodo DOM normal *Nodo Shadow Host* *Nodo Shadow Root*

II. Modelo de Encapsulamiento. Shadow DOM

Shadow Host, Shadow Root & Shadow DOM

Para convertir un nodo normal en un shadow host es necesario incluir un atributo hacia un shadow root. Esto se hace por medio del método createShadowRoot incluido como extensión en el interfaz del la clase nodo.

HTML

```
<div id="myHost">
   contenido del light dom
</div>
```

Shadow Host

Cualquier tipo de nodo DOM puede ser convertido en Shadow Host. A partir de aquí el esquema de rendering ignora el contenido y pasa a procesar el Shadow DOM

HTML/JS

```
<script>
  var host = document.querySelector('#myHost');
  var root = host.createShadowRoot();
  root.innerHTML = "contenido del shadow dom";
</script>
```

Shadow Host

Primero se localiza el nodo que hará de anfitrión y se convierte en shadow host creando en el un nodo shadow root. Después sólo queda inyectar contenido HTML en el shadow DOM por cualquier mecanismo estándar

II. Modelo de Encapsulamiento. Shadow DOM

Shadow.1.html

Shadow Host, Shadow Root & Shadow DOM

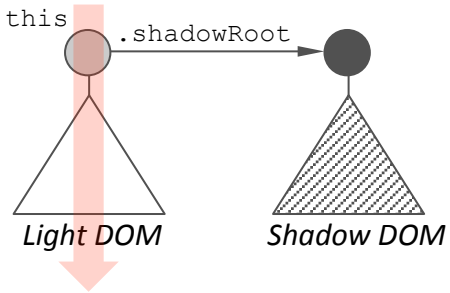
En resumen, el modelo de encapsulación no oculta nada truculento. Cada nodo host dispone de un enlace a un árbol DOM paralelo que puede ser recorrido de forma habitual siempre que se recuerde que su punto de acceso es el shadow root.

A. Acceso a Light DOM

El componente es una etiqueta accesible desde el DOM. Para navegar el light DOM se usan los mecanismos convencionales

JS

```
var host = this;
host.querySelector('.foo');
...
```

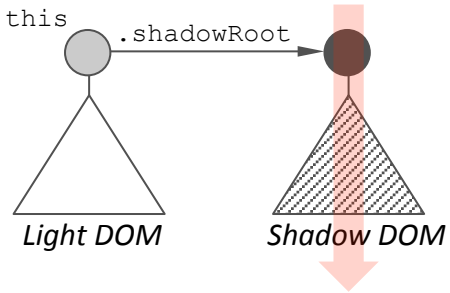


B. Acceso a Shadow DOM

El componente es una etiqueta accesible desde el DOM. Para acceder al root se navega por la propiedad shadowRoot

JS

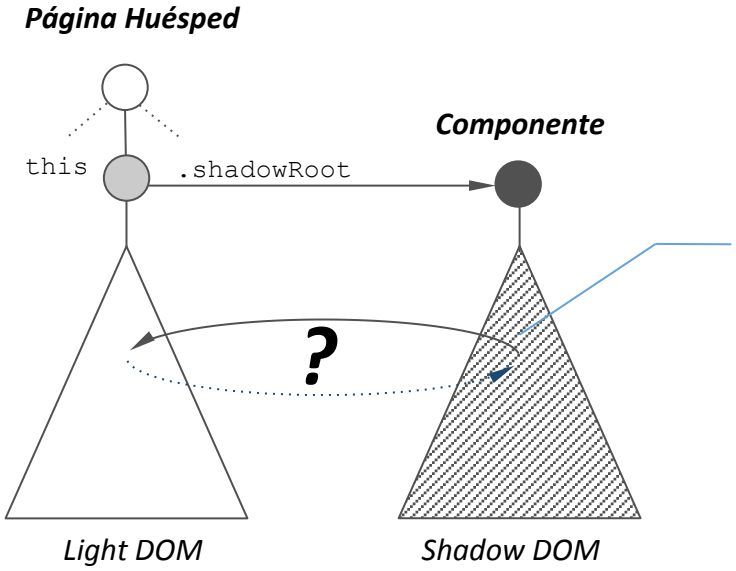
```
var root = this.shadowRoot;
root.querySelector('.foo');
...
```



II. Modelo de Encapsulamiento. Shadow DOM

Puntos de Inserción & Etiqueta Content

En los navegadores sin soporte a componentes, el light DOM permite hacer un renderizado clásico de la información mientras que en aquéllos con soporte el light DOM no es mostrado. Tiene sentido reservar siempre el light DOM para que albergue el contenido exclusivamente semántico de la página mientras que el shadow DOM se encarga del renderizado y la lógica presentacional. Pero, ¿cómo puede accederse desde el shadow DOM al contenido del light DOM?



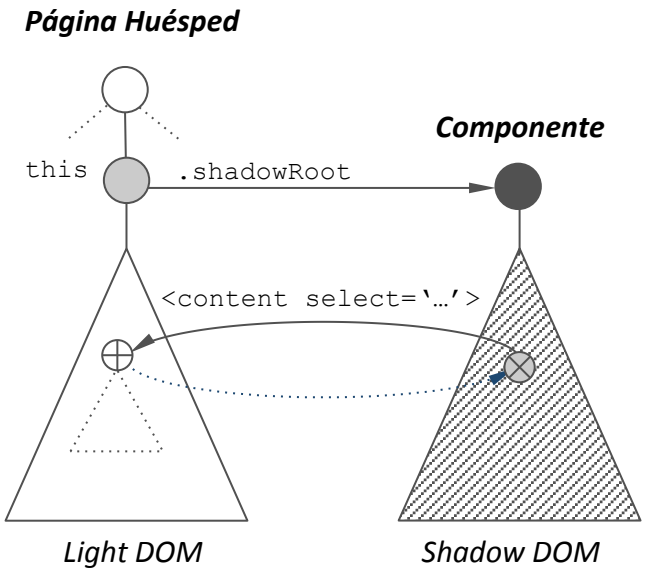
Referencia de información

Si el light DOM contiene el contenido semántico sobre la configuración del componente es muy probable que el shadow DOM requieran discrecionalmente partes de esa información para articular su renderizado. Necesitamos un mecanismo de trasvase de información entre ambos árboles

II. Modelo de Encapsulamiento. Shadow DOM

Puntos de Inserción & Etiqueta Content

En los navegadores sin soporte a componentes el light DOM permite hacer un renderizado clásico de la información. Pero, y aquéllos con soporte el light DOM no es mostrado. Tiene sentido reservar el light DOM para que albergue el contenido exclusivamente semántico de la página mientras que el shadow DOM se encarga del renderizado. Pero, ¿cómo puede accederse desde el shadow DOM al contenido del light DOM?



HTML

```
<div id="myHost">
   <div class="name">Javier Vélez</div>
   <div class="twitter">javiervelezreye</div>
</div>
```

HTML/JS

```
<script>
  var host = document.querySelector('#myHost');
  var root = host.createShadowRoot();
  root.innerHTML =
    'D. <content select=".name"></content>' +
    '<hr>' +
    '@<content select=".twitter"></content>';
</script>
```

⊕ *Nodo con contenido Semántico* ⊗ *Punto de inserción*

II. Modelo de Encapsulamiento. Shadow DOM

Shadow.2.html

Puntos de Inserción & Etiqueta Content

Es posible consultar programáticamente la colección de nodos del light DOM que son referidos desde un punto de inserción así como saber en qué puntos de inserción se distribuye un determinado nodo del light DOM.

A. Acceso a Nodos Distribuidos

Seleccionado un punto de inserción se pueden localizar los nodos del Light DOM que son distribuidos en el shadow DOM

JS

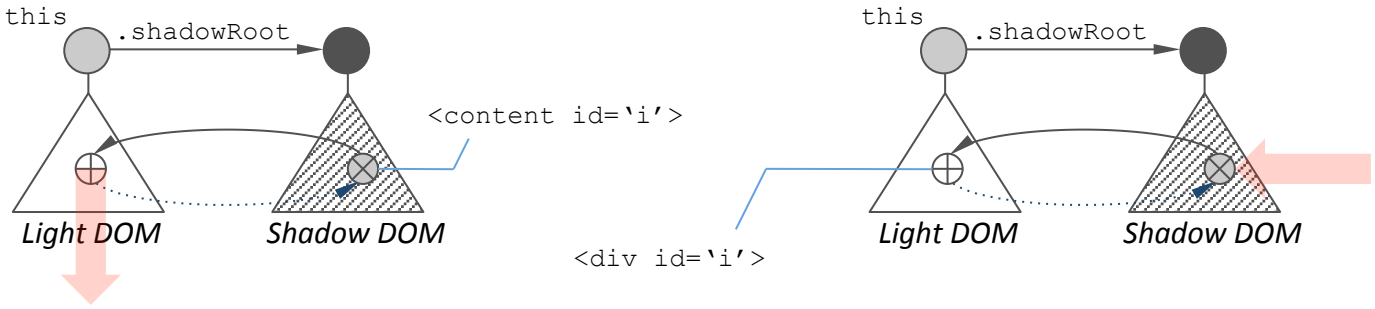
```
var ip = root.querySelector('#i');
var dNodes = ip.getDistributedNodes ();
dNodes...
```

B. Acceso a Puntos de Inserción de Destino

Para saber los puntos de inserción desde donde un determinado nodo del Light DOM es referido se accede a los puntos de inserción de destino

JS

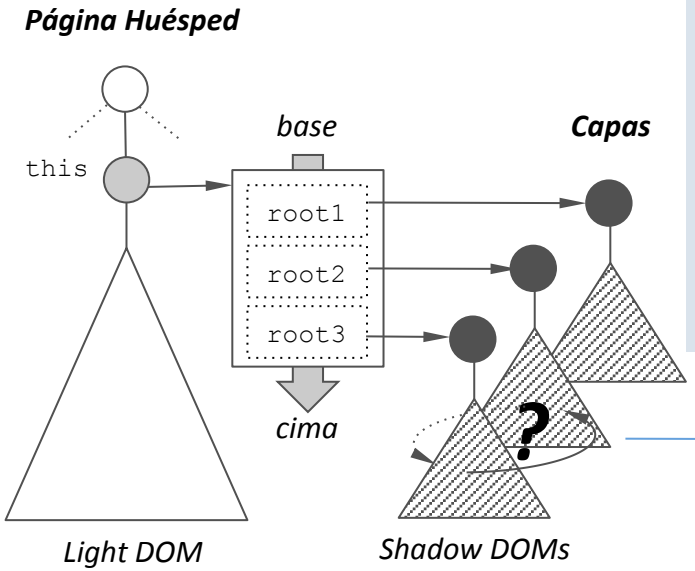
```
var e = host.querySelector ('#i');
var ips = e.getDestinationInsertionPoints();
ips...
```



II. Modelo de Encapsulamiento. Shadow DOM

Puntos de Inserción en la Sombra & Etiqueta Shadow

Para dar mayor flexibilidad en el proceso de diseño es posible incluir dentro de un shadow host varios shadow roots. Éstos son gestionados por el nodo anfitrión de acuerdo a una pila LIFO para emular los mecanismos de diseño por capas. Ya sabemos como referir información del light DOM desde el shadow DOM. Sin embargo, ¿cómo hacer lo mismo desde un shadow DOM hacia el resto de shadow DOM que éste tiene por debajo en la pila?



HTML/JS

```
<script>
  var host = document.querySelector('#myHost');
  var root1 = host.createShadowRoot();
  var root2 = host.createShadowRoot();
  var root3 = host.createShadowRoot();
  root1.innerHTML = ...
  root2.innerHTML = ...
  root3.innerHTML = ...
</script>
```

Trasvase de información entre capas

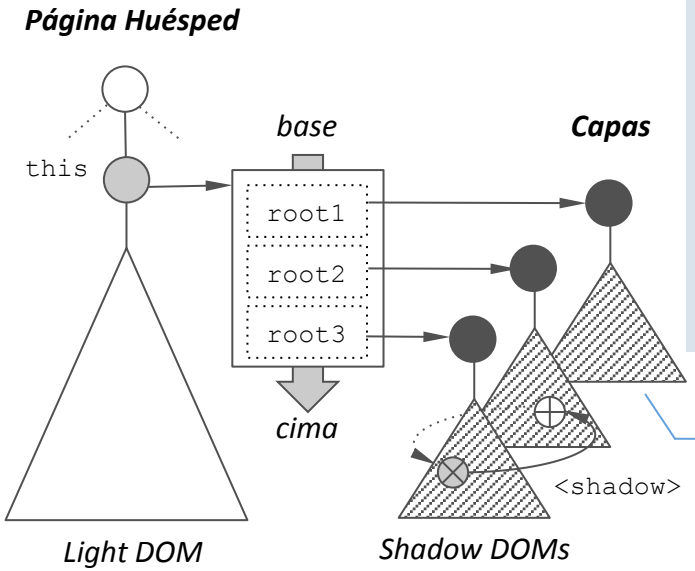
De igual manera, para articular un mecanismo de diseño flexible y ponente, es conveniente disponer de una manera de recuperar información de los shadow DOM por debajo dentro de la pila

II. Modelo de Encapsulamiento. Shadow DOM

Shadow.3.html

Puntos de Inserción en la Sombra & Etiqueta Shadow

Para dar mayor flexibilidad en el proceso de diseño es posible incluir dentro de un shadow host varios shadow roots. Éstos son gestionados por el nodo anfitrión de acuerdo a una pila LIFO para emular los mecanismos de diseño por capas. Ya sabemos como referir información del light DOM desde el shadow DOM. Sin embargo, ¿cómo hacer lo mismo desde un shadow DOM hacia el resto de shadow DOM que éste tiene por debajo en la pila?



HTML/JS

```
<script>
  ...
  root1.innerHTML = 'Esta capa está oculta';
  root2.innerHTML = 'D. <content select...';
  root3.innerHTML =
    '<div class=".box">' +
    '  <shadow></shadow>' +
    '</div>';
</script>
```

Capas ocultas

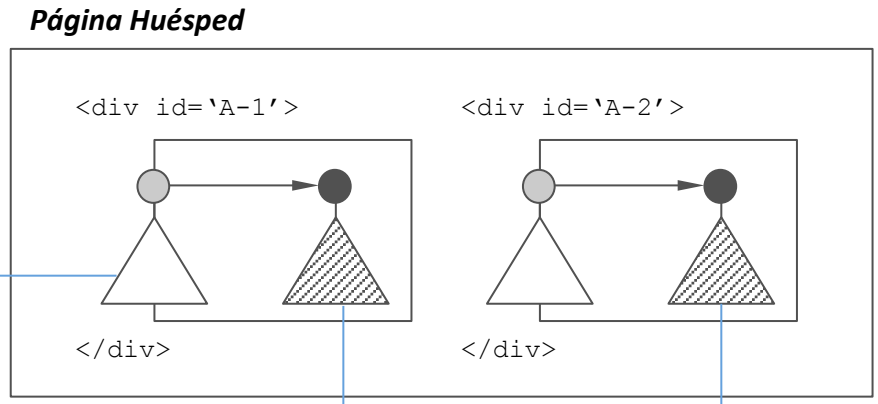
Las capas que no tuvieran una referencia <shadow> desde capas superiores no serían incluidas en el proceso de renderizado

⊕ *Nodo de contenido en capa* ⊗ *Punto de inserción en la sombra*

III. Modelo de Renderizado. Templates

Plantillas Inertes y Activación de Código por Clonación

No es poco frecuente encontrar situaciones en las que es necesario instanciar dos ocurrencias diferentes de un mismo componente sobre la misma página, aunque sea con light DOM diferentes. ¿Pomo podemos evitar el trabajo de reescribir el shadow DOM entero para cada instancia de forma eficaz y segura?



Light DOM diferentes

Cada instancia sin embargo contiene unos contenidos semánticos distintos dentro del subárbol light que sirven para configurar la misma de forma particular

Shadow DOM idénticos

El contenido de los dos subárboles shadow es el mismo puesto que ambas etiquetas responden a instancias distintas (-1 y -2) del mismo componente A

III. Modelo de Renderizado. Templates

Template.1.html

Plantillas Inertes y Activación de Código por Clonación

Una plantilla <template> es un nuevo tipo de etiqueta que puede incluir cualquier contenido HTML y puede ubicarse en cualquier parte del documento. Este contenido es inerte en tanto que no ejecuta los scripts, no carga las imágenes y no reproduce videos o audios. Desde JS puede accederse este contenido inerte y copiarse para formar el shadow DOM momento en el cual será procesado por el algoritmo de rendering y convenientemente activado.

HTML

```
<template id="news">
  <img src="">
  <div class="body"></div>
</template>
<div id="new-1"></div> <div id="new-2"></div>
```

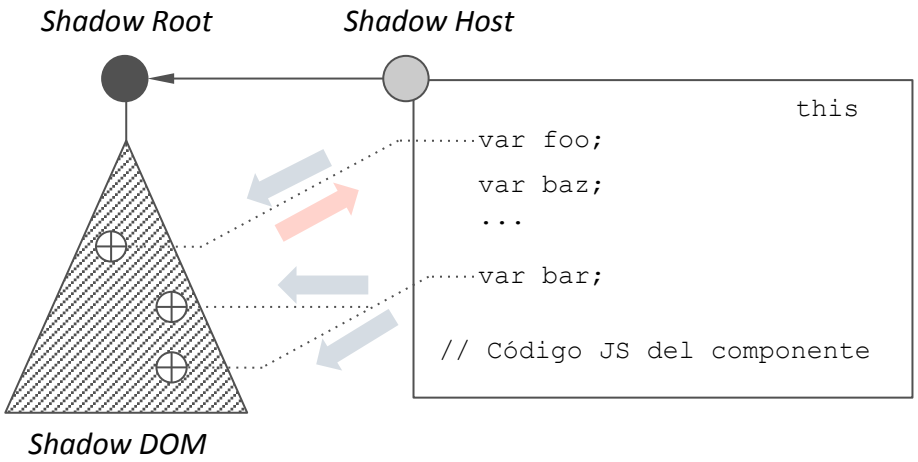
HTML/JS

```
<script>
  function createNew(container, image, body) {
    var root = document.querySelector(container).createShadowRoot();
    var template = document.querySelection("#news").content;
    template.querySelector("img").scr = image;
    template.querySelector(".body").textContent = body;
    root.appendChild(template.content.cloneNode(true));
  }
  createNew('#new-1', 'imagen1.png', 'noticia 1');
  createNew('#new-2', 'imagen2.png', 'noticia 2');
</script>
```

III. Modelo de Renderizado. Templates

Plantillas de Enlace de Datos

Resulta muy potente disponer de un mecanismo de observación que permita a los nodos DOM del árbol shadow mantener su contenido sincronizado dinámicamente con ciertas partes del modelo interno de JS que tiene el componente. En este sentido podemos distinguir dos tipos de situaciones.



Enlace de Datos Unidireccional

Los datos de la vista se sincronizan con los cambios en el modelo pero no al revés. Es típico de nodos de texto y atributos

Enlace de Datos Bidireccional

Los datos de la vista se sincronizan con los cambios programáticos del modelo y recíprocamente, las acciones del usuario sobre la vista propagan cambios en el modelo. Es típico de nodos de entrada de datos, o de aquéllos manejados por un escuchador de evento que cambie su valor en JS

⊕ *Nodo de enlace de datos*

Template.2.html

III. Modelo de Renderizado. Templates

Plantillas de Enlace de Datos

Polymer extiende el concepto de plantilla estándar con plantillas para el enlace de datos. Estas plantillas incluyen en el atributo bind una referencia al la parte de modelo JS con que van a enlazarse. También se proporcionan plantillas para control de flujo condicional, iterativo, recursivo y anidado, atributos condicionales, controladores de eventos, etc.

Plantilla de Enlace de Datos

HTML

```
<template bind="{{person as me}}">
  {{me.name}} - {{me.twitter}}
</template>
```

Plantilla Condicional

HTML

```
<template bind="..." if="truthy">
  {{me.name}} - {{me.twitter}}
</template>
```

Plantilla Iterativa

HTML

```
<template repeat="{{item in items}}">
  <li>{{item}}</li>
</template>
```

Plantillas Anidada

HTML

```
<template repeat="{{u in users}}">
  <template repeat="{{i in u.items}}">
    <li>{{i}}</li>
  </template
</template>
```

Plantillas Referidas & Anidadas

HTML

```
<template repeat="i in items" id="t">
 <li>{{i}}
 <ul>
   <template repeat="{{i.hijos}}" ref="t">
   </template>
 </ul>
</template>
  ...

<template bind ref="t">
</template>
```

III. Modelo de Renderizado. Templates

Plantillas de Enlace de Datos

Polymer extiende el concepto de plantilla estándar con plantillas para el enlace de datos. Estas plantillas incluyen en el atributo bind una referencia al la parte de modelo JS con que van a enlazarse. También se proporcionan plantillas para control de flujo condicional, iterativo, recursivo y anidado, atributos condicionales, controladores de eventos, etc.

Enlace a Texto

HTML

```
<p>{{user.name}}</p>
```

Enlace a Atributo

HTML

```
<img src="{{image}}" alt="imagen">
```

Manejador de Evento Estándar

HTML

```
<button on-click="{{doClick}}">
   Ok
</button>
```

Enlace a Atributo Condicional

HTML

```
<div class="card" hidden?="{{show}}"
```

Manejador de Evento de Negocio

HTML

```
<div id="my-component"
     on-done="{{doAlarm}}">
   ...
</div>
```

Enlace con Operador Condicional

HTML

```
<template repeat="{{item, i in items}}">
  <div class="{{(i%2==0)? 'on' : 'off'}}">
    {{item}}
  </div>
</template>
```

IV. Modelo de Extensión. Custom Elements

</>

Creación de Etiquetas Personalizadas. Custom Tags

Ahora que tenemos los mecanismos necesarios para definir y encapsular comportamiento y lógica de presentación personalizada necesitamos una manera de referirlo nominalmente dentro de nuestra página Web en forma de etiquetas personalizadas. Esto permite distinguir los componentes de las etiquetas estándar lo que aumenta la legibilidad de la página.

Página Huésped

```
...
<wc-calendar>
  <wc-calendar-day>12</wc-calendar-day>
  <wc-calendar-month>06</wc-calendar-month>
  <wc-calendar-year>2014</wc-calendar-year>
</wc-calendar>
...
```

Etiquetas Personalizadas

Las etiquetas personalizadas son un tipo de elementos personalizados que permite utilizar una referencia nominal para instanciar un componente ya construido dentro de una página. No obstante existen dos reglas de nombrado

Regla 1. Nombres con al menos un guión

Los nombres de las etiquetas personalizadas deben tener al menos un guión en el nodeName para diferenciarlas de las etiquetas estándar.

```
<wc-calendar> ... </wc-calendar>
```

Regla 2. Nombres reservados

No pueden utilizarse como nombres de etiquetas personalizadas cualquiera de las cadenas con guión que se enumeran a continuación.

```
annotation-xml color-profile font-face
font-face-src font-face-uri font-face-format
font-face-name missing-glyph
```

IV. Modelo de Extensión. Custom Elements

</>

Custom.html

Creación de Etiquetas Personalizadas. Custom Tags

Debemos preocuparnos, por un lado, de registrar cada componente construido como un nuevo tipo de etiqueta dentro del sistema de etiquetas del navegador y después de instanciar componentes tanto desde el lenguaje de marcado como a través de JS.

Registro de un Componente

HTML/JS

```
<polymer-element name="wc-foo">
  <template>...</template>
  <script>
    Polymer ('wc-foo', {
     // prototipo del componente (foo, bar...)
    });
  </script>
</polymer-element>
```

JS

```
var p = Object.create(HTMLElement.prototype);
p.foo = function (){...};
p.bar = 7;
var Foo = document.registerElement(
            'wc-foo',
            {prototype: p});
```

Instanciación de un Componente

HTML

```
<wc-foo id="a">
  ...
</wc-foo>
<wc-foo id="b">
  ...
</wc-foo>
```

JS

```
var a = new Foo ();
var b = document.createElement('wc-foo');
document.body.appendChild (a);
document.body.appendChild (b);
```

Tanto los snippets de código de registro como de instanciación son versiones equivalentes y mutuamente excluyentes

IV. Modelo de Extensión. Custom Elements

Extensión de Etiquetas Estándar. Extended Element Types

Por otro lado, resulta de interés extender las propias etiquetas del estándar HTML para generar variantes que incluyan nuevo comportamiento o lógica presentacional cuando se rendericen. Estas etiquetas mantienen, naturalmente, el nombre de la etiqueta a la que extienden pero incluyen un calificador is para indicar al navegador de que se trata de una variante distinta

Página Huésped

```
...
<button id="btn" is="fancy-button">
  Ok
</button>

<p is="lorem-ipsum"><p>

...


<script>
  var btn = document.querySelector('#btn');
  btn.addEventListener('click', function(e){
    ...
  });
</script>
```

Extensión de Etiquetas Estándar

Las etiquetas del estándar se extienden con variantes cualificadas nominalmente dentro del atributo is. Estas etiquetas mantienen el nombre, métodos y atributos de la etiqueta de la que derivan pero pueden sobrescribir o agregar nuevas capacidades

Herencia de Capacidades

La nueva variante de botón incluye, por herencia todos los atributos, métodos y modelo de eventos que tiene la etiqueta base de la que deriva

IV. Modelo de Extensión. Custom Elements

</>

Extensión de Etiquetas Estándar. Extended Element Types

De forma similar al caso anterior, comenzaremos por registrar cada componente construido como una variante que extiende una etiqueta del estándar y después nos ocuparemos de instanciar componentes tanto desde el lenguaje de marcado como a través de JS.

Extensión de una Etiqueta

HTML/JS

```
<polymer-element name="fancy-button"
                 extends="button">
  <template>...</template>
  <script>
    Polymer ('fancy-button', {
      // prototipo de la etiqueta (foo, bar...)
    });
  </script>
</polymer-element>
```

JS

```
var p = Object.create(HTMLButtonElement.prototype);
p.foo = function (){...};
p.bar = 7;
Var FancyButton = document.registerElement(
                    'fancy-button',
                    {extends: 'button',
                     prototype: p});
```

Instanciación de una Etiqueta

HTML

```
<button is="fancy-button"
        id="a"> Ok </button>

<button is="fancy-button"
        id="b"> Ok </button>
```

JS

```
var a = new FancyButton ();
Var b = document.createElement
              ('fancy-button');
document.body.appendChild (a);
document.body.appendChild (b);
```

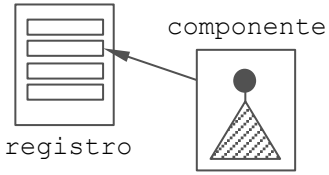
Tanto los snippets de código de registro como de instanciación son versiones equivalentes y mutuamente excluyentes

IV. Modelo de Extensión. Custom Elements

Es importante recalcar que los Componentes Web, una vez establecidos como elementos personalizados atraviesan distintas fases a lo largo del proceso de construcción, inserción y potencial borrado del árbol DOM de la página huésped. El estándar ofrece método manejadores que permiten la interposición de código en respuesta a cada una de las fases que caracterizan dicho ciclo de vida. A continuación describimos someramente las mismas.

CREATED

El componente se ha creado y ha sido registrado con éxito como una nueva etiqueta o extensión

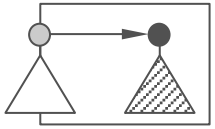


READY

El componente está preparado. Shadow DOM creado. Observadores establecidos. Manejadores de eventos vinculados

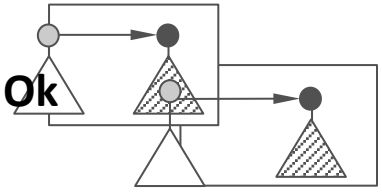
ATTACHED

Una nueva instancia de algún componente creado se ha insertado dentro de la página huésped



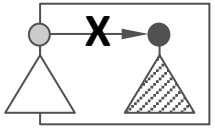
DOM_READY

Se garantiza que todos los componentes anidados dentro de éste están preparados



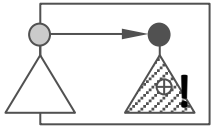
DETACHED

La instancia del componente se ha eliminado de la página huésped



ATRIBUTE_CHANGED

Un atributo de la instancia del componente fue añadido, borrado o actualizado



V. Modelo de Modularización. HTML Imports

Importación de documentos HTML

Según lo expuesto, la implementación de un componente incluye un código de plantilla inerte, especificaciones de estilo y no poca lógica de scripting para especificar el modelo de comportamiento del mismo. Parece necesario disponer de un mecanismo de modularización que permita incluir todo ese contenido en un fichero de índice a parte HTML que pueda importarte a la página huésped.

Página Huésped

```
<!doctype html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <script src="platform.js"></script>
    <script src="polymer.js"></script>
    <link rel="import" href="wc-foo.html">
  </head>
  <body>

    <wc-foo></wc-foo>

  </body>
</html>
```

wc-foo.html

```
<polymer-element name="wc-foo">
  <template>...</template>
  <script>
    Polymer ('wc-foo', {
      // prototipo del componente
    });
  </script>
</polymer>
```

Import Location

La URL asociada a un import se llama import location. Para acceder a un documento hospedado en otro dominio deben habilitarse los permisos CORS.

V. Modelo de Modularización. HTML Imports

Empaquetamiento de Recursos

Import.html

Los mecanismos de modularización proporcionados por la directiva import ofrecen oportunidades para el empaquetamiento de colecciones de recursos y manera que éstos puedan ser entregados al cliente final a través de una única import location.

Página Huésped

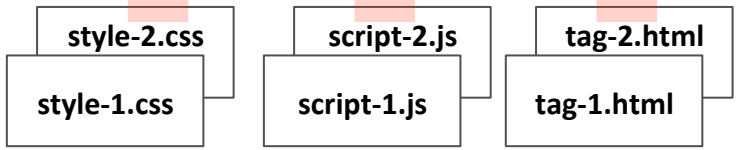
```
<!doctype html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <script src="platform.js"></script>
    <script src="polymer.js"></script>
    <link rel="import" href="my-lib.html">
  </head>

  <body>
   ...
  </body>
</html>
```

my-lib.html

```
<!doctype html>
<html lang="es">
  <head>
    <link rel="import" href="tag-1.html">
    <link rel="import" href="tag-2.html">
    <link rel="stylesheet" href="style-1.html">
    <link rel="stylesheet" href="style-2.html">
    <script src="script-1.js"></script>
    <script src="script-2.js"></script>
  </head>
  <body>
    <polymer-element>
      ...
    </polymer-element>
  </body>
</html>
```

Carga Indirecta

A través de la carga indirecta de un fichero de índice se cargan en la página huésped todos los recursos a los que éste apunta o contiene



V. Modelo de Modularización. HTML Imports

Eventos de Carga y Error

Para controlar programáticamente desde JavaScript los procesos de carga de recursos de documentos HTML importados mediante la clausula import la plataforma proporciona sendos eventos sobre los que se pueden inyectar manejadores.

Enlace a Texto

El elemento link lanza un evento load cuando un recurso HTML ha terminado su proceso de carga de forma exitosa.

Enlace a Atributo

Si se produce algún error durante el proceso de carga del recurso HTML, por ejemplo error 404, recurso no encontrado, la etiqueta link lanzará un evento error

HTML/JS

```
<script async>
  function handleLoad(e) {
    console.log('Loaded import:' + e.target.href);
  }
  function handleError(e) {
    console.log('Error loading import:' + e.target.href);
  }
</script>

<link rel="import" href="file.html"
      onload="handleLoad(event)" onerror="handleError(event)">
```

V. Modelo de Modularización. HTML Imports

Manipulación del Contenido de los Recursos Importados

Importar un recurso HTML mediante la cláusula import no supone un volcado tal cual del contenido sino que requiere de un procesamiento del mismo. Esto implica que es desarrollador tiene la oportunidad de capturar programáticamente dicho el contenido y operar sobre él a conveniencia.

HTML/JS

```
<head>
  <link id="foo-link" rel="import" href="wc-foo.html">
</head>
<body>
  ...
  <script>
    var link = document.querySelector('#foo-link');
    var content = link.import;
    var el = content.querySelector('...');
    ...
  </script>
</body>
```

V. Modelo de Modularización. HTML Imports

Acceso al Recurso Importado y a la Página Huésped

Debemos ser conscientes de que los ficheros importados, en tanto que son documentos HTML, pueden incluir fragmentos de scripting. Dentro de estos scripts es posible referir tanto al documento HTML que los contiene como al documento HTML donde son hospedados a través de la directiva de importación.

Página Huésped

```
<!doctype html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <script src="platform.js"></script>
    <script src="polymer.js"></script>
    <link rel="import" href="file.html">
  </head>

  <body>
   ...
  </body>
</html>
```

file.html

```
<!doctype html>
<html lang="es">
  <head>
    ...
  </head>
  <body>
    <script>
      var f = document.currentScript.ownerDocument;
      var h = document;
      // f refiere a este documento HTML
      // h refiere a la página huésped
    </script>
  </body>
</html>
```

Regla de importación 1.

El contexto en el que se evalúan los script dentro de una página importada es el de la página huésped

Regla de importación 2.

Las importaciones son no bloqueantes. Sin embargo los script dentro de la página importada se ejecutan en orden

Regla de importación 3.

Las importaciones desde una misma URL son procesadas solo una vez con lo que sus script sólo se ejecutan una vez

Javier Vélez Reyes
@javiervelezreye
Javier.veler.reyes@gmail.com

3

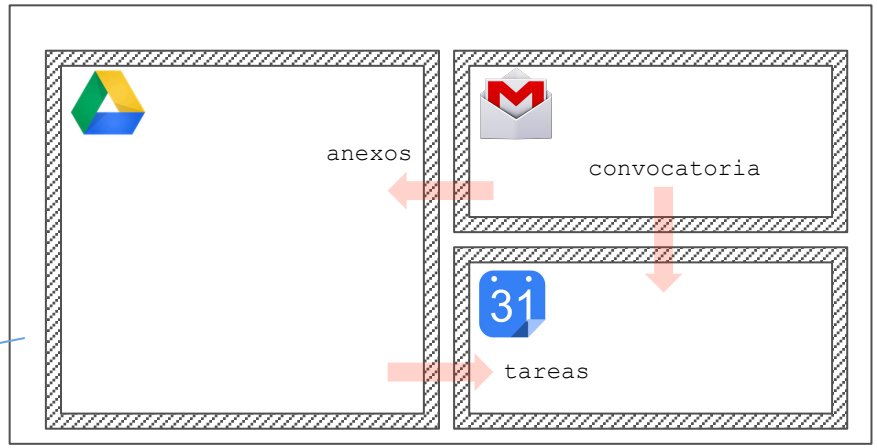
Aplicaciones Web Basadas en Componentes

- Coordinación de Componentes Web
- Modelo de Localización entre Componentes
- Modelos de Comunicación entre Componentes
- Modelos de Coordinación entre Componentes
- Ejemplos

I. Introducción

No debemos olvidar que nuestro último objetivo cuando construimos componentes Web es elaborar **sistemas funcionales complejos** que puedan operar sobre la Web como plataforma. Para hacer frente este problema es necesario poner foco de atención en la relación de coordinación inherente que debe establecerse entre los servicios para dar lugar a componentes que ofrezcan servicios de mayor valor.

Sistema Orientado a Componentes



El poder de la relación

Cada componente ofrece de forma aislada un servicio de valor al usuario. La composición coordinada de estos componentes ofrece un servicio de mayor nivel.

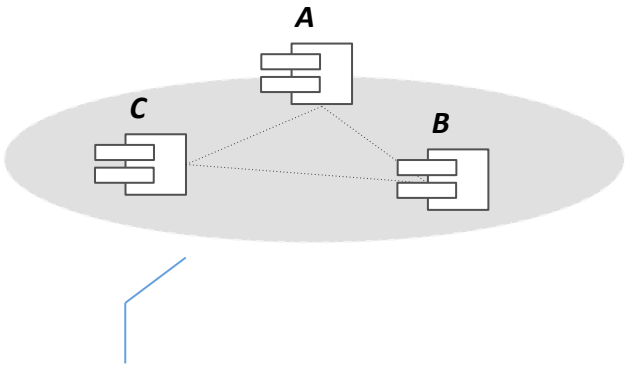
Herencia de Capacidades

El diseño orientado a componentes de esos sistemas ofrece grandes ventajas en relación a los principios fundacionales clásicos de reutilización, modularidad, carácter abierto, sustitución, segregación por roles, etc.

I. Introducción

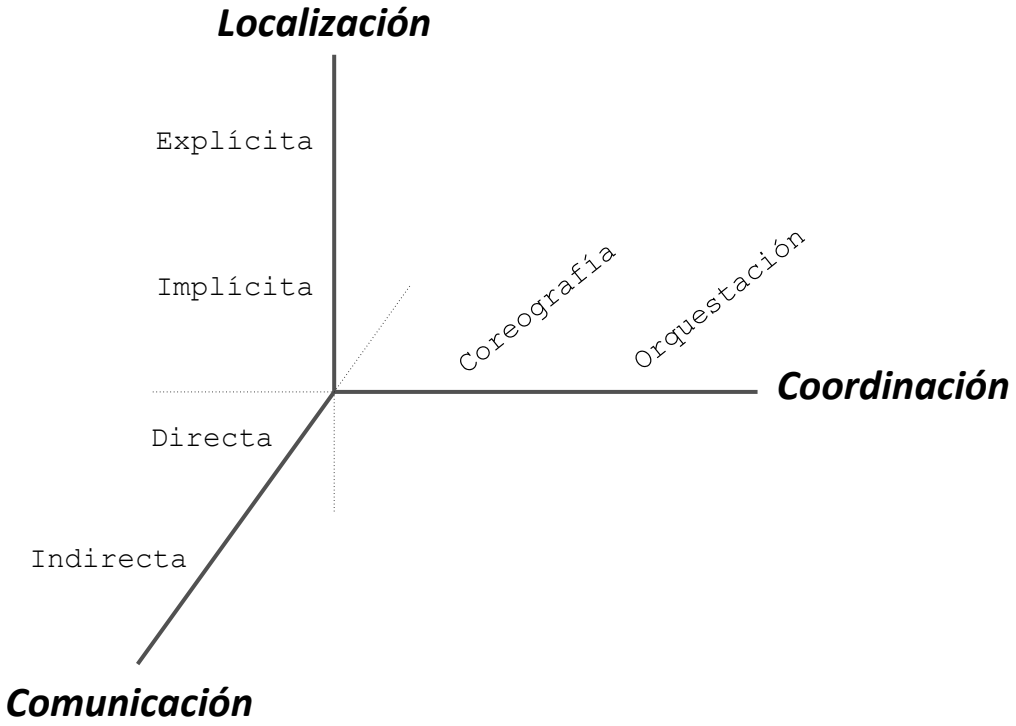
Espacios de Coordinación

En relación a los procesos de coordinación necesarios para construir aplicativos operativos sobre la Web basados en componentes es necesario prestar atención a tres ejes dimensionales que se dibujan como cuestiones independientes que hay que resolver. Estos ejes caracterizan las propiedades de lo que daremos en llamar espacios de coordinación entre componentes.



Espacio de Coordinación

Un espacio de coordinación es un ámbito que da soporte a la interacción entre componentes para ofrecer servicios de mayor nivel

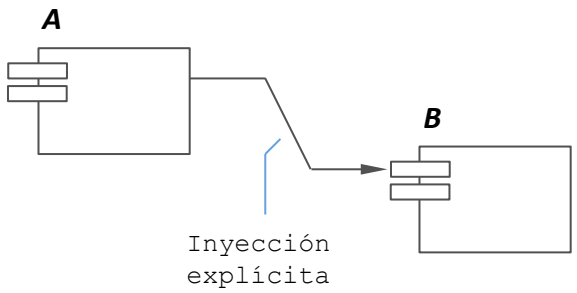


II. Modelos de Localización entre Componentes Web

El modelo de localización está relacionado con la selección de los mecanismos adecuados para el establecimiento efectivo de conexiones semánticas o funcionales entre componentes que deben mantenerse en coordinación para ofrecer un servicio de valor añadido al contexto donde habitan.

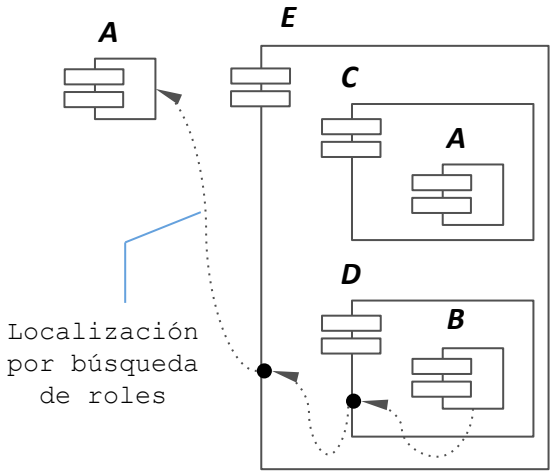
Localización Explícita

Cada componente recibe una referencia explícita nominal de los componentes con los que tiene que establecer un ejercicio de coordinación



Localización Implícita

Cada componente es responsable de localizar (por rol) dentro del entorno a los demás componentes con los que tiene que establecer un ejercicio de coordinación

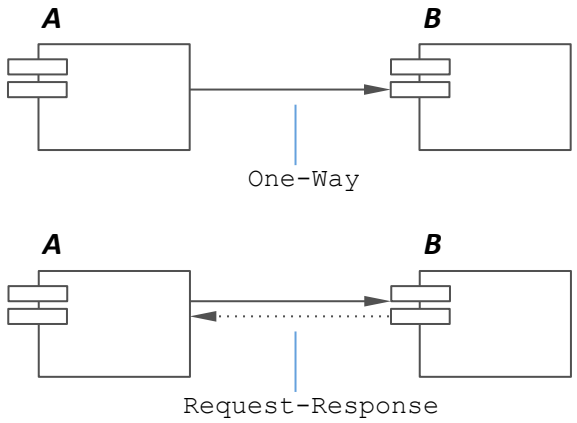


III. Modelos de Comunicación entre Componentes Web

Por otro lado, es preciso determinar el modelo de comunicación que se establecerá entre los componentes implicados en una coordinación. A este respecto, podemos distinguir entre esquemas de comunicación directa, o esquemas de comunicación indirecta, que se llevan a cabo a través de un tercer componente como intermediario.

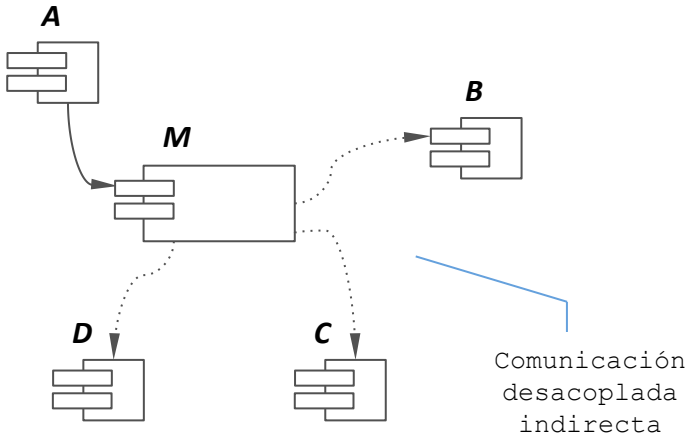
Comunicación Directa

En la comunicación directa los componentes se relacionan 1:1 a través de la invocación directa de métodos de negocio de acuerdo a esquemas One-Way o Request-Response



Comunicación Indirecta

En la comunicación indirecta, la relación entre componentes se produce por eventos, típicamente a través de un mediador que puede intervenir discrecionalmente en el proceso

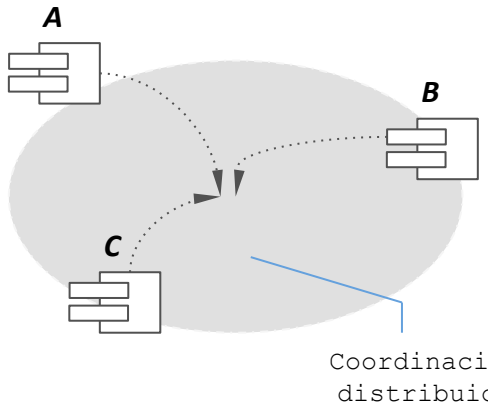


IV. Modelos Coordinación de Componentes Web

Asimismo, necesitamos establecer los mecanismos de coordinación que serán aplicados entre los componentes implicados en aras a ofrecer servicios de valor añadido. En términos generales, es posible establecer dos grandes familias de esquemas coordinativos, los coreográficos y los de orquestación en función de su carácter distribuido o centralizado.

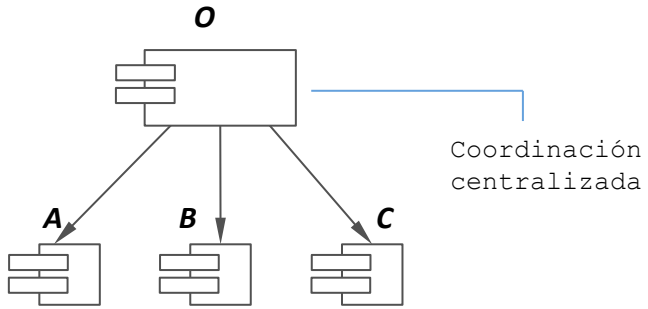
Coordinación Coreográfica

La coordinación coreográfica responde a esquemas de participación distribuida donde cada componente es responsable de reaccionar de manera autónoma de acuerdo a un consenso preestablecido



Coordinación Orquestal

En los mecanismos por orquestación la lógica de coordinación es ejecutada por un componente central que dispone de un script que detalla el proceso coordinativo que es necesario efectuar entre los componentes participantes

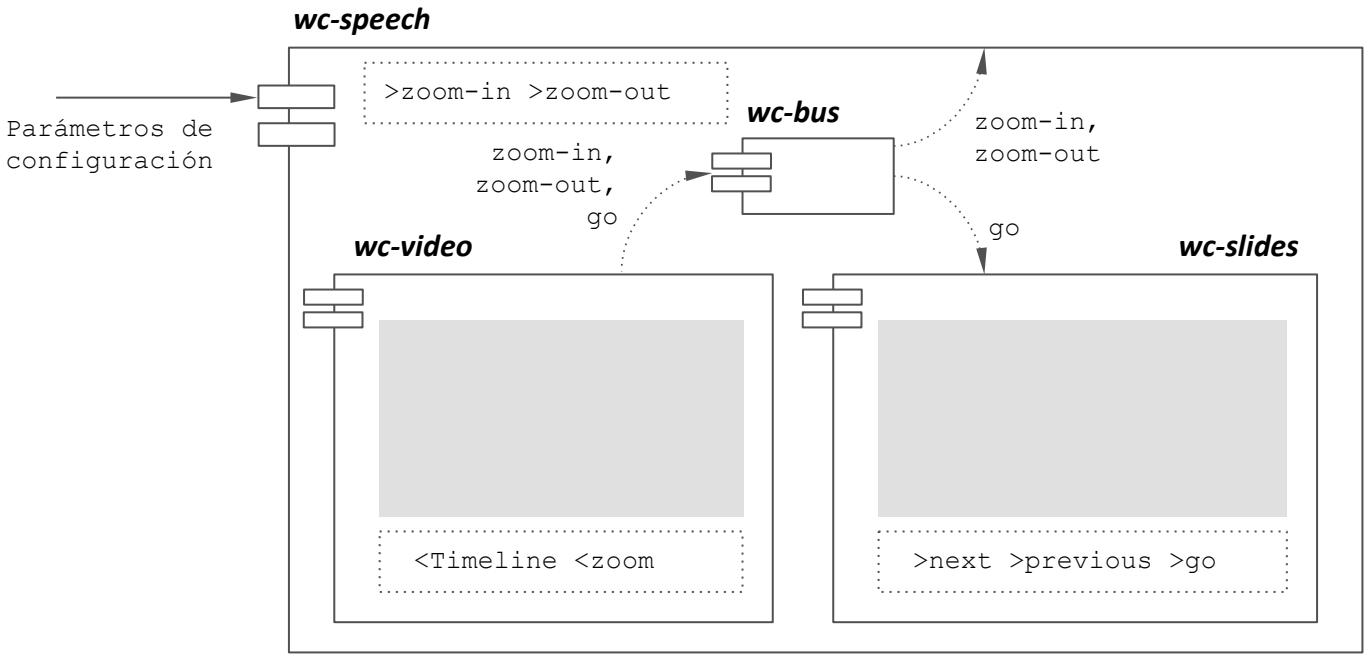


V. Ejemplos

wc-speech (1)

Ejemplo I. Coordinación de Video y Transparencias en Local

Localización. Explícita. [Despliegue Explícito de Variantes según parámetro]
Comunicación. Indirecta Local. [Arquitectura de Eventos en Bus]
Coordinación. Coreografía. [Desacoplamiento nominal por eventos]



V. Ejemplos

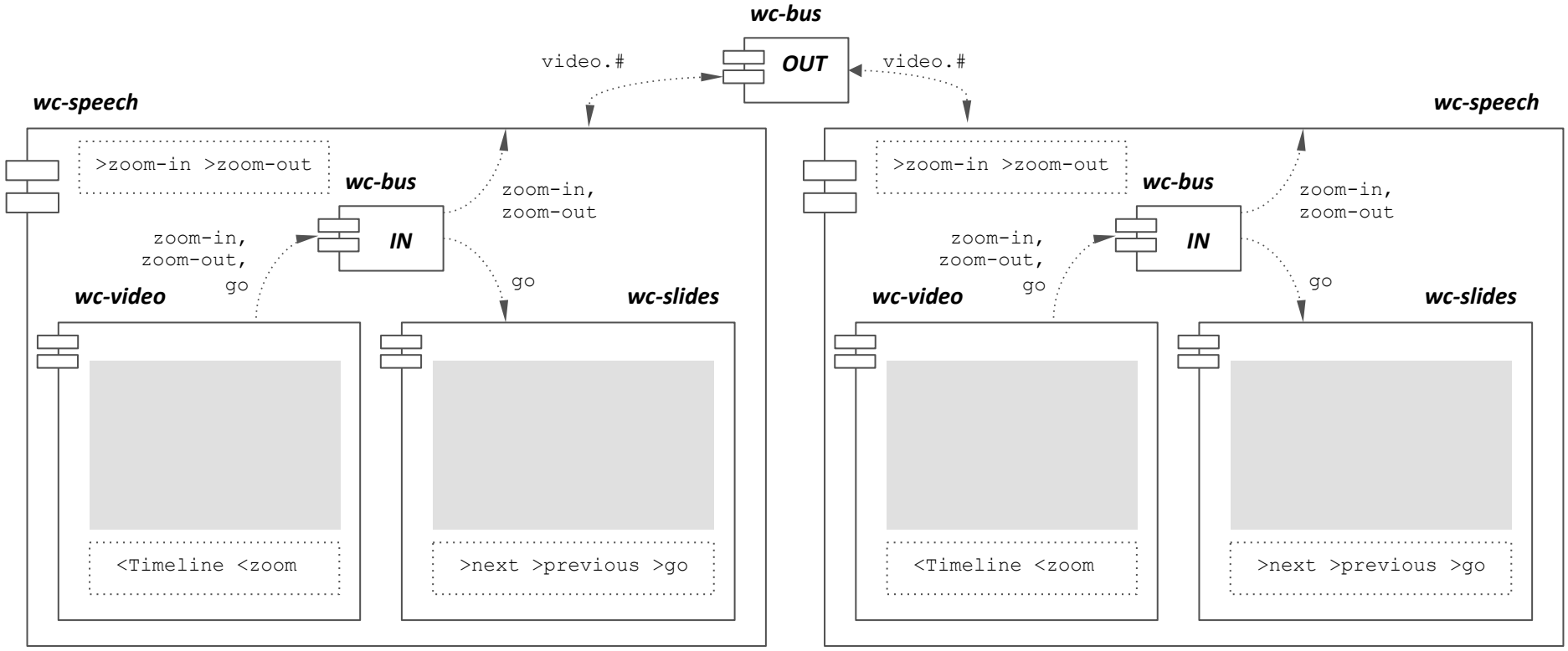
wc-speech (2)

Ejemplo II. Coordinación de Video y Transparencias en Remoto

Localización. Explícita. [Despliegue Explícito de Variantes según parámetro]

Comunicación. Indirecta Remota. [Arquitectura de Eventos en Bus]

Coordinación. Coreografía. [Desacoplamiento nominal por eventos]



V. Ejemplos

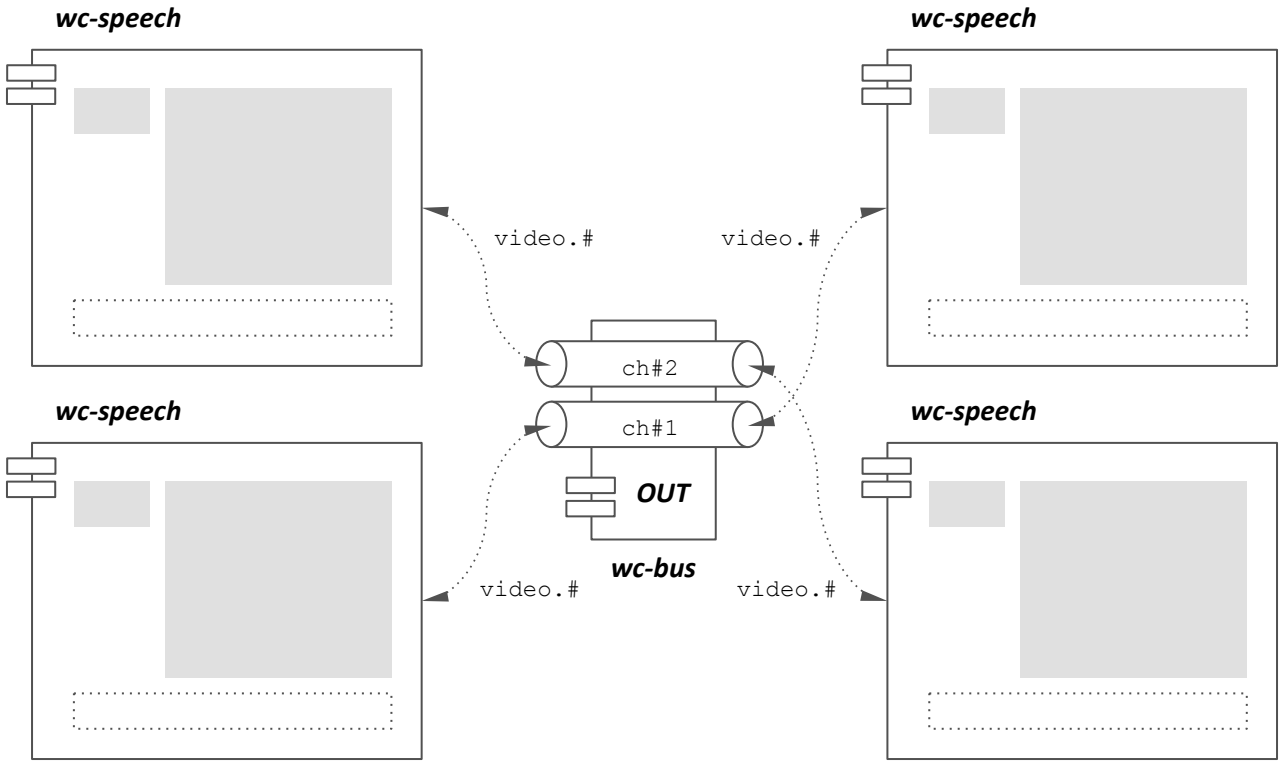
wc-speech (2+2)

Ejemplo III. Coordinación de Video y Transparencias 2 a 2 Cruzadas

Localización. Explícita. [Despliegue Explícito de Variantes según parámetro]

Comunicación. Indirecta Remota. [Arquitectura de Eventos en Bus]

Coordinación. Coreografía. [Desacoplamiento nominal por eventos]



V. Ejemplos

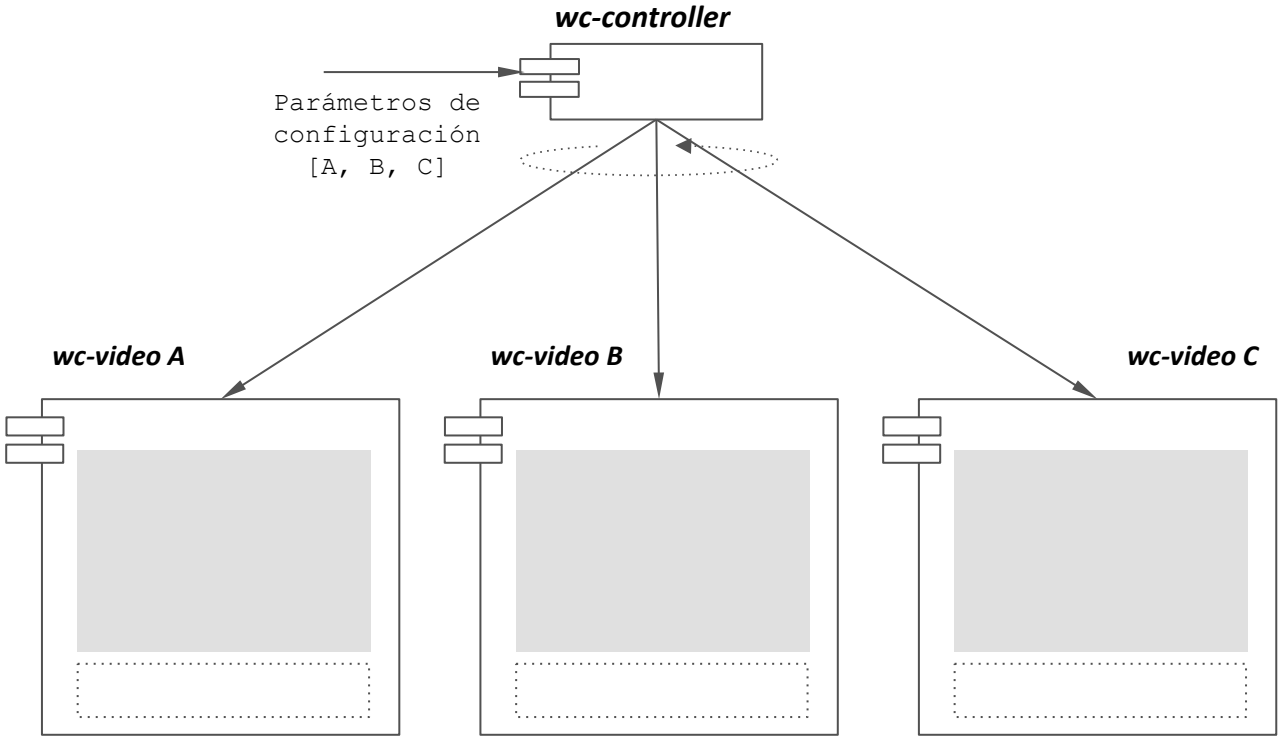
wc-speech (3)

Ejemplo IV. Coordinación de Videos en Round Robin

Localización. Explícita. [Inyección explícita de componentes controlados]

Comunicación. Directa Local. [Arquitectura de Eventos en Bus]

Coordinación. Orquestación. [Centralización de la lógica de coordinación]



La Plataforma Web
Orientada a Componentes

Modelo de
Roles

Aplicaciones basadas
en Componentes Web

Javier Vélez Reyes

@javiervelezreye
Javier.velez.reyes@gmail.com

Orientando a Componentes la Web

Componentes Web y El Framework Polymer

Javier Vélez Reyes

@javiervelezreye
Javier.velez.reyes@gmail.com

Junio 2014