Arquitecturas Orientadas a Componentes Web

Patrones de Composición

Javier Vélez Reyes

@javiervelezreye
Javier.velez.reyes@gmail.com

Noviembre 2016

{codemotion}

¿Quién Soy?

Licenciado por la UPM desde el año 2001 y doctor en informática por la UNED desde el año 2009, Javier conjuga sus labores como profesor e investigador con la consultoría y la formación técnica para empresa. Su línea de trabajo actual se centra en la innovación y desarrollo de tecnologías para la Web. Además realiza actividades de evangelización y divulgación en diversas comunidades IT y es coordinador de varios grupos de ámbito local como NodeJS Madrid o Madrid JS. Forma parte del programa Polymer Polytechnic Speaker y es mentor del capítulo de Madrid de Node School.

javier.velez.reyes@gmail.com

@javiervelezreye

linkedin.com/in/javiervelezreyes

gplus.to/javiervelezreyes

jvelez77

javiervelezreyes

youtube.com/user/javiervelezreyes

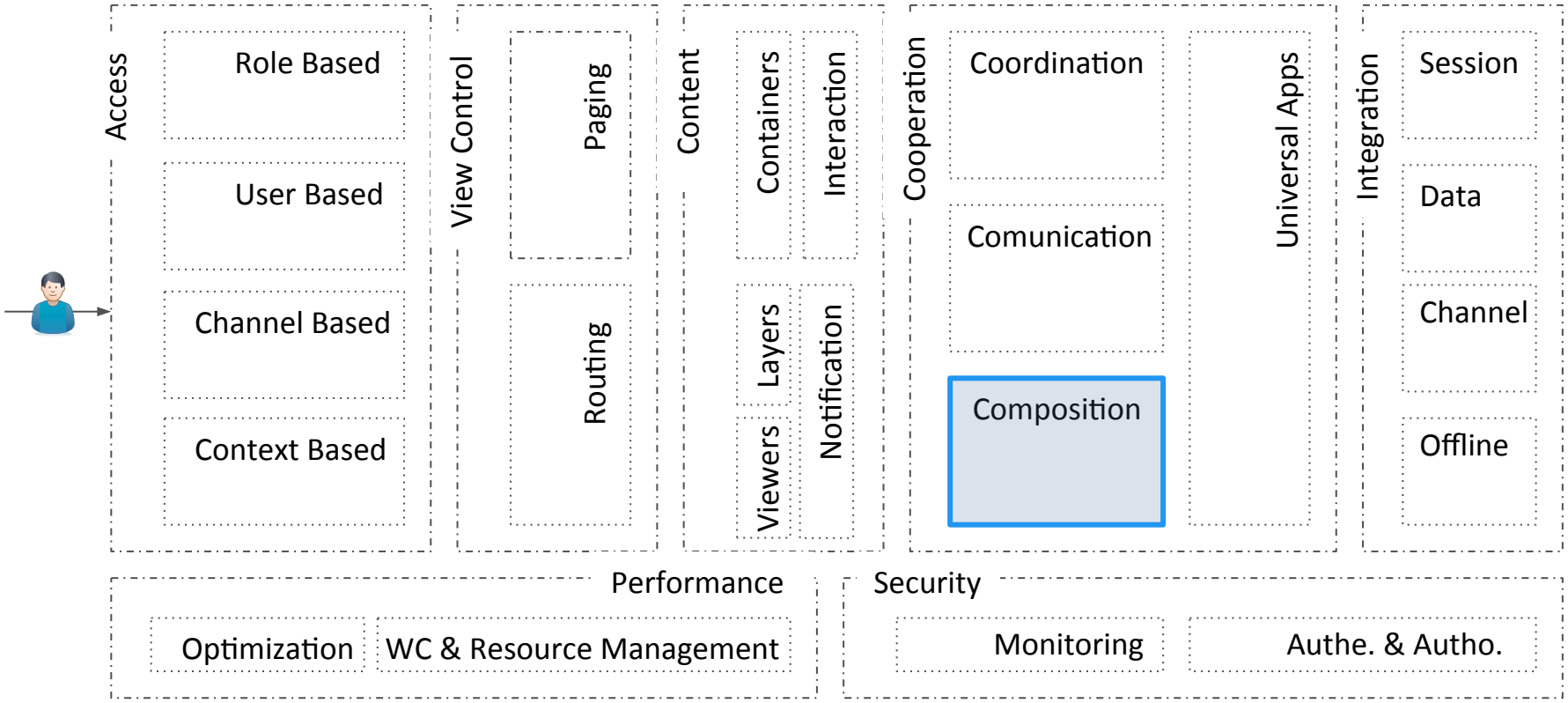
Javier Vélez Reyes
@javiervelezreye
Javier.velez.reyes@gmail.com

1 Introducción

- Arquitectura de Referencia para Componentes Web
- Arquitectura de Referencia & Composición
- El Problema de Composición

I. Arquitectura de Referencia Para Componentes Web

Una arquitectura de referencia establece orientación acerca de cómo pueden construirse soluciones orientadas a componentes Web.

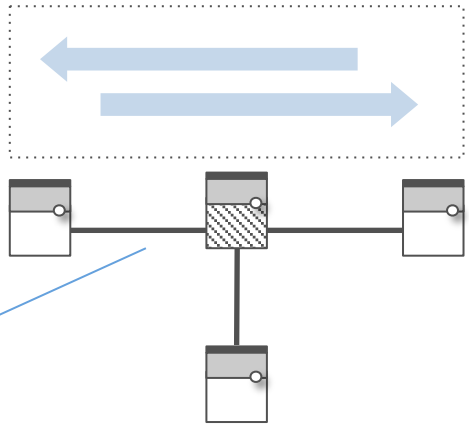


III. El Problema de Composición

La Composición en Palabras

La composición es el proceso por el cual se establece un conjunto de enlaces compositivos entre componentes para permitir la comunicación.

Qué



Habilitar la comunicación

Cubierto dicho espacio se posibilita la comunicación entre componentes lo que a su vez da paso a colaboraciones

Actividad bilateral

La composición es un proceso local que, en principio, sólo atañe al par de componentes en los extremos de un enlace compositivo


Componente objetivo

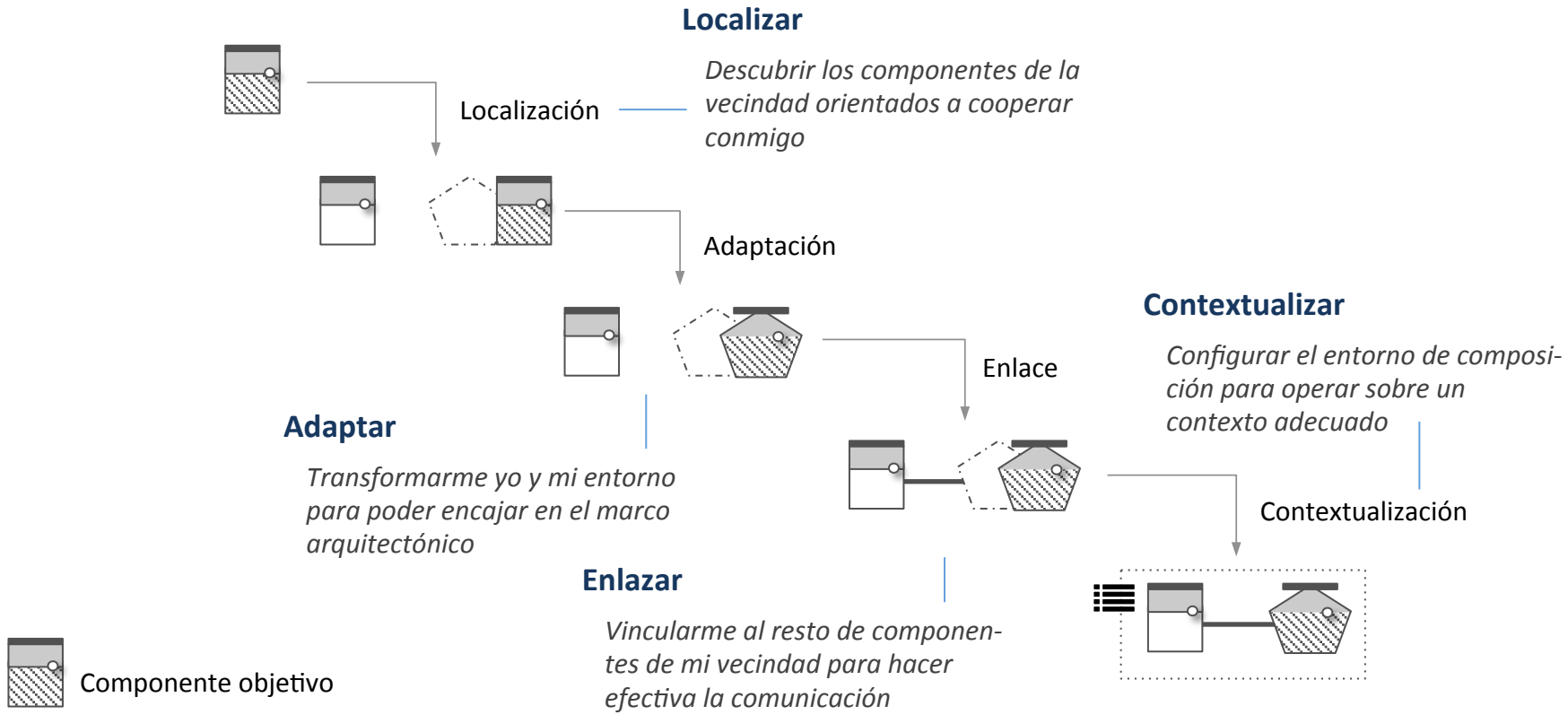

Componente Participante

III. El Problema de Composición

El Proceso de Composición. Perspectiva de Caja Blanca

Internamente, el proceso de composición se entiende como una sucesión de 4 fases que se desarrollan en cascada: localizar, adaptar, enlazar y contextualizar.

Cómo

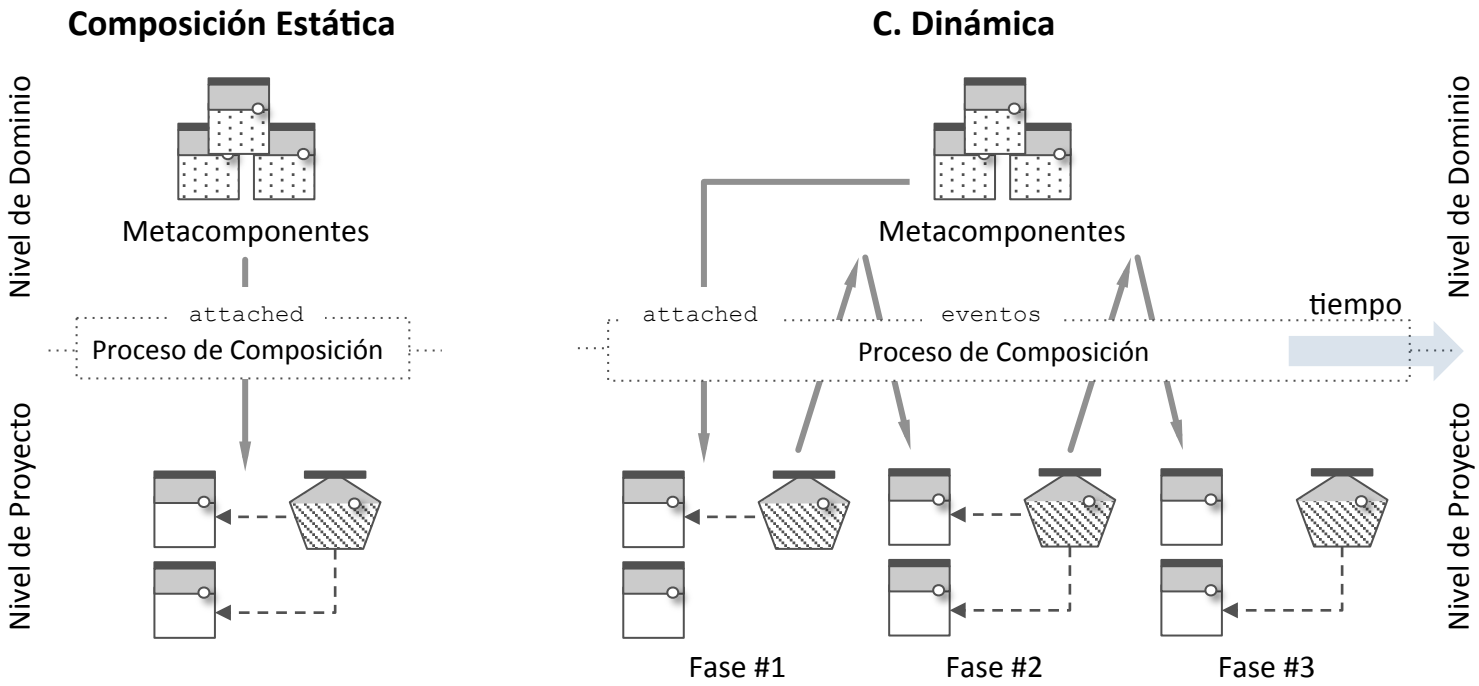


III. El Problema de Composición

Composición Estática & Dinámica

La composición estática se produce una vez al principio del ciclo de vida. La composición dinámica se repite recurrentemente en respuesta a cambios ambientales.

Cuándo

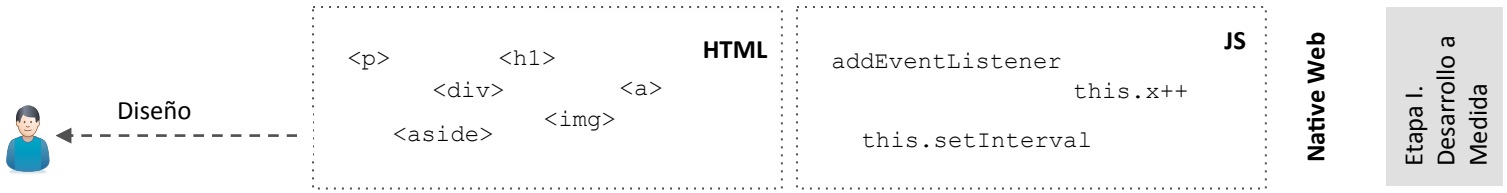


III. El Problema de Composición

Construcción por Composición

En la etapa de madurez I el desarrollo se orienta a cliente, se opera a muy bajo nivel de abstracción y los índices de reutilización son casi inexistentes.

Dónde

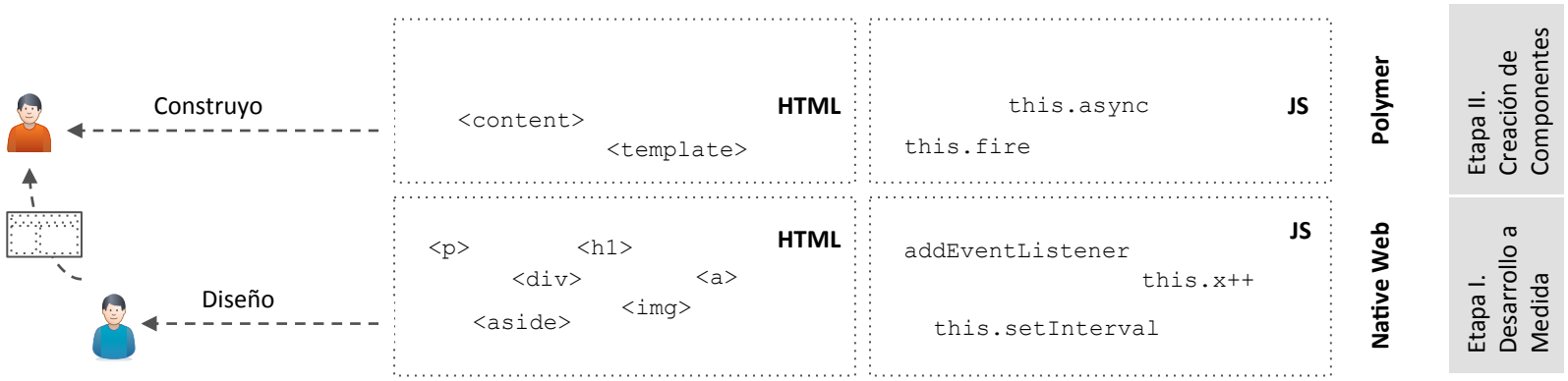


III. El Problema de Composición

Construcción por Composición

En la etapa de madurez II se construyen componentes Web que encapsulan modelos de interacción recurrentes. Se fomenta la abstracción y la reutilización.

Dónde

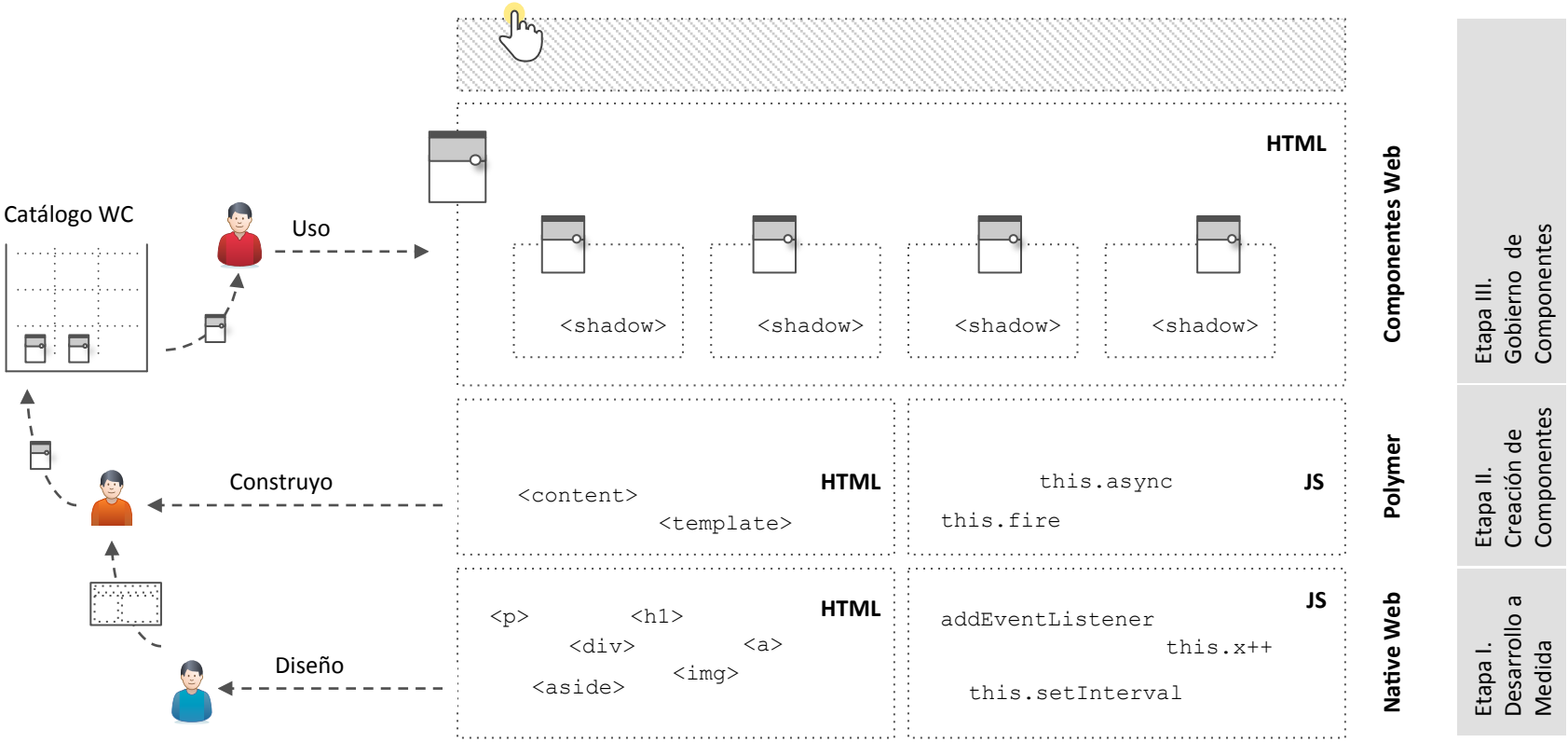


III. El Problema de Composición

Construcción por Composición

En la etapa de madurez III, las corporaciones reconocen las ventajas de operar con un catálogo de componentes. Se construyen aplicaciones por composición declarativa.

Dónde



III. El Problema de Composición

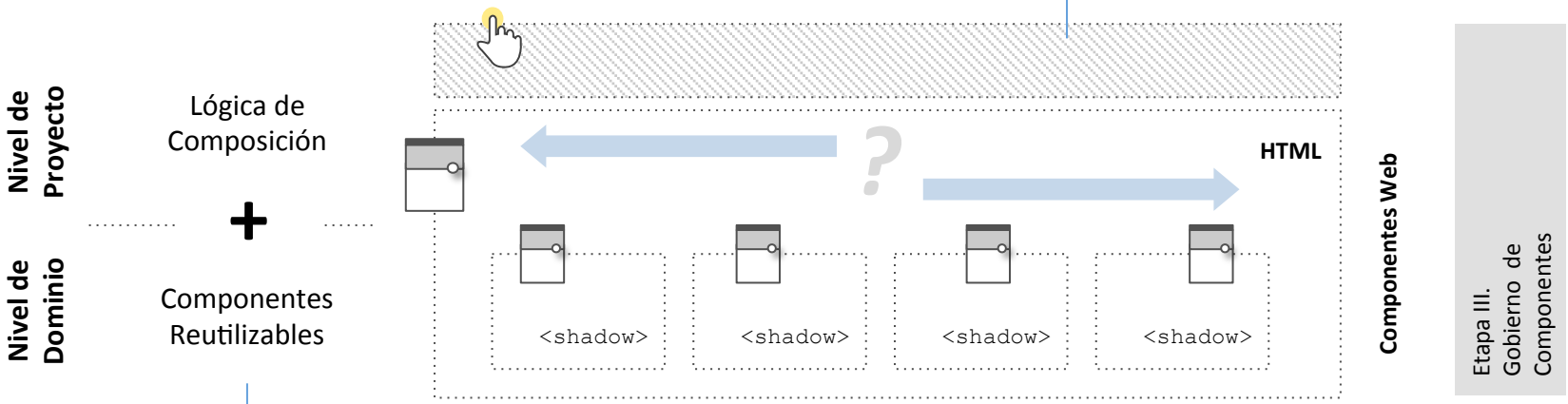
Construcción por Composición

Construir por composición consiste en combinar estratégicamente lógica reutilizable en componentes Web con lógica compositiva específica

Experiencia de Continuidad

Se pretende articular un modelo de interacción que ofrezca una sensación de continuidad en su uso

Dónde



Etapa III. Gobierno de Componentes

Construcción por Composición

Cómo construir soluciones combinando componentes y código pegamento de forma declarativa

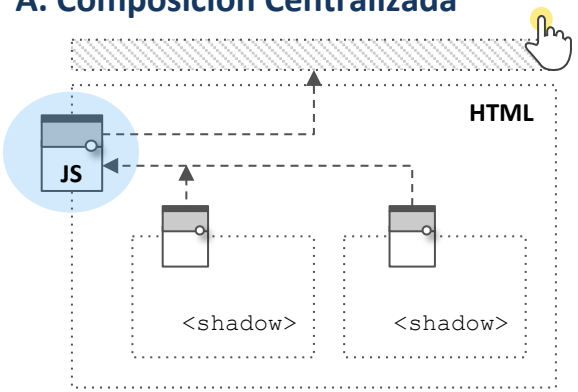
III. El Problema de Composición

Construcción por Composición

Construir por composición consiste en combinar estratégicamente lógica reutilizable en componentes Web con lógica compositiva específica.

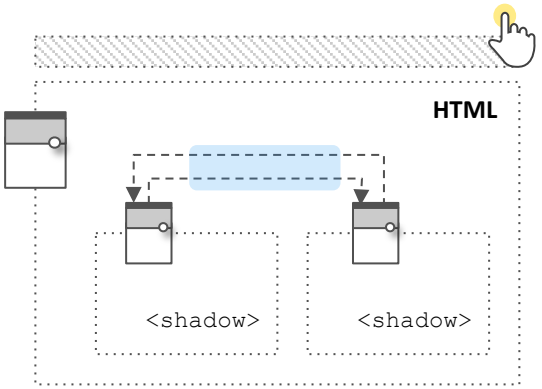
Dónde

A. Composición Centralizada



- Abstracción ✗
- Acoplamiento ✗
- Reutilización ✗
- Declaratividad ✗
- Compositividad ✗
- Productividad ✗

B. Composición Distribuida



- ✓ Abstracción
- ✓ Acoplamiento
- ✓ Reutilización
- ✓ Declarativdad
- ✓ Compositividad
- ✓ Productividad

III. El Problema de Composición

Construcción por Composición

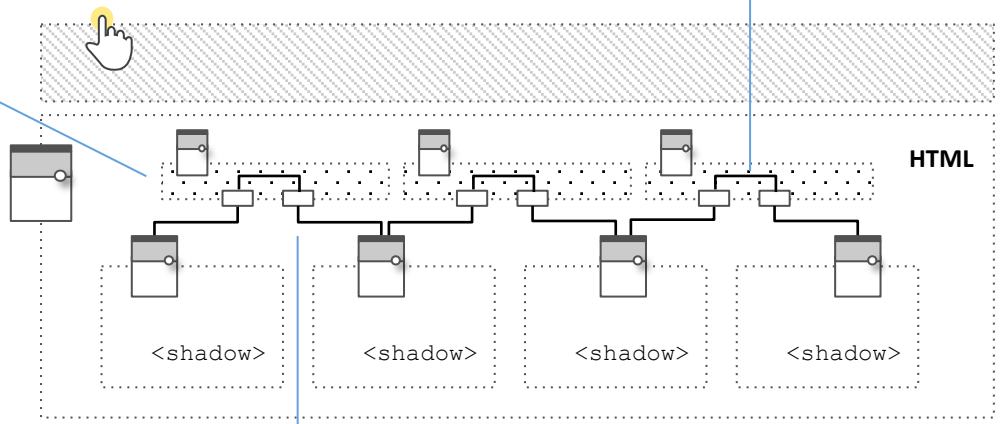
Construir por composición consiste en combinar estratégicamente lógica reutilizable en componentes Web con lógica compositiva específica

Flexibilidad compositiva

El comportamiento de los artefactos de composición es flexible y abierto

Encapsulación de Lógica Compositiva

Código pegamento recurrente también como componentes



Configuración declarativa

La composición como un ejercicio de configuración declarativa

Dónde

Etapa III. Gobierno de Componentes

Componentes Para Componer

Abstracción Reutilización

Declaratividad Compositividad

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

2

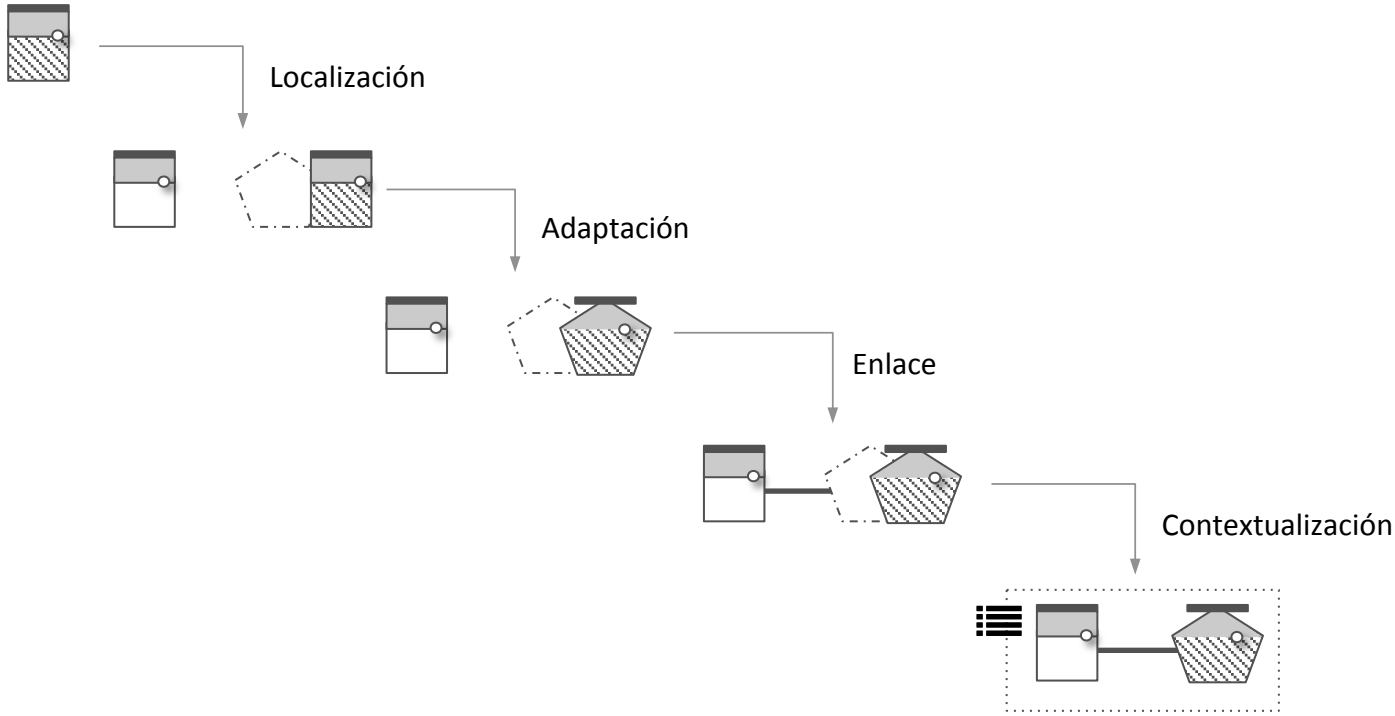
Patrones de Composición

- El Proceso de Composición como Marco Organizativo
- Patrones de Localización & Adaptación
- Patrones de Enlace & Contextualización

Patrones de Composición

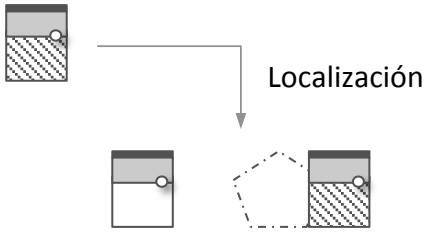
I. Patrones de Composición

Utilizaremos el proceso de composición como marco organizativo para ordenar los patrones que presentamos en este subsistema arquitectónico.



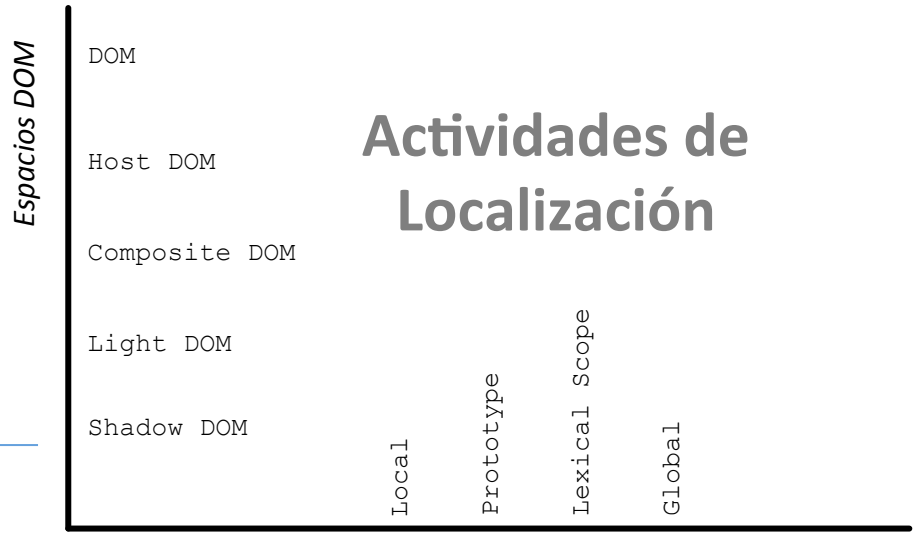
II. Patrones de Localización

Los patrones de localización exploran un espacio de búsqueda para localizar otros componentes de la vecindad con los que establecer vínculos compositivos.



Técnicas de Exploración

Muchos de los procesos de localización operan sobre espacios DOM donde residen otros componentes alojados allí en tiempo de diseño. Sobre ellos se aplican técnicas de búsqueda exploratorias



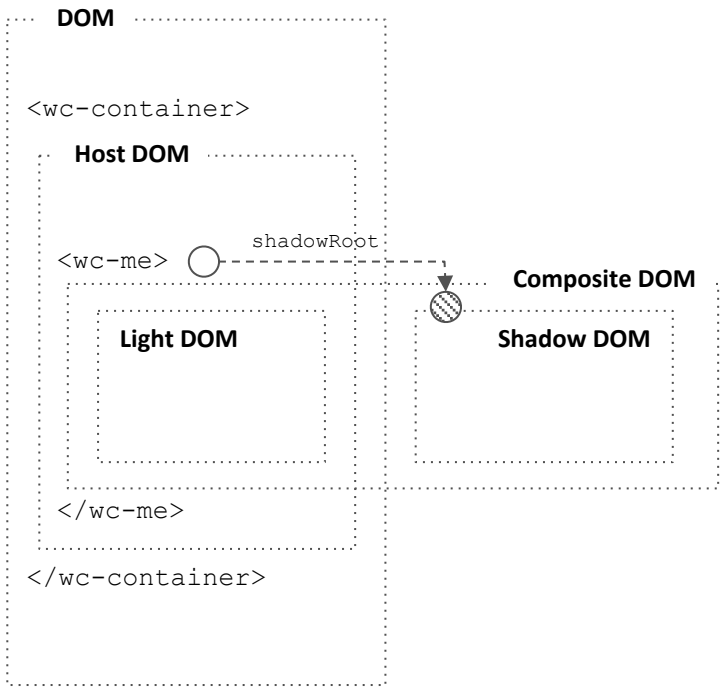
Técnicas de Look Up

Aunque menos utilizados, también deben considerarse aquellos espacios de JS donde pueden alojarse componentes activos en memoria. En este caso se usan técnicas de Look Up

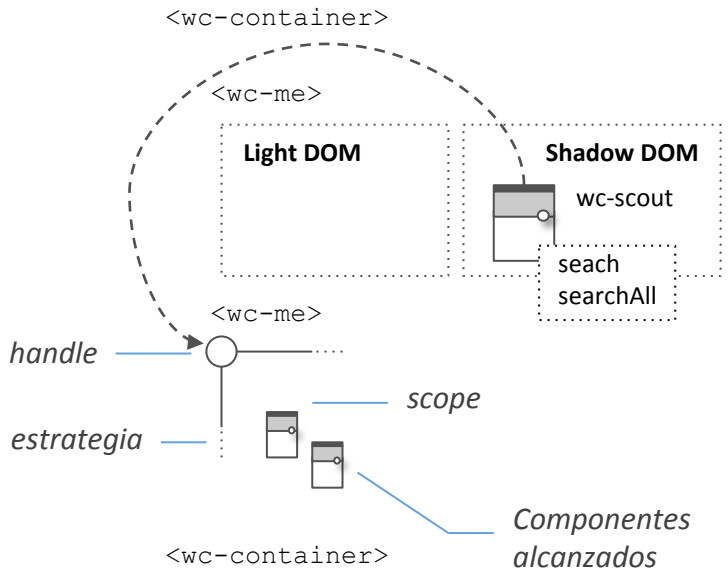
II. Patrones de Localización. Técnicas de Exploración

Las técnicas de exploración operan sobre los espacios DOM. Estás pueden clasificarse en localización o descubrimiento.

Espacios de Exploración



Técnicas de Exploración

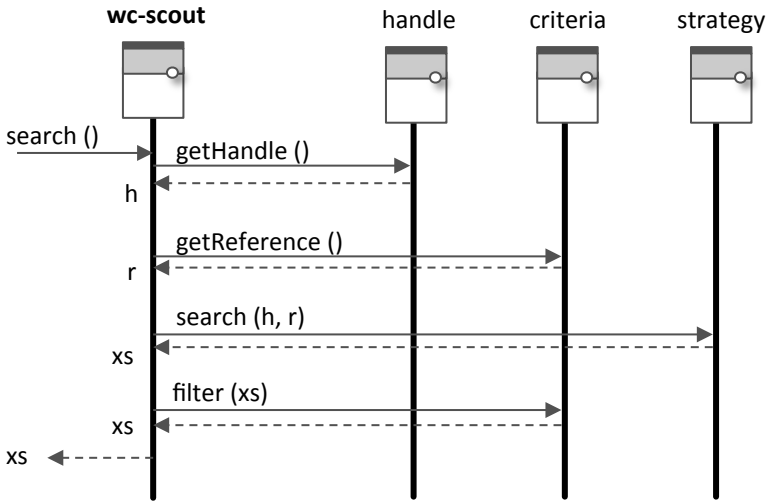


II. Patrones de Localización. Técnicas de Exploración

Explorador Scout

Se requiere llevar a cabo un proceso de exploración sobre un determinado ámbito DOM utilizando técnicas de exploración y criterios de selección específicos.

El componente wc-scout es responsable de articular un proceso de búsqueda exploratoria para buscar componentes en la vecindad.



El componente se configura indicando una estrategia de búsqueda, un espacio sobre el que buscar y un criterio de búsqueda.

```
<wc-scout result="{{out}}">
  <wc-x strategy/>
  <wc-y handle/>
  <wc-z criteria/>
</wc-scout>
```

Cada uno a un componente implementará cierta lógica de estrategia, alcance o criterio según corresponda como veremos a continuación

II. Patrones de Localización. Técnicas de Exploración

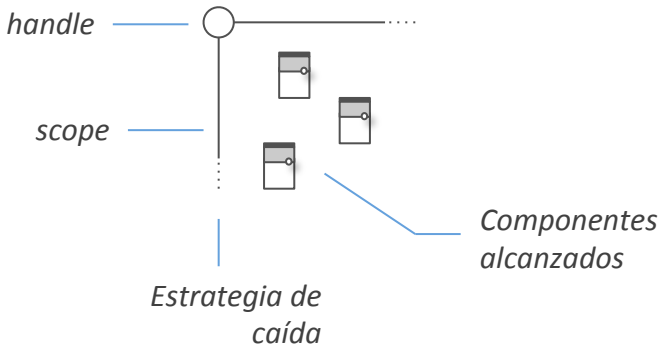
Estrategia de Caída

Se requiere llevar a cabo un proceso de exploración basado en una estrategia descendente para encontrar componentes bajo el handle.

Estrategias

El componente wc-falling implementa la estrategia de prospección clásica de la web basada en localizar elementos bajo un nodo.

El componente se configura indicando esencialmente si la estrategia debe hacer prospección en los contenidos en la sombra.




```
<wc-scout>
  <wc-falling strategy shadow/>
  <wc-y handle/>
  <wc-z criteria/>
</wc-scout>
```

La estrategia descendente busca elementos que caen bajo un determinado nodo DOM

Indica que la estrategia debe hacer prospección también en los contenidos en la sombra

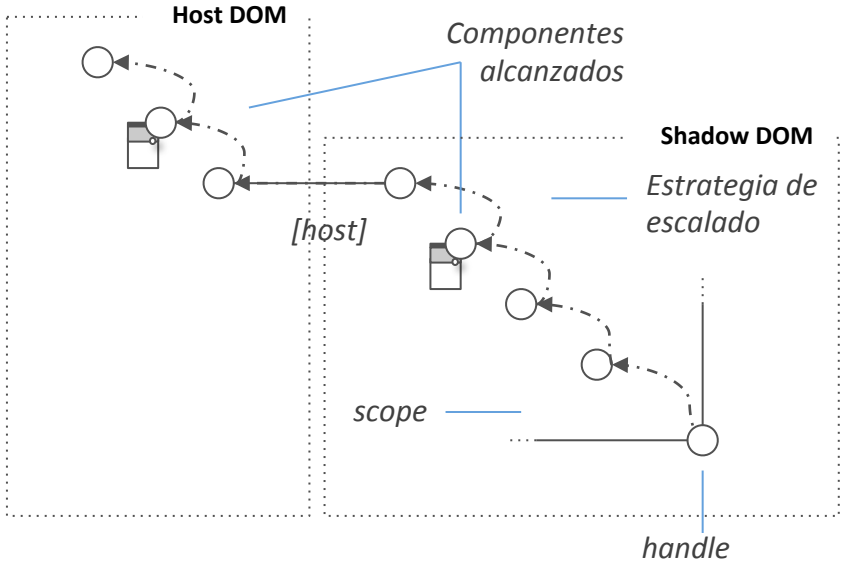
II. Patrones de Localización. Técnicas de Exploración

Estrategia de Escalado

Se requiere llevar a cabo un proceso de exploración basado en una estrategia ascendente para encontrar componentes en la cadena de antecesores del handle.

Estrategias

El componente wc-climbing implementa una estrategia de escalado para encontrar elementos en la cadena de antecesores.



El componente se configura indicando esencialmente si la estrategia debe continuar el escalado al llegar al nodo raíz.

```
<wc-scout>
  <wc-climbing strategy host/>
  <wc-y handle/>
  <wc-z criteria/>
</wc-scout>
```

La estrategia ascendente busca elementos que caen dentro de la cadena de antecesores de un determinado nodo DOM

Indica que la estrategia debe atravesar la raíz en la sombra y continuar ascendiendo por el nodo host

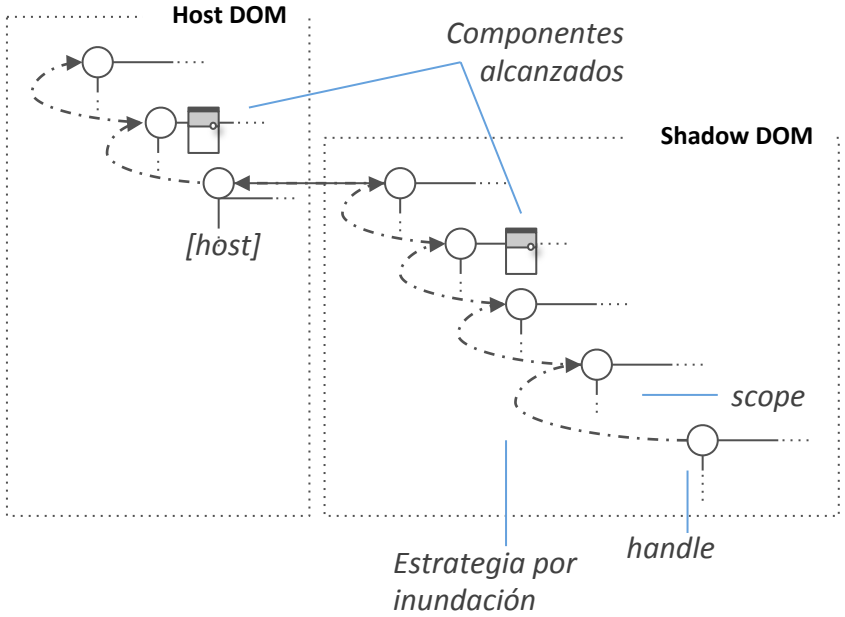
II. Patrones de Localización. Técnicas de Exploración

Estrategia por Inundación

Se requiere llevar a cabo un proceso de exploración basado en una estrategia por inundación para encontrar los componentes más próximos al handle.

Estrategias

El componente wc-flooding aplica una estrategia de búsqueda que alterna recursivamente el escalado y la caída.



El componente se configura indicando esencialmente si debe hacer prospección en la sombra y escalado hacia el host.

```
<wc-scout>
  <wc-flooding strategy
       host
       shadow/>
  <wc-y handle/>
  <wc-z criteria/>
</wc-scout>
```

La estrategia ascendente busca elementos que caen dentro de la cadena de antecesores de un determinado nodo DOM

Se utilizan los mismos flags que aparecían en las estrategias de caída y escalado

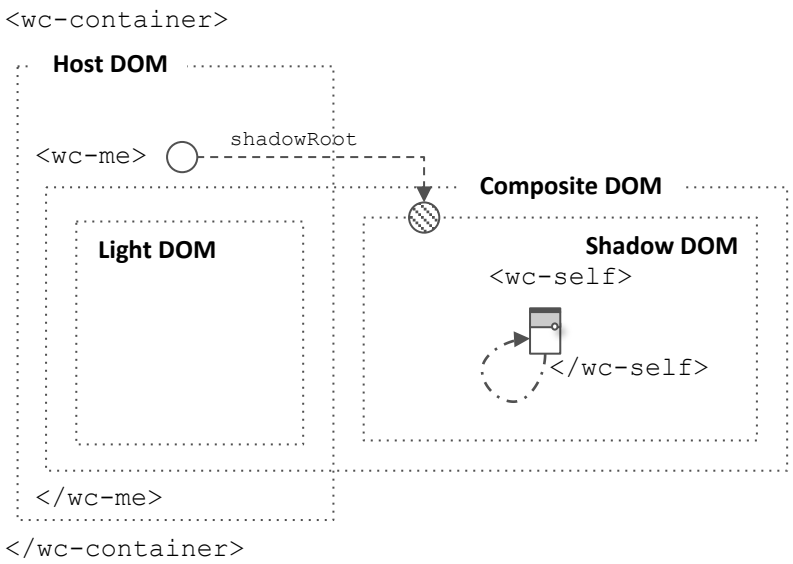
II. Patrones de Localización. Técnicas de Exploración

Handle de Auto-referencia

Handles

Se requiere identificar el handle correspondiente a la propia ubicación donde reside el componentes wc-scout. Es frecuente articular estrategias de búsqueda que comienzan en dicho lugar.

El componente wc-self identifica el elemento del DOM correspondiente al componente wc-scout.



La configuración del componente sólo requiere indicar el valor de la referencia CSS que será utilizada para referir al handle.

```
<wc-self/>
```

Referencia a la ubicación en curso

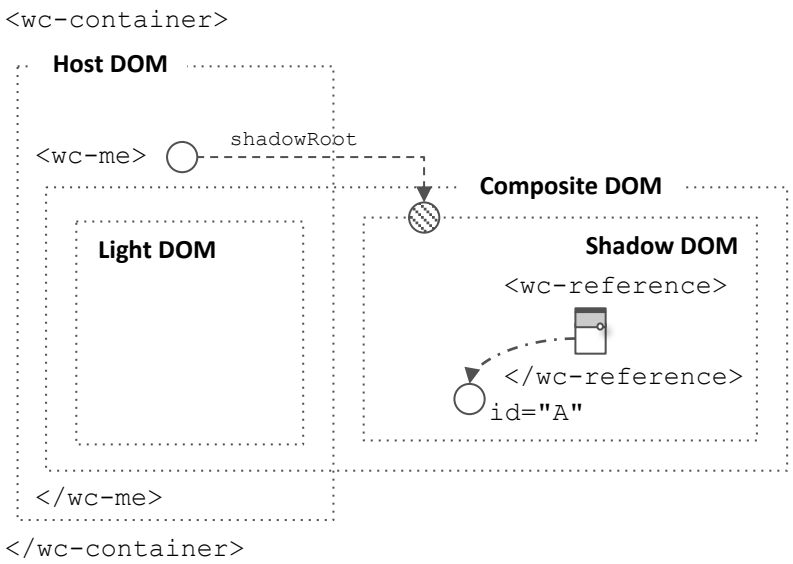
II. Patrones de Localización. Técnicas de Exploración

Handle Por Referencia

Handles

Se requiere identificar un handle a través del uso de una referencia CSS. Dicha referencia es relativa al punto donde se usa el componente.

El componente wc-reference identifica un elemento del DOM a través de una referencia CSS relativa a la posición de esta etiqueta.



La configuración del componente sólo requiere indicar el valor de la referencia CSS que será utilizada para referir al handle.

La referencia apunta al nodo handle

```
<wc-reference ref="#A" handle/>
<wc-reference ref="[[x]]" handle/>
```

La referencia es el nodo handle

II. Patrones de Localización. Técnicas de Exploración

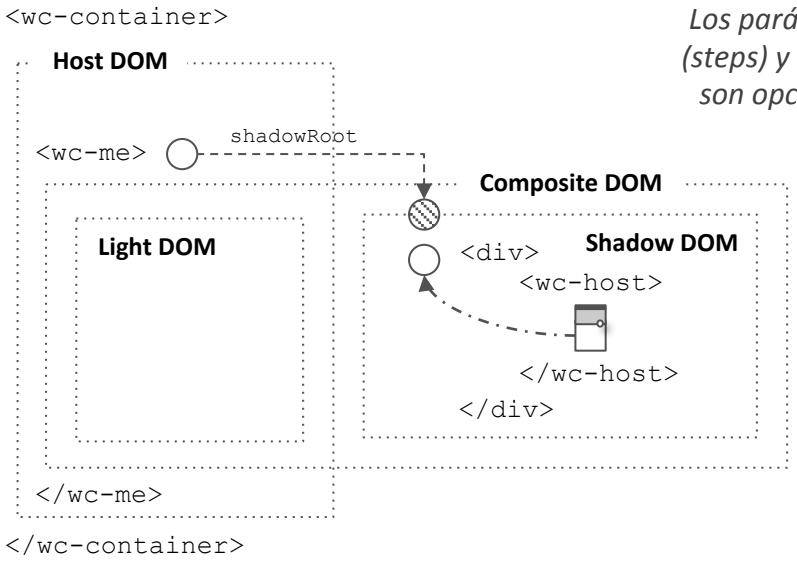
Handle Parent

Handles

Se requiere identificar el handle correspondiente al nodo padre desde el que realizamos la aplicación de estrategias de búsqueda por exploración.

El componente wc-parent identifica el handle que corresponde al padre directo de esta etiqueta.

La configuración indica el número de saltos ascendentes y el desplazamiento descendente que se realizará una vez localizado el host.



Los parámetros (steps) y (offset) son opcionales

```
<wc-parent handle
     steps="3"
     offset="#A"/>
```

Tras alcanzar el padre escala 3 nodos. Después, dirígete a #A para alcanzar el handle

II. Patrones de Localización. Técnicas de Exploración

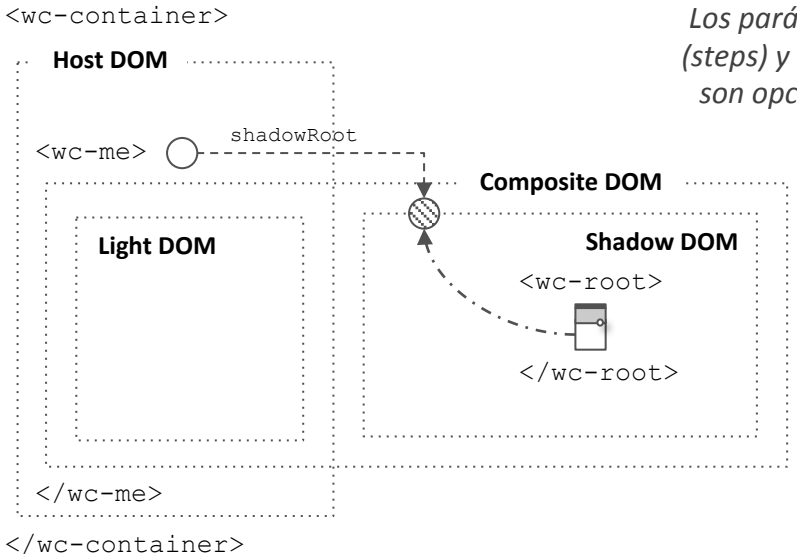
Handle Root

Handles

Se requiere identificar el handle correspondiente al nodo raíz del que cuelga el contenido en la sombra que corresponde al espacio DOM donde nos encontramos.

El componente wc-root identifica el handle que corresponde al nodo raíz que aloja el contenido en la sombra donde estamos.

La configuración indica el número de saltos ascendentes y el desplazamiento descendente que se realizará una vez localizado el host.



Los parámetros (steps) y (offset) son opcionales

```
<wc-root handle
     steps="3"
     offset="#A"/>
```

Tras alcanzar la raíz escala 3 nodos. Después, dirígete a #A para alcanzar el handle

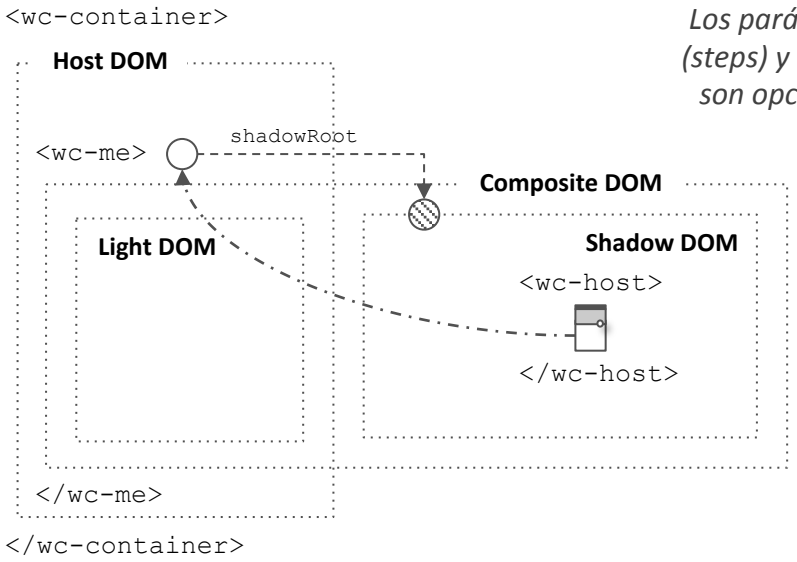
II. Patrones de Localización. Técnicas de Exploración

Handle Host

Se requiere identificar el handle correspondiente al nodo host que mantiene al contenido en la sombra que corresponde al espacio DOM donde nos encontramos.

Handles

- El componente wc-host identifica el handle que corresponde al nodo anfitrión que aloja el contenido en la sombra donde estamos.
- La configuración indica el número de saltos ascendentes y el desplazamiento descendente que se realizará una vez localizado el host.



Los parámetros (steps) y (offset) son opcionales

```
<wc-host handle
     steps="3"
     offset="#A"/>
```

Tras alcanzar el host escala 3 nodos. Después, dirígete a #A para alcanzar el handle

II. Patrones de Localización. Técnicas de Exploración

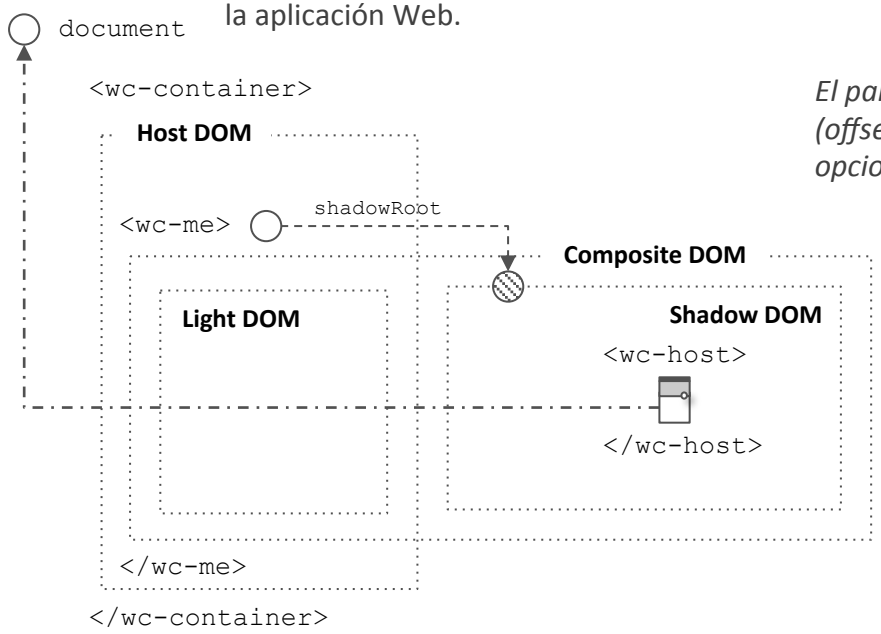
Handle Document

Se requiere identificar el handle correspondiente al nodo principal del documento HTML desde el que realizamos la exploración.

Handles

El componente wc-document identifica el handle que corresponde al documento raíz de la aplicación Web.

La configuración indica el desplazamiento descendente que se realizará una vez localizado el nodo document.



El parámetro (offset) es opcionales

```
<wc-document handle
    offset="#A"/>
```

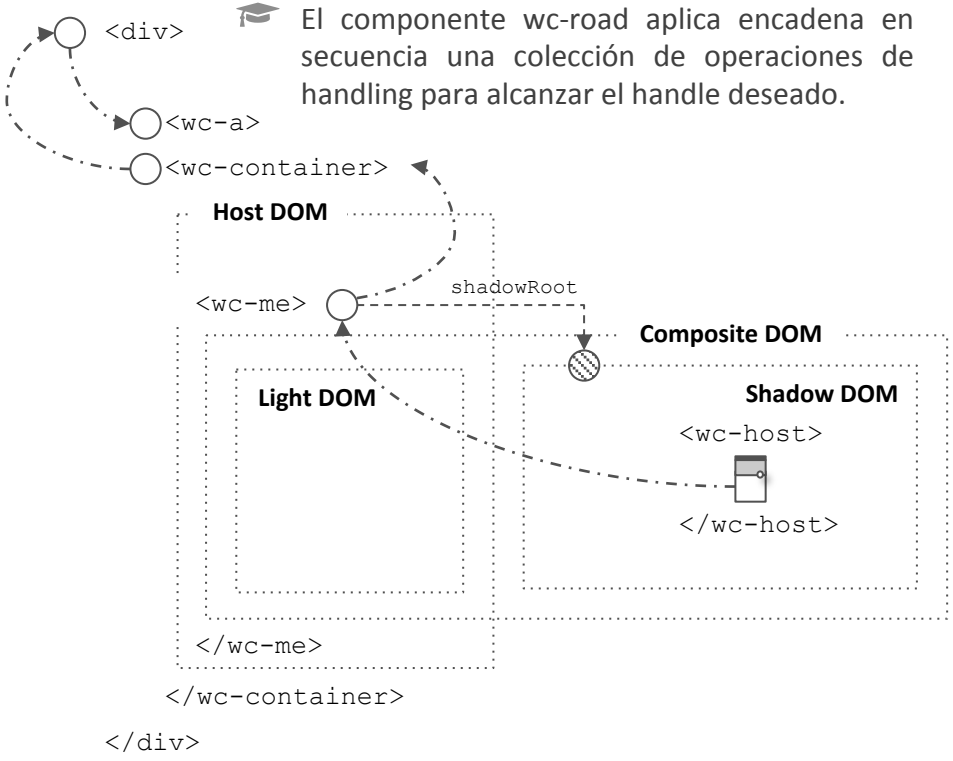
Tras alcanzar document dirígete a #A para alcanzar el handle

II. Patrones de Localización. Técnicas de Exploración

Handle Road

Para localizar un handle se requiere llevar a cabo un proceso de recorrido por pasos sobre la estructura de espacios DOM. En cada paso se aplica un handle de los anteriores.

Handles



El componente wc-road aplica encadena en secuencia una colección de operaciones de handling para alcanzar el handle deseado.

</> El componente se configura indicando la colección de operaciones de handling que deben aplicarse para llegar al handle final.

```
<wc-road handle>
     <wc-host/>
     <wc-host/>
     <wc-parent offset="#A"/>
</wc-road>
```

Cada paso de handling se aplica sobre el resultado que arrojo la operación de handling anterior en la cadena. Ejemplo:

- *Primero, alcanza el host*
- *Una vez en el, alcanza el host*
- *Ahora sube al padre*
- *Desplazate hasta localizar #A*

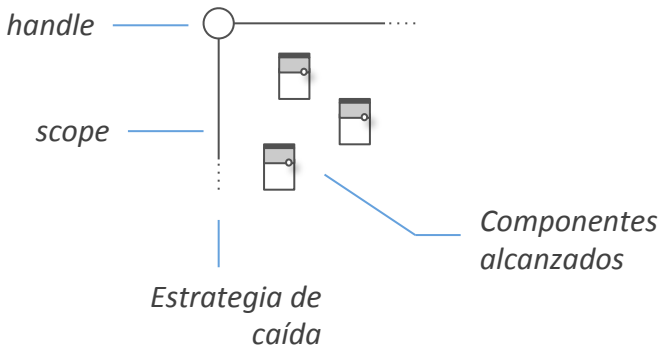
II. Patrones de Localización. Técnicas de Exploración

Búsqueda Por Consulta

Criteria

Se requiere aplicar un criterio de búsqueda clásico basado en el uso de referencias CSS. La consulta será relativa al handle y se aplicará la estrategia establecida.

El componente wc-query, permite definir un criterio de búsqueda basado en el uso de referencias CSS.



El componente se configura indicando el criterio de búsqueda que se desea aplicar sobre el espacio.

```
<wc-scout>
  <wc-x strategy/>
  <wc-y handle/>
  <wc-query criteria
        ref=".A"/>
</wc-scout>
```

Sobre el handle y de acuerdo a la estrategia de exploración establecida, busca a #A

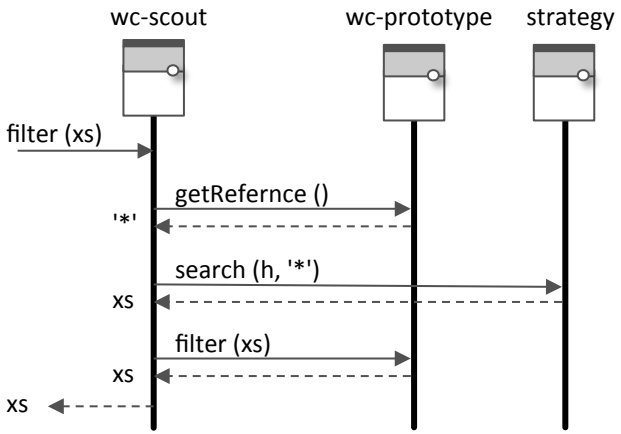
II. Patrones de Localización. Técnicas de Exploración

Búsqueda Por Prototipo

Se requiere localizar componentes que respondan a un determinado prototipo JavaScript. Este tipo de búsqueda ofrece garantías de un cierto comportamiento y características.

Criteria

El componente wc-prototype devuelve la referencia universal (*) al scout. Después aplica técnicas de filtro para seleccionar a los candidatos adecuados.



El componente se configura indicando esencialmente el tipo o tipos de prototipo que se desea buscar.

Sintaxis de búsqueda única

```
<wc-scout>
  <wc-x strategy/>
  <wc-y handle/>
  <wc-prototype criteria
        type="wcA"/>
</wc-scout>
```

Sintaxis de búsqueda disyuntiva

```
<wc-scout>
  <wc-x strategy/>
  <wc-y handle/>
  <wc-prototype criteria
        in="wcA wcB wcC"/>
</wc-scout>
```

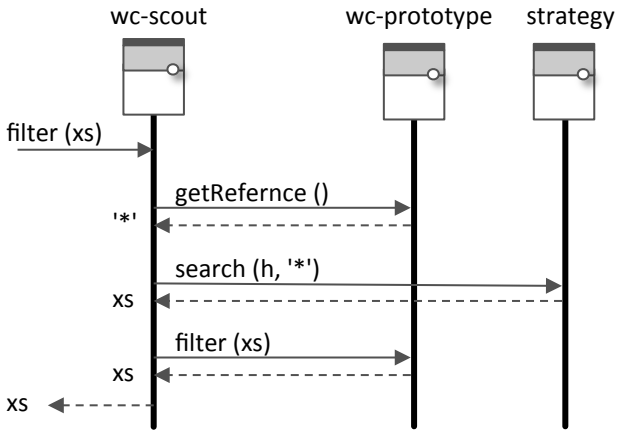
II. Patrones de Localización. Técnicas de Exploración

Búsqueda Por Perfil

Criteria

Se requiere localizar componentes que respondan a un determinado perfil JavaScript. Este tipo de búsqueda ofrece garantías de un cierto comportamiento y características.

El componente wc-prototype devuelve la referencia universal (*) al scout. Después aplica técnicas de filtro para seleccionar a los candidatos adecuados.



El componente se configura indicando esencialmente el perfil buscado, como una colección de características

Sintaxis dinámica como array de características

```
<wc-scout>
  <wc-x strategy/>
  <wc-y handle/>
  <wc-profile criteria
       profile="{{pf}}"/>
</wc-scout>
```

Sintaxis estática con enumeración explícita

```
<wc-scout>
  <wc-x strategy/>
  <wc-y handle/>
  <wc-profile criteria>
     <wc-rule feature="p"/>
     <wc-rule feature="q"/>
  </wc-profile>
</wc-scout>
```

II. Patrones de Localización. Técnicas de Look Up

Las técnicas de Look up articulan procesos de búsqueda sobre los espacios de memoria que gestiona JS. Se distinguen entre servicios de páginas blancas y amarillas.

Espacios de Look Up

```
Global
Polymer (
  (function () {
      Retención Léxica
      return {
         is : 'wc-me'
         ready : function () {
            this.prototype.y = prototipo
            this.x = Local
         }
      };
  })();
);
```

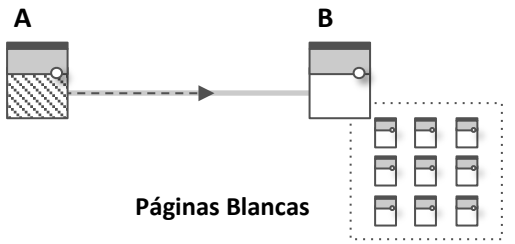
Acceso global

Acceso por tipo

Acceso individual

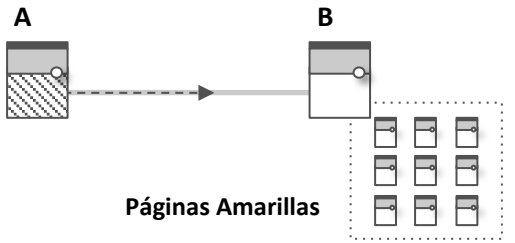
Técnicas de Look Up

Se busca un componente por clave de registro



P. Blancas

- A <wc-a>
- B <wc-b>
- C <wc-c>



P. Amarillas

- [p, q] <wc-a>
- [q] <wc-b>
- [r, s] <wc-c>

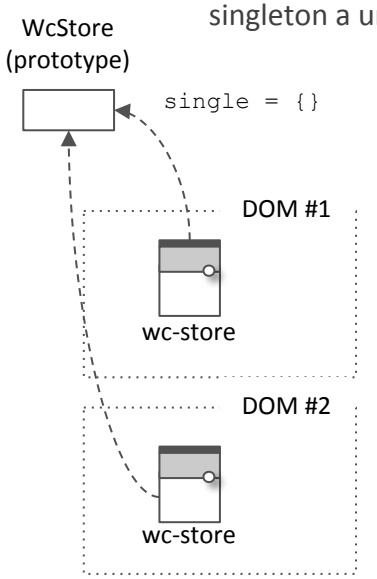
Se buscan componentes por metadatos de capacidades y servicios

II. Patrones de Localización. Técnicas de Look Up

Acceso Único

Se requiere disponer de un mecanismo sencillo para compartir datos con agnosticismo de su disposición geográfica en los espacios DOM.

El behavior wc-single proporciona una forma sencilla de proporcionar acceso en lógica singleton a un almacén de datos.




```
Polymer ({
   is : 'wc-store',
   behaviors : [
      Behaviors ['wc-single']
   ],
   init : function (single) {
     single = {}
     ...
   }
});
```

Se declara una variable (single) a nivel de prototipo para que sea compartida por todas las instancias del mismo tipo de componente.

```
Behaviors = Behaviors || {};
Behaviors ['wc-single'] = {
   ready : function () {
      this.init (this.prototype.single);
   }
};
```

La variable se pasa como parámetro a un método init para su inicialización

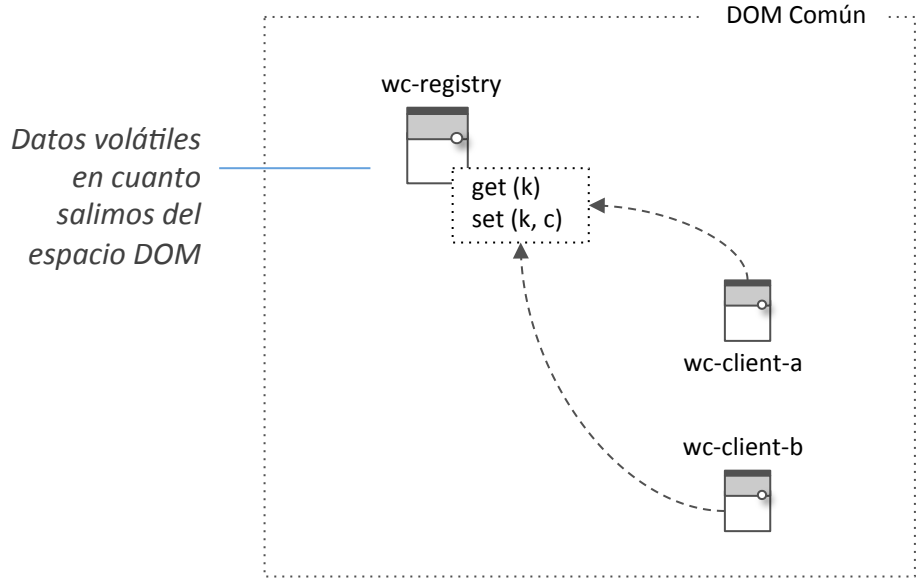
Se registra en el prototipo una variable (single)

II. Patrones de Localización. Técnicas de Look Up

Registro de Componentes

Se requiere disponer de un mecanismo para registrar y recuperar componentes con agnosticismo de su disposición geográfica en los espacios DOM.

El componente wc-registry proporciona una diccionario de datos para realizar tareas de registro y recuperación esenciales



El componente se configura indicando entradas por defecto para la configuración del registro.

```
<wc-registry>
  <wc-rule key="A" target="{{c1}}"/>
  <wc-rule key="B" target="{{c2}}"/>
  <wc-rule key="C">
     <wc-c></wc-c>
  </wc-rule>
 <wc-rule key="C">
     <wc-reference ref="#D"/>
  </wc-rule>
</wc-registry>
```

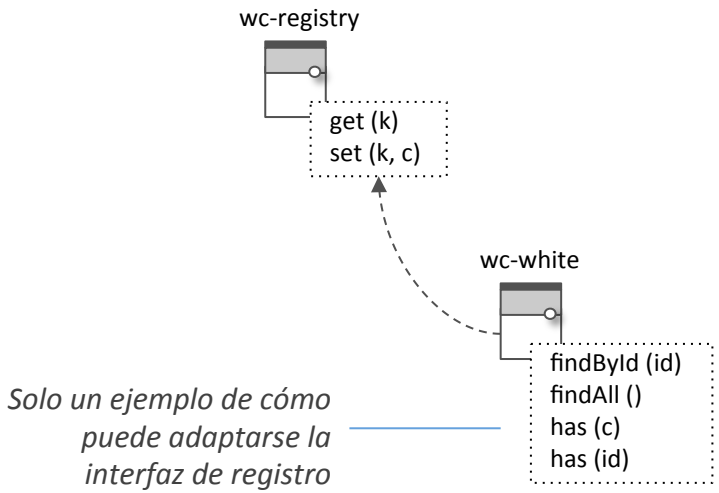
Distintas formas de introducir entradas por defecto en un registro de componentes

II. Patrones de Localización. Técnicas de Look Up

Perfil de Páginas Blancas

Se requiere disponer de un almacén de componentes para registrar y recuperar componentes con facilidades de acceso de páginas blancas.

El componente wc-whites es una interfaz de acceso al registro que proporciona al cliente capacidades de páginas blancas.



Solo un ejemplo de cómo puede adaptarse la interfaz de registro

El componente se configura indicando quién es el registro de componentes que se utiliza sobre el que se opera.

```
<wc-white registry="[[reg]]">
</wc-white>
```

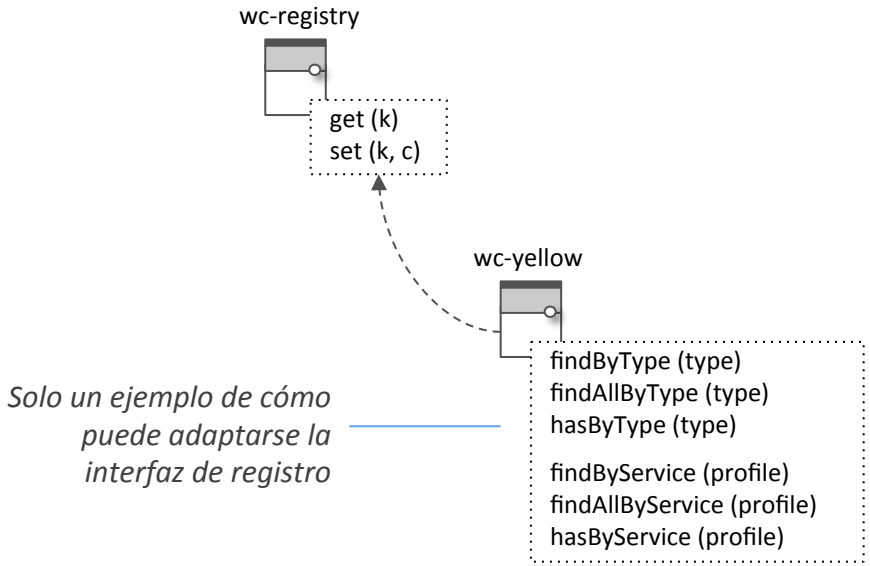
Se supone acceso al registro. El componente solo es un adaptador que proporciona una interfaz apropiada para dar servicios de páginas blancas

II. Patrones de Localización. Técnicas de Look Up

Perfil de Páginas Amarillas

Se requiere disponer de un almacén de componentes para registrar y recuperar componentes con facilidades de acceso de páginas amarillas.

El componente wc-yellows es una interfaz de acceso al registro que proporciona al cliente capacidades de páginas amarillas.



Solo un ejemplo de cómo puede adaptarse la interfaz de registro

El componente se configura indicando quién es el registro de componentes que se utiliza sobre el que se opera.

```
<wc-yellow registry="[[reg]]">
</wc-yellow>
```

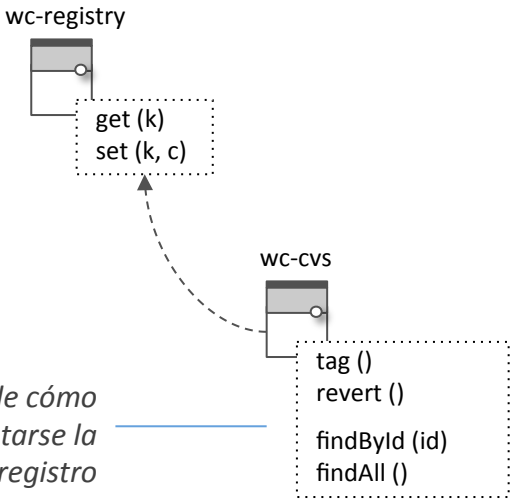
Se supone acceso al registro. El componente solo es un adaptador que proporciona una interfaz apropiada para dar servicios de páginas blancas

II. Patrones de Localización. Técnicas de Look Up

Perfil de Control de Versiones

Se requiere disponer de un almacén de componentes para registrar y recuperar componentes con facilidades de control de versiones

El componente wc-cvs es una interfaz de acceso al registro que proporciona al cliente capacidades de control de versiones.



Solo un ejemplo de cómo puede adaptarse la interfaz de registro

El componente se configura indicando quién es el registro de componentes que se utiliza sobre el que se opera.

```
<wc-cvs registry="[[reg]]">
</wc-cvs>
```

Se supone acceso al registro. El componente solo es un adaptador que proporciona una interfaz apropiada para dar servicios de páginas blancas

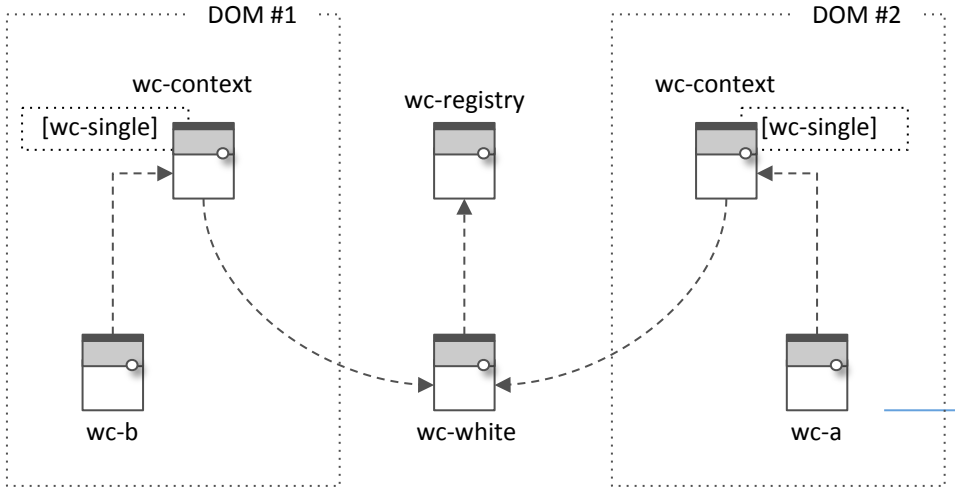
II. Patrones de Localización. Técnicas de Look Up

Contexto de Componentes

Se requiere proporcionar un mecanismo para acceder de manera sencilla a un único registro de componentes en todo el espacio de memoria de la aplicación.

El componente wc-context proporciona acceso unitario a un registro de componentes. Para ello utiliza el hehavior wc-single.

El componente fabrica su propia instancia de registro de manera interna y transparente y ofrece servicio de paginas blancas o amarillas según se configure.




```
<wc-context
     registry="[[white]]">
</wc-context>
```

La configuración del registro sólo es necesaria hacerla una vez

El componente de contexto adquiere los métodos de acceso expuestos por wc-white de manera que para el cliente es transparente la existencia de infraestructura por detrás

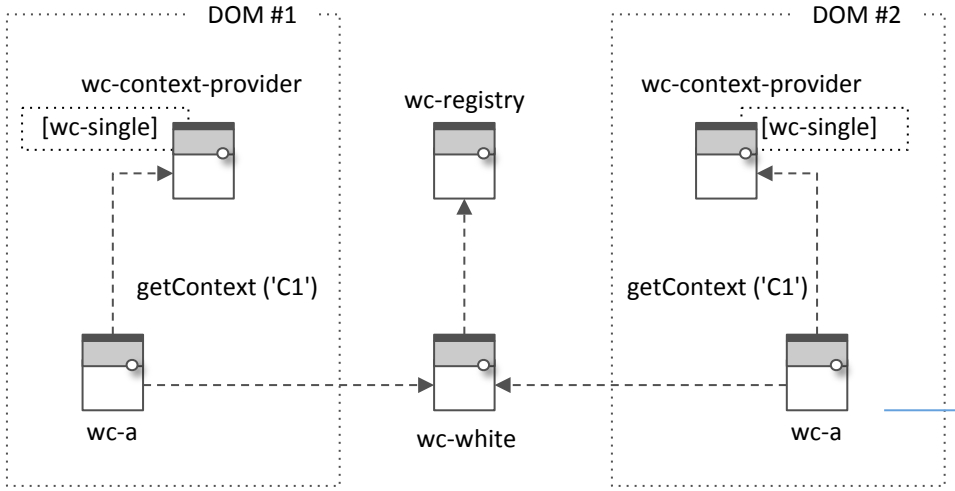
II. Patrones de Localización. Técnicas de Look Up

Proveedor de Contexto

Se requiere proporcionar un mecanismo para acceder de manera sencilla a una colección de registros de componentes en todo el espacio de memoria de la aplicación.

El componente wc-context-provider da acceso a una colección de contextos diferentes organizados en un espacio de nombres.

El componente se configura indicando la colección de contextos que deben ser incluidos por defecto en el entorno.



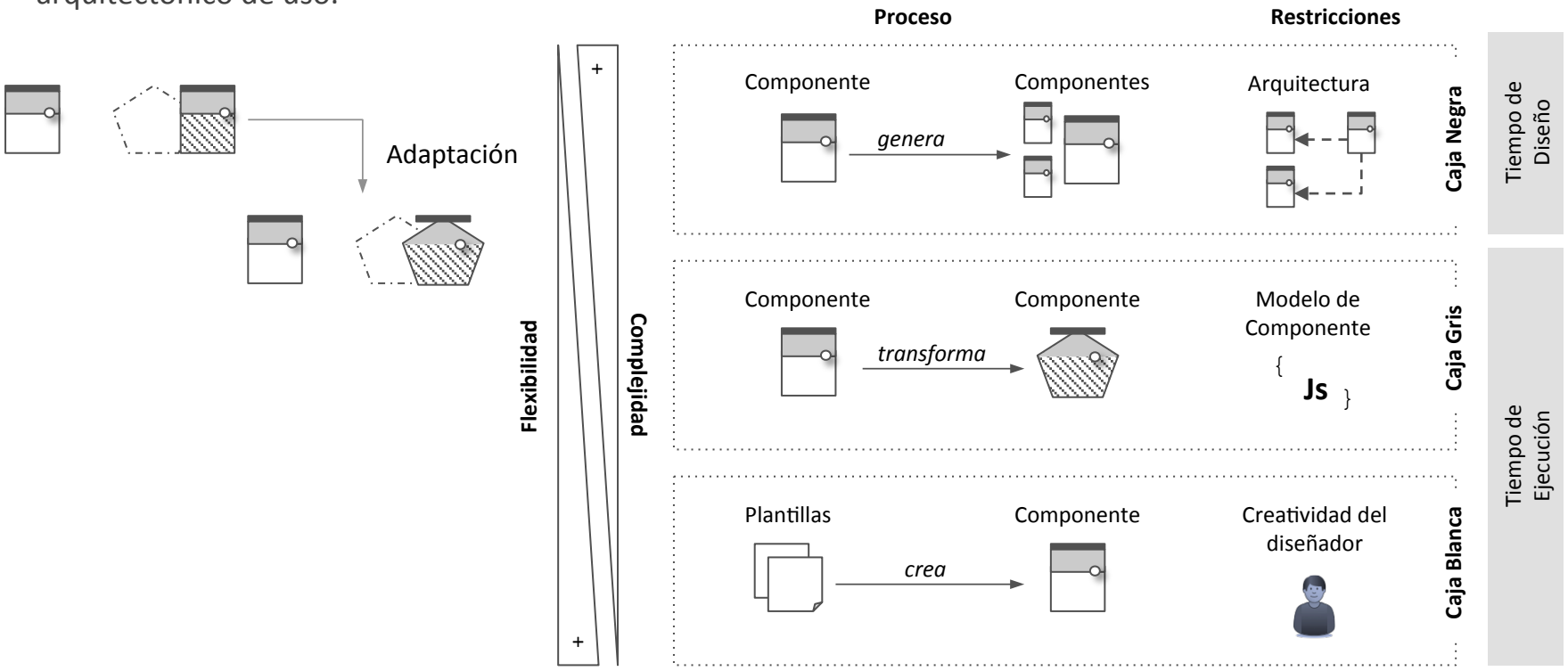

```
<wc-context-provider registry="[[w]]">
  <wc-rule key="C1" context="[[c1]]"/>
  <wc-rule key="C1" context="[[c2]]"/>
  <wc-rule key="C1" context="[[c3]]"/>
</wc-context-provider>
```

La configuración del registro sólo es necesaria hacerla una vez

El componente de contexto adquiere los métodos de acceso expuestos por wc-white de manera que para el cliente es transparente la existencia de infraestructura por detrás

III. Patrones de Adaptación

Los patrones de adaptación realizan transformaciones sobre los componentes para que encajen en el contexto arquitectónico de uso.

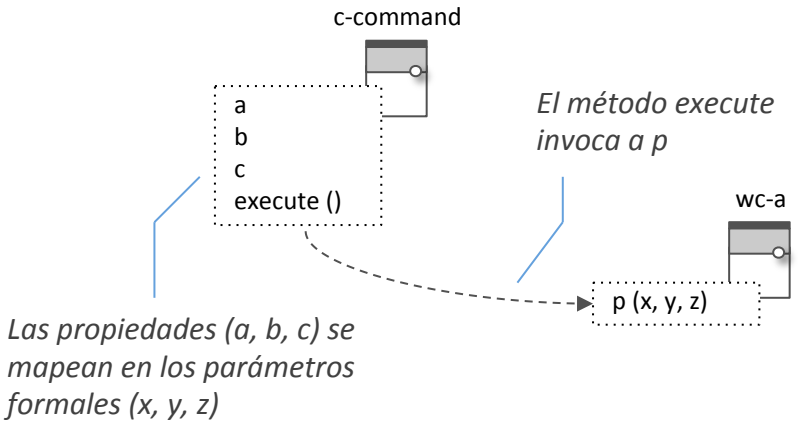


III. Patrones de Adaptación

Orientación a Comando

Se requiere adaptar un método de un componente para que no use parámetros en su llamada y así pueda ser invocado asíncronamente como un comando por otros componentes.

El componente wc-comand transforma su contrato para recibir los parámetros del método como propiedades y genera un método (execute) que realice la invocación.



La configuración recibe el componente objetivo (target) y el método sobre el que se desea generar el comando (method).

Se indica el componente y el método objetivo

```
<wc-command target="#C" method="m">
  <wc-rule key="a"/>
  <wc-rule key="b"/>
  <wc-rule key="c"/>
</wc-command>
```

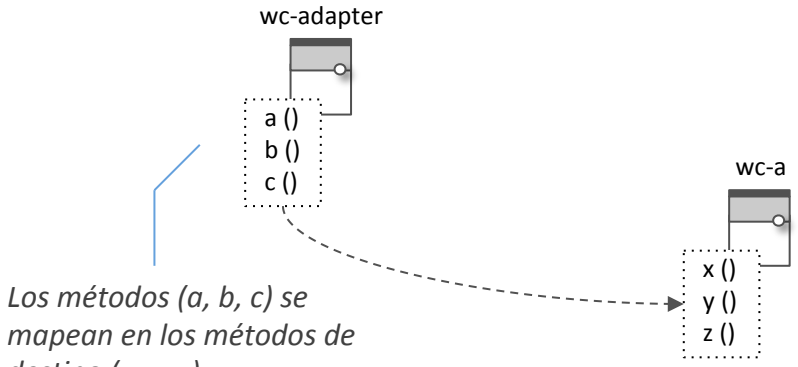
Las reglas indican por orden posicional en qué atributo HTML se mapea cada parámetro

III. Patrones de Adaptación

Adaptador de Contrato

Se requiere adaptar el contrato de un componente de manera que pueda ser explotado en el marco de un nuevo contexto arquitectónico de uso.

El componente wc-adapter adapta su contrato para crear nuevos métodos de alias sobre aquellos que existen en el mismo.



Los métodos (a, b, c) se mapean en los métodos de destino (x, y, z)

El componente se configura indicando esencialmente el perfil buscado, como una colección de características

El target hace referencia al componente sobre el que se aplica la adaptación

```
<wc-adapter target="#A">
  <wc-rule key="x" as="a"/>
  <wc-rule key="y" as="b"/>
  <wc-rule key="z" as="c"/>
</wc-adapter>
```

Las reglas indican la lógica de transformación que debe aplicarse

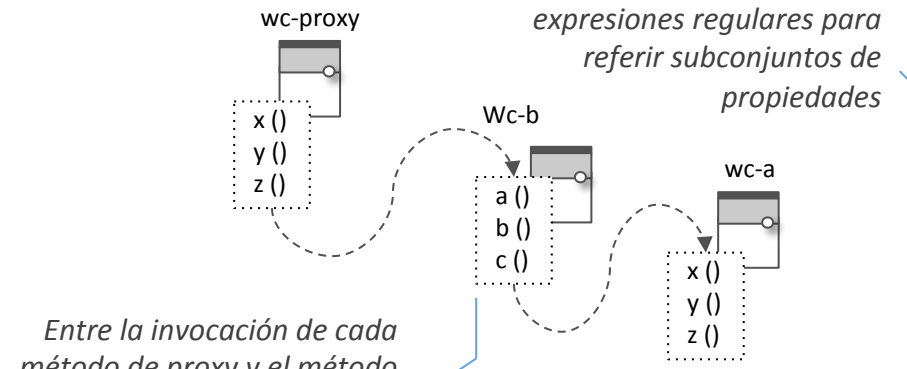
III. Patrones de Adaptación

Proxy de Contrato

Se requiere generar un nuevo componente que inyecte lógica de intercesión para que se ejecute en algún momento durante la invocación de los métodos de un componente.

El componente wc-proxy se encarga de realizar operaciones de intercesión para que ejecuten operaciones antes, durante o después de los métodos de un componente

El componente recibe el target sobre el que se hace la intercesión y el componente que contiene el código a inyectar. Las reglas dan detalles sobre la inyección.



Las claves (key) pueden usar expresiones regulares para referir subconjuntos de propiedades

Entre la invocación de cada método de proxy y el método original se interpone el código de decoración (a, b, c)

Se indica el componente objetivo y el que contiene los aspectos a inyectar

```
<wc-proxy target="#A" aspect="#B">
  <wc-rule key="x" with="a" before/>
  <wc-rule key="y" with="b" after/>
  <wc-rule key="z" with="c" around/>
</wc-proxy>
```

Las reglas indican qué métodos del aspecto deben ejecutarse (with) cuando se invoca cada método del target (key)

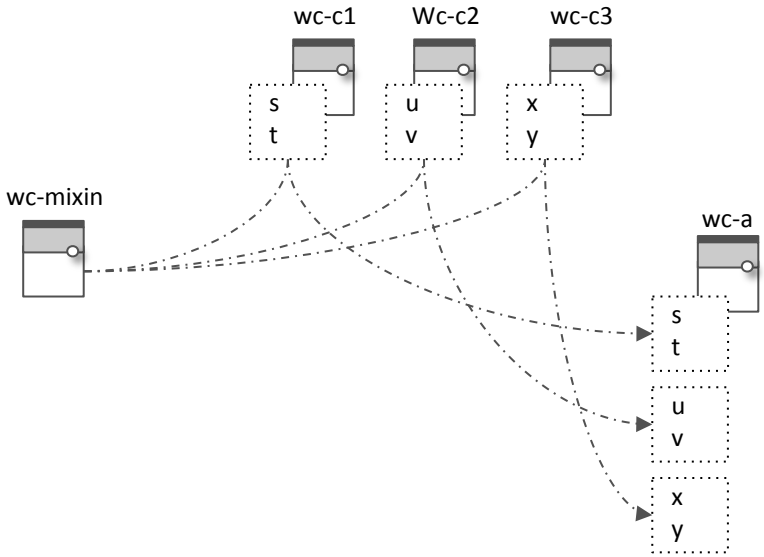
Se indica cuando se ejecuta cada decoración. Antes (before), durante (around) o después (after)

II. Patrones de Localización. Técnicas de Look Up

Mixin de Contratos

Se requiere alterar el contrato de un componente por medio de la adquisición dinámica de nuevas características definidas en los contratos de otros componentes.

El componente wc-mix da respuesta a esta necesidad de manera completamente declarativa.



El componente recibe en base a reglas de configuración cada uno de los componentes que hay mezclar.

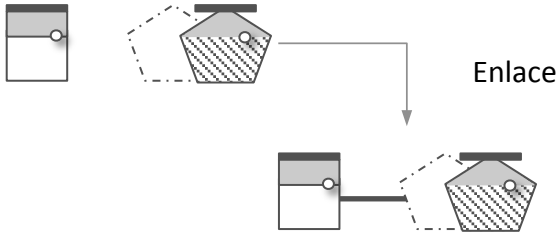
Se indica el componente objetivo y el que contiene los aspectos a inyectar

```
<wc-mix target="#A" policy="overide">
  <wc-rule with="#C1"/>
  <wc-rule with="#C2"/>
  <wc-rule with="#C3"/>
</wc-mix>
```

Las políticas indican cómo proceder cuando existen problemas de colisión de claves

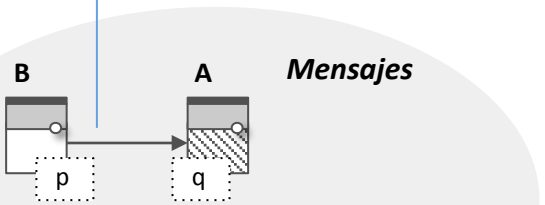
IV. Patrones de Enlace

Los patrones de enlace se encuentran estrechamente condicionados con la forma de comunicación que pretendamos establecer.



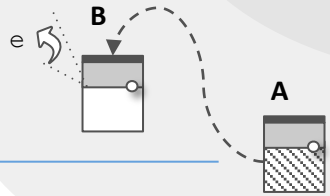
Enlace por Mensajes

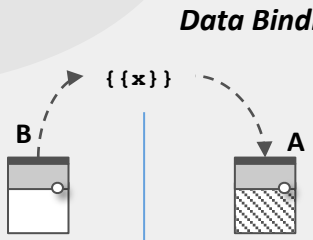
La composición en este caso enlaza el comportamiento de A y B invocando a m desde n



Enlace por Eventos

La composición aquí consiste en registrar a A como escuchador de los eventos de tipo e que emite B





Enlace por Datos

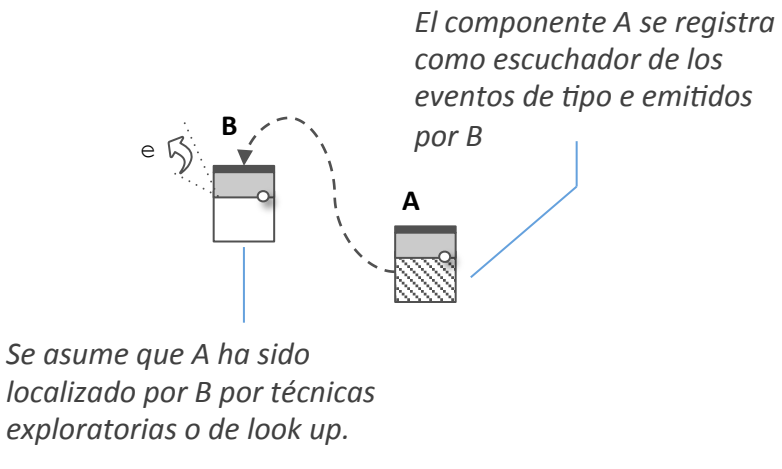
Dado que B tiene permiso para escribir en x, la composición aquí consiste en registrar a A como interesado en los cambios de x

IV. Patrones de Enlace

Enlace Por Eventos

Se requiere realizar un enlace compositivo por eventos para que un componente responda reactivamente ante cierto tipo de eventos emitido por otro componente.

El componente objetivo debe registrarse como escuchador de los eventos de cierto tipo emitidos por el componente emisor.



La lógica de composición encapsula un registro de eventos convencional utilizando los mecanismos de la Web.

```
function bind (self) {
  return {
    eventBind : function (binding) {
      var {target, on} = binding;
      target.addEventListener (
                on, self.execute);
    }
    ...
  };
}
```

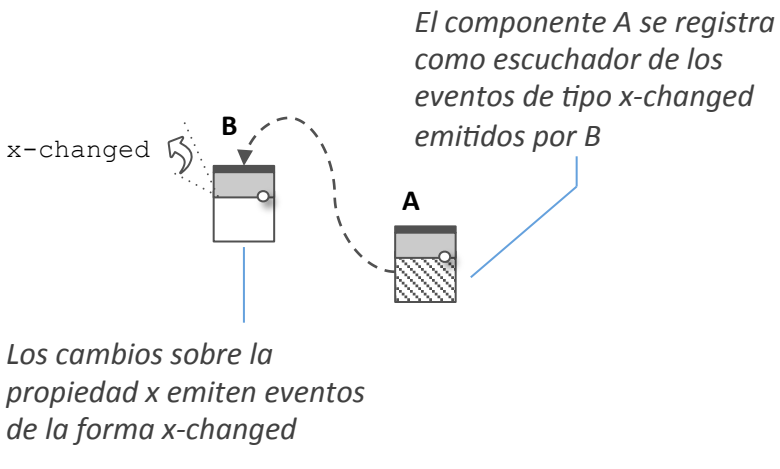
Por convenio, el manejador es un método (execute) que reside el cliente donde se implementa toda la lógica reactiva

IV. Patrones de Enlace

Enlace Por Datos

Se requiere realizar un enlace compositivo por datos para que un componente responda actualizando una de sus propiedades al valor adquirido por otra propiedad en otro componente.

El componente objetivo debe registrarse como escuchador de cambios de propiedades mientras que el emisor debe configurar su propiedad notify a true.



La lógica de composición consisteahora en escuchar eventos de cambio y proceder actualizando una de sus propiedades.

```
function bind (self) {
  return {
    dataBind : function (binding) {
      var {target, from, to} = binding;
      var on = from + '-changed';
      target.addEventListener (on,
        function (e, ctx) {
          self[to] = ctx.value;
        }
      );
    }
    ...
  };
}
```

La reacción consiste en actualizar la variable destino cuando cambia la de origen

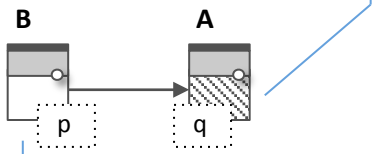
IV. Patrones de Enlace

Enlace Por Mensajes

Se requiere realizar un enlace compositivo por mensajes para que la llama a un método de un componente desencadene la invocación de otro método del componente objetivo.

El componente objetivo debe registrar una intercesión funcional sobre el método de otro componente para que en algún punto invoque a uno de sus métodos.

El método q de A en before es normal, en after debe esperar un parámetro correspondiente al resultado. En around, uno correspondiente a la propia función p



Al invocar al método p de B se pretende que se ejecute el método q de A. Esta intercesión resulta transparente para B

Existen 3 tipos de estrategias de intercesión sobre un método. Before, after y around que llaman, respectivamente, al objetivo antes, después y durante la ejecución de dicho método.

```
function bind (self) {
 return {
  messageBind : function (binding) {
   var {target, on, to, when} = binding;
   var fn = self[to];
   target[on] = function (...args) {
    if (when === BEFORE) fn.apply(self, args);
    var r = target[on].apply (target, args);
    if (when === AFTER) fn.apply(self. [...args, r]);
   }
  }
 }
}
```

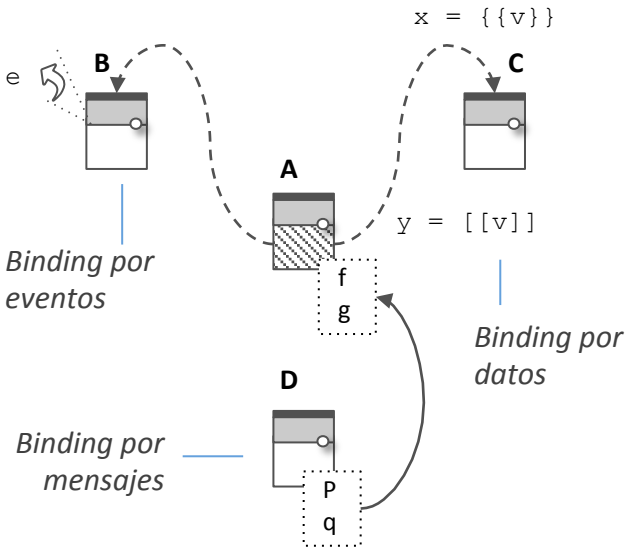
Detalles en 'Metaprogramación en JavaScript'. JSDay 2015

IV. Patrones de Enlace

Enlace Externo

Se requiere proporcionar un mecanismo declarativo para definir todos los tipos de enlaces compositivos que presenta un enlace con su vecindad.

El componente wc-bind permite cubrir esta necesidad articulando los 3 tipos de enlace compositivo anteriores.



Las reglas de configuración determinan qué enlaces hay que hacer con cada componente de la vecindad. (for) marca el componente objetivo.

```
<wc-bind for="A">
  <wc-rule event   with="B" on="e"/>
  <wc-rule data    with="C" from="x" to="y"/>
  <wc-rule message with="D" on="q" to="f" before/>
  <wc-rule message with="D" on="q" to="g" after/>
</wc-bind>
```

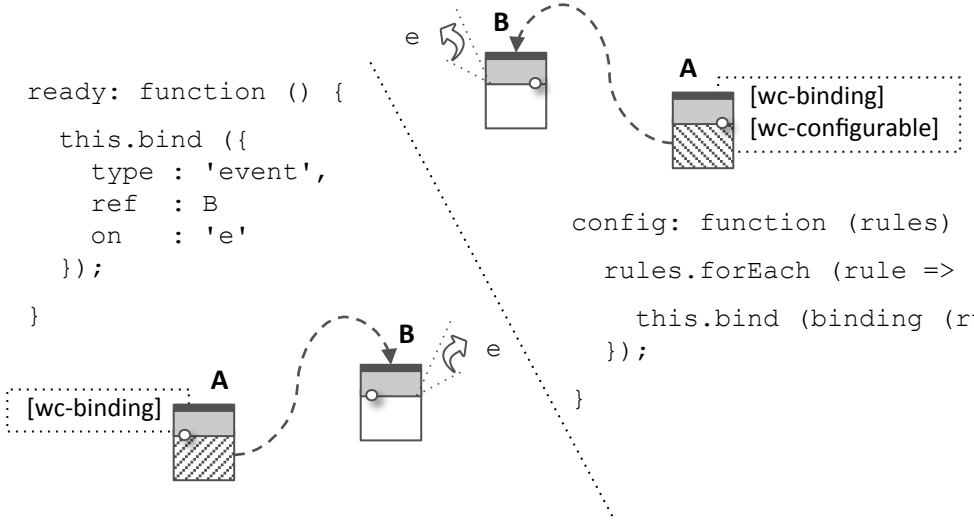
Las reglas de configuración prescribe toda la lógica de enlace compositivo que es necesario articular

IV. Patrones de Enlace

Enlace Interno

Se requiere proporcionar un mecanismo para permitir a los componentes definir sus propios enlaces compositivos con los componentes de su vecindad.

El behavior wc-composable proporciona métodos de binding para que estos puedan ser invocados desde la lógica de composición del componente.



Las capacidades de binding del behabior se usan al principio en el ciclo de vida. También suelen usarse en conjunción con el behabior configurable dentro del método config.

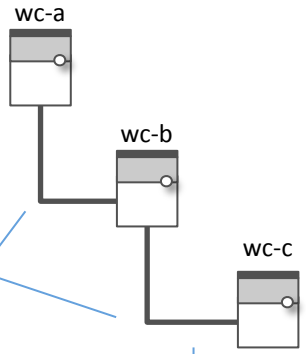
```
Behaviors ['wc-composable'] = (function () {
 var Binds = {
    event   : function eventBind () {...},
    data    : function dataBind () {...},
    message : function messageBind () {...}
};
 return {
    bind : function (binding) {
       Binds[binding.type] (binding);
    }
 };
})();
```

IV. Patrones de Enlace

Cadena de Composición

Se requiere proporcionar un mecanismo sencillo para especificar secuencias de composición entre componentes de forma completamente declarativa

- El componente wc-chain proporciona una manera sencilla de crear cadenas de composición de forma declarativa.



El componente itera sobre los elementos de la cadena y los enlaza secuencialmente de acuerdo a la lógica de enlace en el bind

La lógica de composición consiste en este caso en un enlace por evento e, otro por datos (x, y) y uno por mensaje en before (f, g)

- Se asume que la lógica de composición por pares es siempre la misma y puede factorizarse en un wc-bind externo.

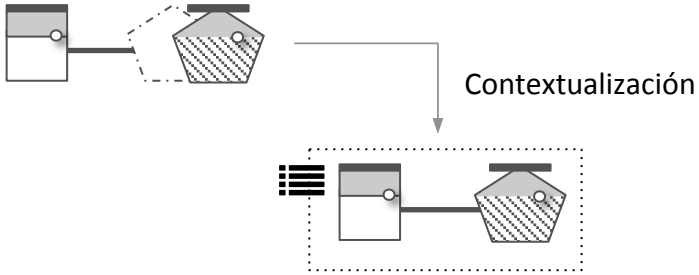
```
<wc-chain binder="#bnd">
  <wc-x/>
  <wc-y/>
  <wc-z/>
</wc-chain >
<wc-bind id="bnd">
  <wc-rule event   on="e"/>
  <wc-rule data    from="x" to="y"/>
  <wc-rule message on="f" to="g" before/>
</wc-bind>
```

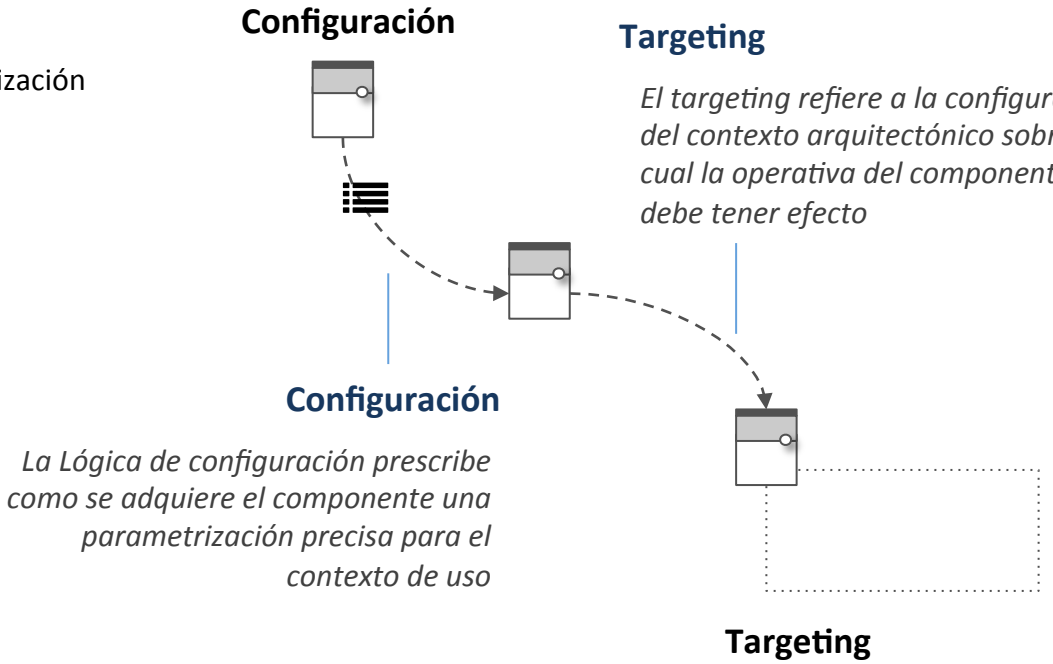
Lógica de composición común para ser aplicada iterativamente por cada par en secuencia

Los parámetros (for) y (with) de wc-bind son inyectados por wc-chain

V. Patrones de Contextualización

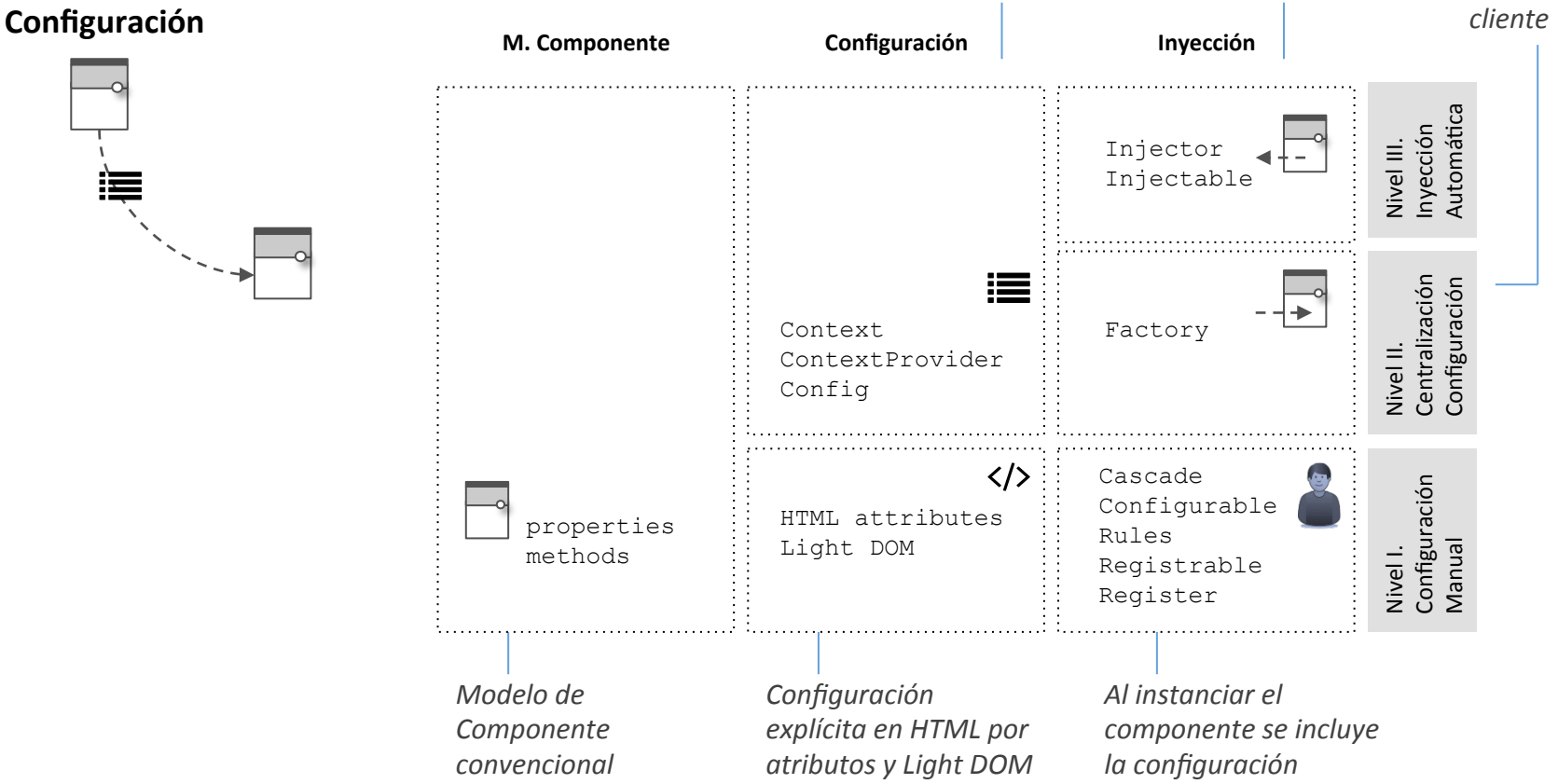
Los patrones de contextualización permiten interpretar cada colaboración entre componentes dentro del contexto de uso adecuado.





V. Patrones de Contextualización. Configuración

En relación a las estrategia de configuración se distinguen tres niveles de madurez. A continuación exponemos los criterios que definen cada nivel.



V. Patrones de Contextualización. Configuración

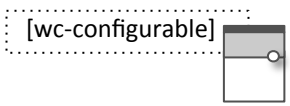
Configuración Estática Por Reglas

Nivel I

Se requiere homogenizar el proceso de configuración estática de los componentes y facilitar su procesamiento de forma semiautomática.

El componente wc-rule cubrir las necesidades de homogenización en la configuración. El behavior wc-configurable asiste en el proceso.

Cualquier componente con lógica de configuración compleja usa reglas para realizarla de forma asistida.




```
config: function (rules) {
     // process config
   }
});

     Behaviors = Behaviors || {};
     Behaviors ['wc-configurable'] = {
        ready : function () {
           var rules = getRules ();
           this.config (rules);
        }
     };
```

```
<wc-x>
  <wc-rule key="users" source="#ds1"/>
  <wc-rule key="prods" source="#ds2"/>
   ...
</wc-x>
```

Este no es más que un ejemplo de componente con configuración por reglas

Los atributos de cada regla son genéricos. No estás prefijados como propiedades. De esta manera la regla es un artefacto reutilizable y muy flexible

V. Patrones de Contextualización. Configuración

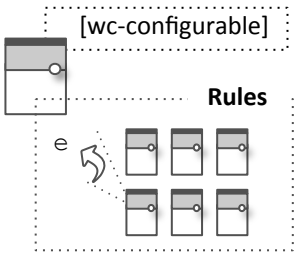
Configuración Dinámica Por Reglas

Nivel I

Se requiere realizar un proceso de configuración por reglas de manera que si se producen cambios en la configuración, estos se propaguen para actualizar el componente.

El componente wc-rule cubrir ahora las necesidades de reconfiguración dinámica al propagar por eventos los cambios.

Cualquier componente con lógica de configuración compleja usa reglas para realizarla de forma asistida.




```
config: function (rules) {
      // process config
   }
});
      Behaviors = Behaviors || {};
      Behaviors ['wc-reconfigurable'] = {
         listeners: { change: onChange },
         onChange: function (e) {
            var rules = getRules (e);
            this.config (rules);
         }
      };
```

```
<wc-x>
  <wc-rule key="users" source="#ds1"/>
  <wc-rule key="prods" source="[[ds2]]"/>
   ...
</wc-x>
```

Este no es más que un ejemplo de componente con configuración por reglas

Cada componente regla dispara un evento de cambio cada vez que se produce un cambio en los atributos

V. Patrones de Contextualización. Configuración

Contextualización de Reglas

Nivel I

Se requiere que el proceso de configuración por reglas pueda ser reinterpretado en el marco de otro componente recibido como parámetro de configuración.

El behavior wc-contextualizable añade además un atributo de contexto que permite reinterpretar la configuración del componente.

```
Behaviors ['wc-contextualizable'] = {
   properties : {
     context : Object
   },
   ready : function () {
      var rules = getRules ();
      this.config (
        this.context (rules)
      );
   },
   context: function (rule) {
     // Contextualize Rule
   }
};
```

El behavior incorpora al componente un nuevo atributo que indica el nuevo contexto donde debe interpretarse la configuración.

```
<wc-x context="[[ctx]]">
  <wc-rule key="users" source="#ds1"/>
  <wc-rule key="prods" source="[ds2]"/>
   ...
</wc-x>
```

La lógica de configuración se interpreta ahora en el marco de otro contexto

Los valores entre corchetes [] son interpretados como propiedades dentro del nuevo contexto

V. Patrones de Contextualización. Configuración

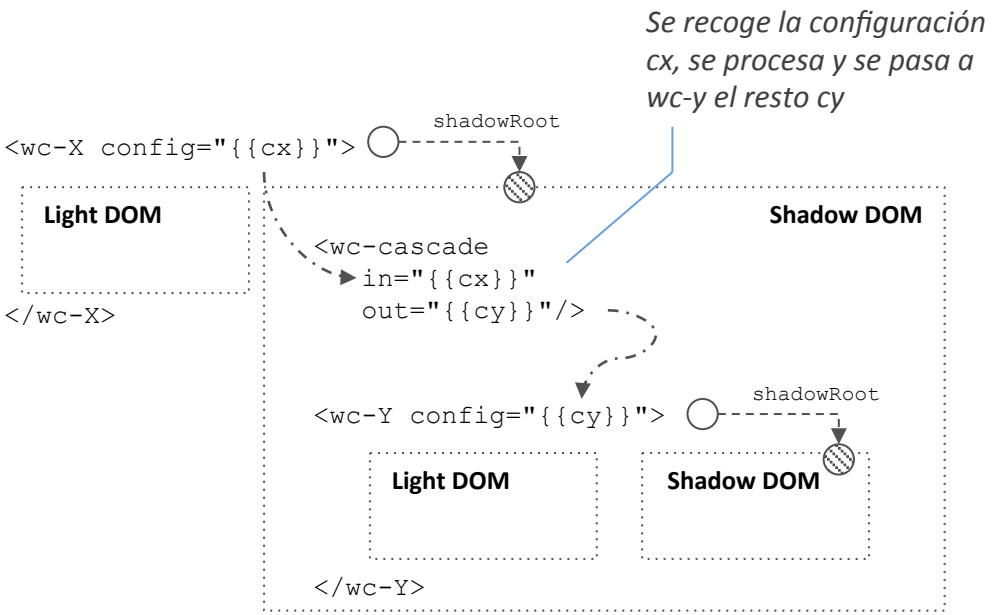
Configuración en Cascada

Nivel I

La lógica de configuración de los componentes internos a un esquema de fuerte anidamiento DOM reside en el padre. Se requiere un mecanismo de configuración sencillo.

El componente wc-cascade permite arrastrar la información de configuración en cascada aplicándola a cada paso.

Se recoge una configuración de entrada (in), se aplica en el componente en curso y genera una configuración de salida (out) para el siguiente componente en cascada.




```
<wc-cascade in="{{cx}}" out="{{cy}}">
  <wc-rule key="x" to="a" out/>
  <wc-rule key="y" to="b" out/>
  <wc-rule key="z" to="c"/>
</wc-cascade>
```

(key) es el nombre de una propiedad en cx. (to) indica en que propiedad del componente se inyecta. (out) determina si el valor no debe copiarse en cy

V. Patrones de Contextualización. Configuración

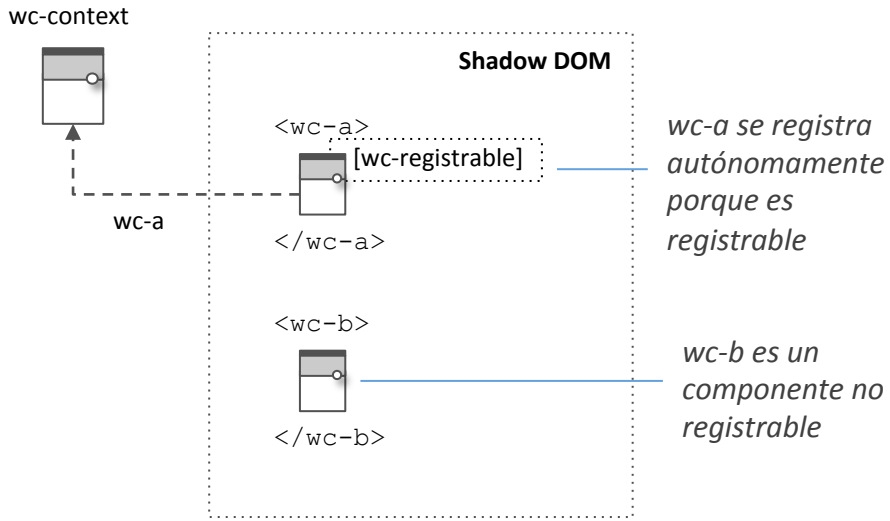
Registro Interno

Nivel I

Se requiere proporcionar un mecanismo para que los componentes lleven a cabo su proceso de registro en el contexto de configuración de forma autónoma.

El behavior wc-registrable proporciona facilidades de registro automático que quedan vinculadas al ciclo de vida del componente.

El behavior se encarga de llevar a cabo el proceso de registro en el contexto de configuración de forma automática.




```
Behaviors = Behaviors || {};
Behaviors ['wc-registrable'] = {
    ready : function () {
        var props = getProperties (this);
        this.register (props);
    },
    register : function () {
      // register props
   }
};
```

V. Patrones de Contextualización. Configuración

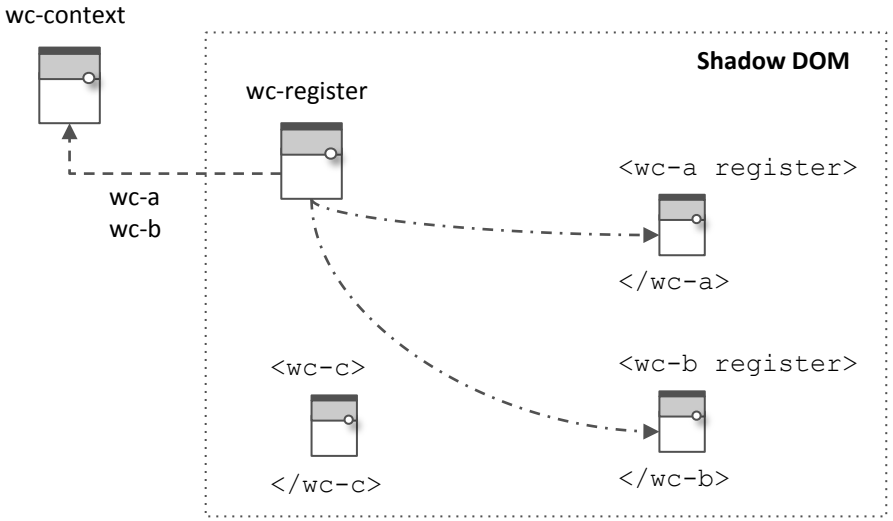
Registro Externo

Nivel I

Se requiere proporcionar un mecanismo para que un agente externo lleve a cabo las tareas de registro en el contexto de configuración de forma autónoma.

El componente wc-register permite llevar a cabo las tareas de registro automático con asistencia de técnicas exploratorias.

Internamente, el registro utiliza un wc-scout por lo que su lógica de configuración, debe incluir un handle.




```
<wc-register
   target="#container"
   ref="[register]">
</wc-register>
```

Sólo los componentes etiquetados con el rol 'register' son registrados

Por conveniencia el nombre del atributo handle se llama (target) en este componente

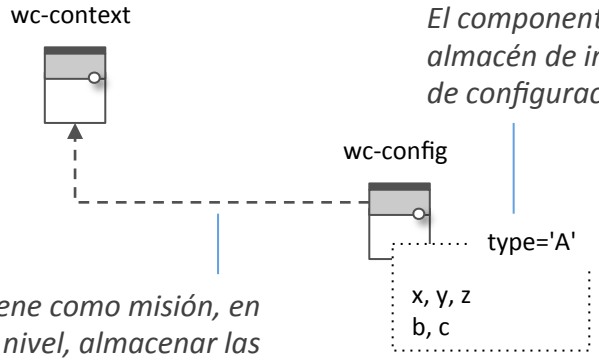
V. Patrones de Contextualización. Configuración

Encapsulación de la Configuración

Nivel II

Se requiere un mecanismo para encapsular la lógica de configuración correspondiente a un componente de manera que pueda ser almacenada y aplicada posteriormente.

El componente wc-config permite encapsular lógica de configuración y persistirla en un contexto de configuración determinado.



El componente es un almacén de información de configuración

El contexto tiene como misión, en este nivel, almacenar las configuraciones de componentes para ser utilizadas en procesos de inyección posterior

El componente recoge la configuración explícita que necesita un componente para ser generado.

Nombre y tipo del componente a construir

```
<wc-config name="a" type="A">
  <wc-rule key="x" value="[[r]]"/>
  <wc-rule key="y" value="[[s]]"/>
  <wc-rule key="z" value="3"/>
  <wc-rule key="b" type="B" prototype/>
  <wc-rule key="c" type="C" single/>
</wc-config>
```

Configuración de propiedades en la construcción. Pueden ser valores concretos o tipos de artefactos a construir

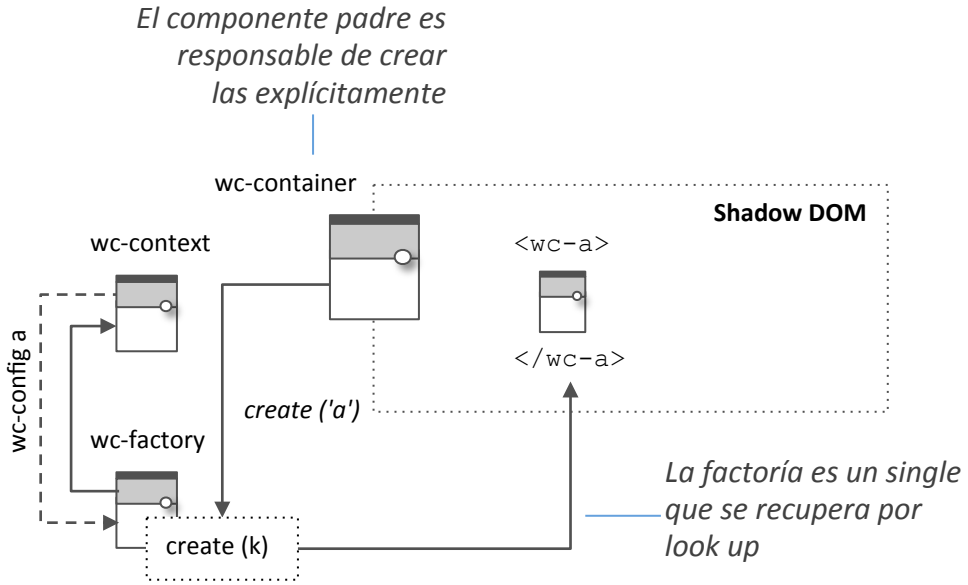
V. Patrones de Contextualización. Configuración

Inyección Bajo Demanda

Nivel II

Se requiere un mecanismo para construir bajo demanda las instancias de componentes que van a operar dentro del contenido en la sobra de un componente padre.

El componente wc-factory da respuesta a esta necesidad por medio del uso de las configuraciones en el contexto.



El componente factoría se configura indicando sobre que contexto DOM requiere operar. Consulte targeting más adelante.

```
<wc-factory target="#here">
</wc-factory>
```

Dentro de este espacio se instancian los componentes que es necesario crear dentro del contenido en la sombra

V. Patrones de Contextualización. Configuración

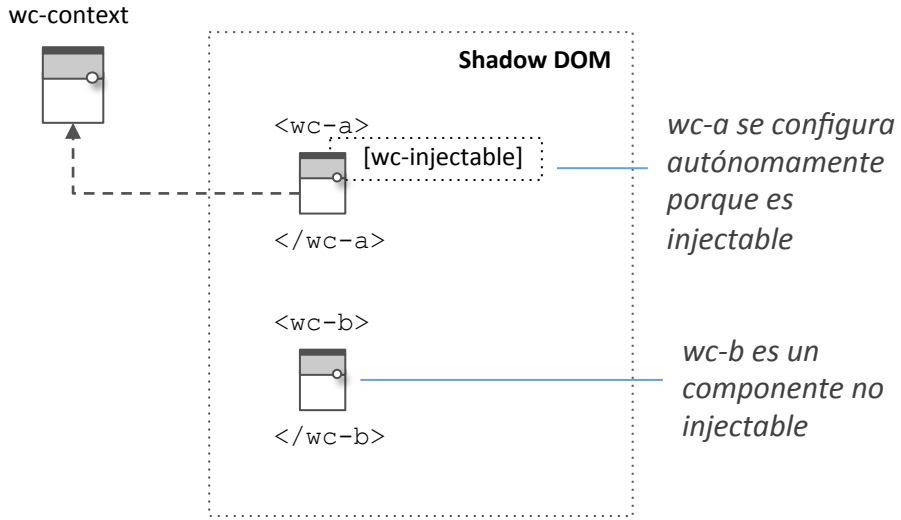
Inyección Interna

Nivel III

Se requiere proporcionar un mecanismo para que los componentes lleven a cabo su proceso de inyección de dependencias de forma autónoma.

El behavior wc-injectable proporciona facilidades de configuración automática que quedan vinculadas al ciclo de vida.

El behavior se encarga de llevar a cabo el proceso de configuración del componente de manera automática.




```
Behaviors = Behaviors || {};
Behaviors ['wc-injectable'] = {
    ready : function () {
        this.config (ctx);
    },
    config: function (ctx) {
      var cfg = ctx.get (this.is);
      // inject dependencies
      // using wc-context
   }
};
```

II. Patrones de Localización. Técnicas de Look Up

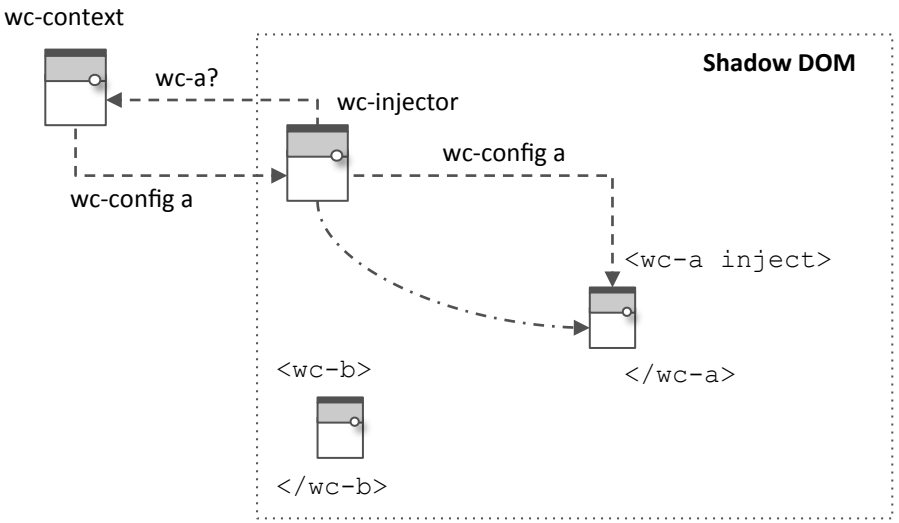
Inyección Externa

Nivel III

Se requiere proporcionar un mecanismo para que un agente externo lleve a cabo las tareas de inyección de dependencias de forma autónoma.

El componente wc-injector llevar a cabo las tareas de configuración automático con asistencia de técnicas exploratorias.

Internamente, el registro utiliza un wc-scout por lo que su lógica de configuración, debe incluir un handle.



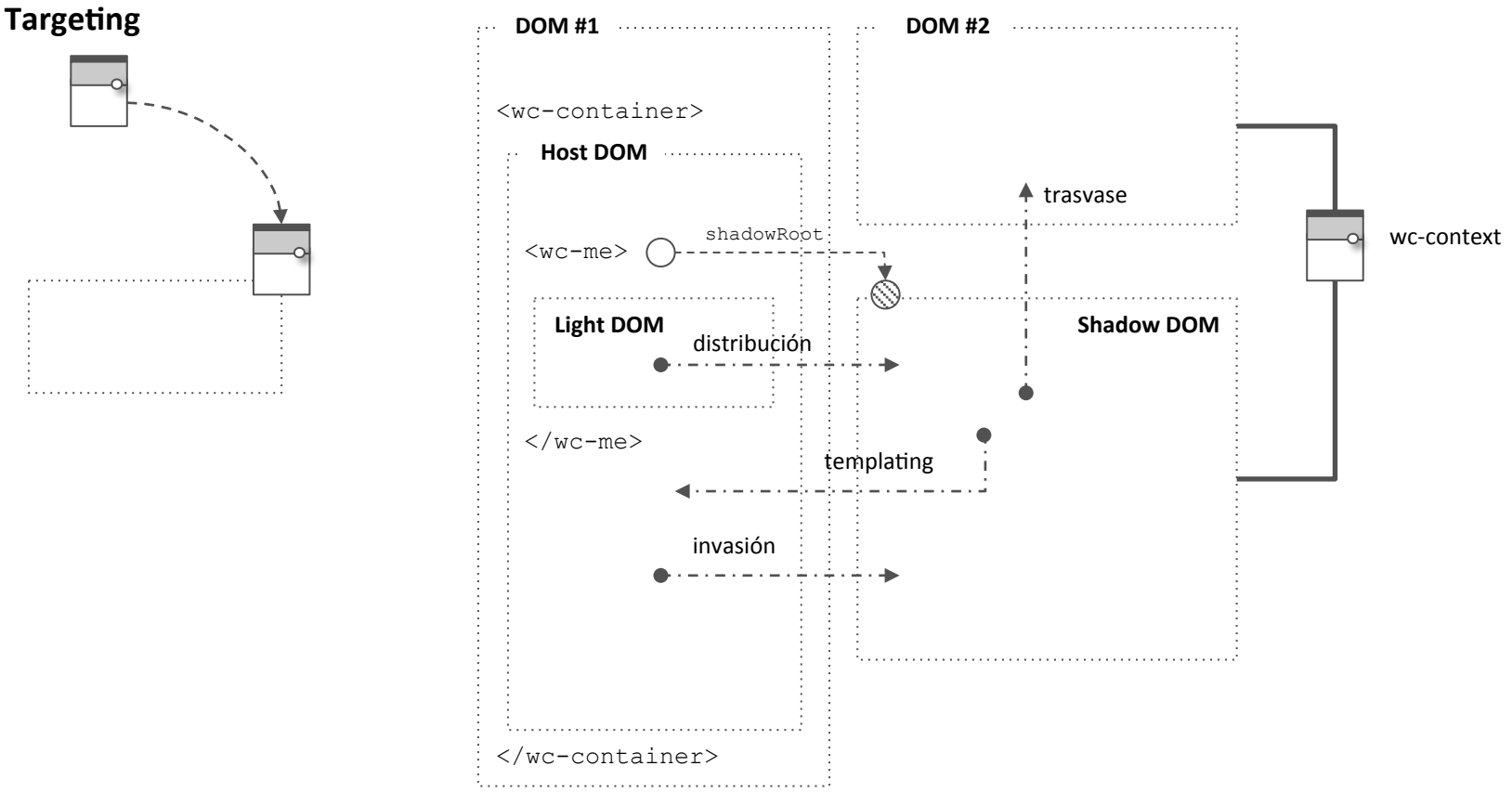

```
<wc-register
   target="#container"
   ref="[register]">
</wc-register>
```

Sólo los componentes etiquetados con el rol 'register' son registrados

Por conveniencia el nombre del atributo handle se llama (target) en este componente

V. Patrones de Contextualización. Targeting

Los patrones de targeting permiten especificar el contexto arquitectónico sobre el cual la operativa del componente debe tener efecto.



V. Patrones de Contextualización. Targeting

Targatable

Se requiere proporcionar un mecanismo para que el contenido HTML generado por un componente pueda ser ubicado en un target diferente.

El behavior targatable incorpora un atributo target que se utiliza para capturar el nuevo destino de los contenidos generados.

```
Behaviors ['wc-targatable'] = {
   properties : {
     target: Object
   },
   ready : function () {
     this.render (target);
  },
};
```

La configuración de este tipo de componentes requiere especificar el destino de los contenidos en el (target).

```
Polymer ({
   is : 'wc-x',
   behaviors : [
      Behaviors ['wc-targatable']
   ],
   render: function (target) {
     // Render
   }
});
```

```
<wc-x
   target="#container">
</wc-x>
```

Este es un componente genérico que hace uso del comportamiento targatable

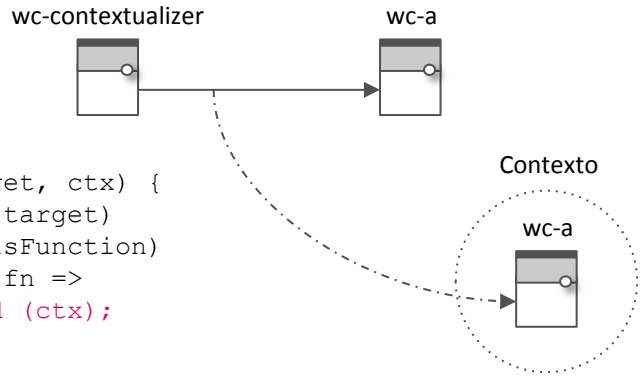
El behavior incorpora al componente el atributo (target)

V. Patrones de Contextualización. Targeting

Contextualizador de Contrato

Se requiere adaptar el contrato de un componente de manera que su uso opere no sobre el propio componente sino sobre otro componente destinatario.

El componente wc-contextulizer se encarga de realizar esta operativa de recontextualización para que opere sobre otro componente




```
function (target, ctx) {
  Object.keys(target)
    .filter (isFunction)
    .forEach (fn =>
       fn.bind (ctx);
    );
}
```

El componente se configura indicando el componente original (target) y el que opera como nuevo destino (context).

```
<wc-contextualizer
  target="#A"
  context="#B"
  keys="x y z">
</wc-contextualizer>
```

(keys) indica los métodos que deben contextualizarse. Cuando se omite este atributo se entiende que deben recontextualizarse todos los métodos

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

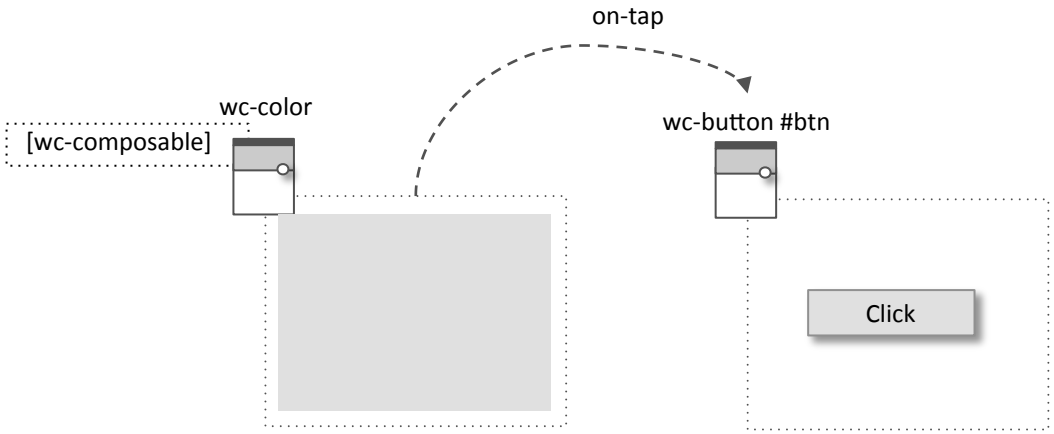
3

La Composición en la Práctica

- Composición Directa por Eventos
- Composición Indirecta por Datos
- Composición de Flujos de Eventos & Datos

I. Composición Directa por Eventos

A. Composición Interna



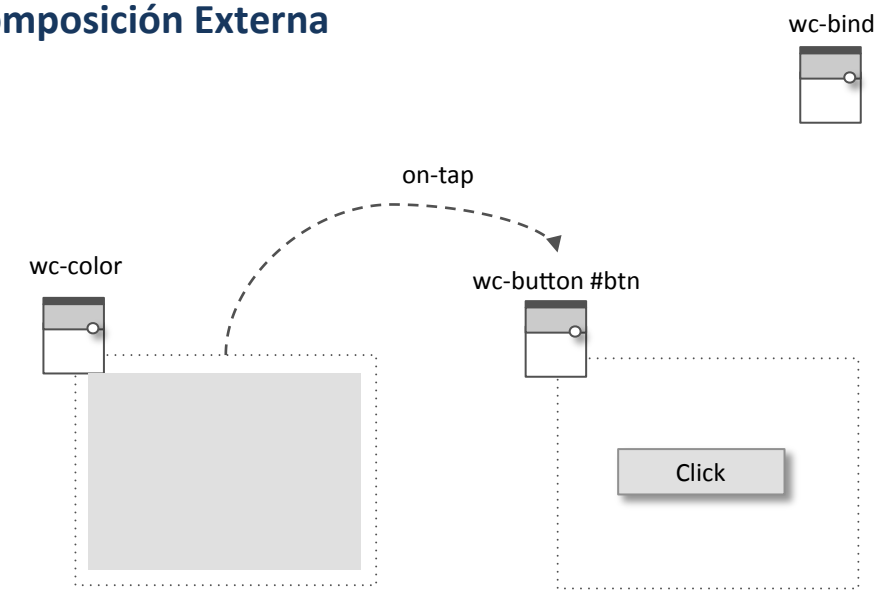

```
execute: function () {
   this.style.background = 'CCC';
}
```

- 2 componentes independientes
- Uno de ellos es composable
- Configurando sus atributos target & on se consigue la operativa

```
<wc-button id="btn">
</wc-button>

<wc-color
   target="#btn"
   on="tap">
</wc-color>
```

I. Composición Directa por Eventos

B. Composición Externa



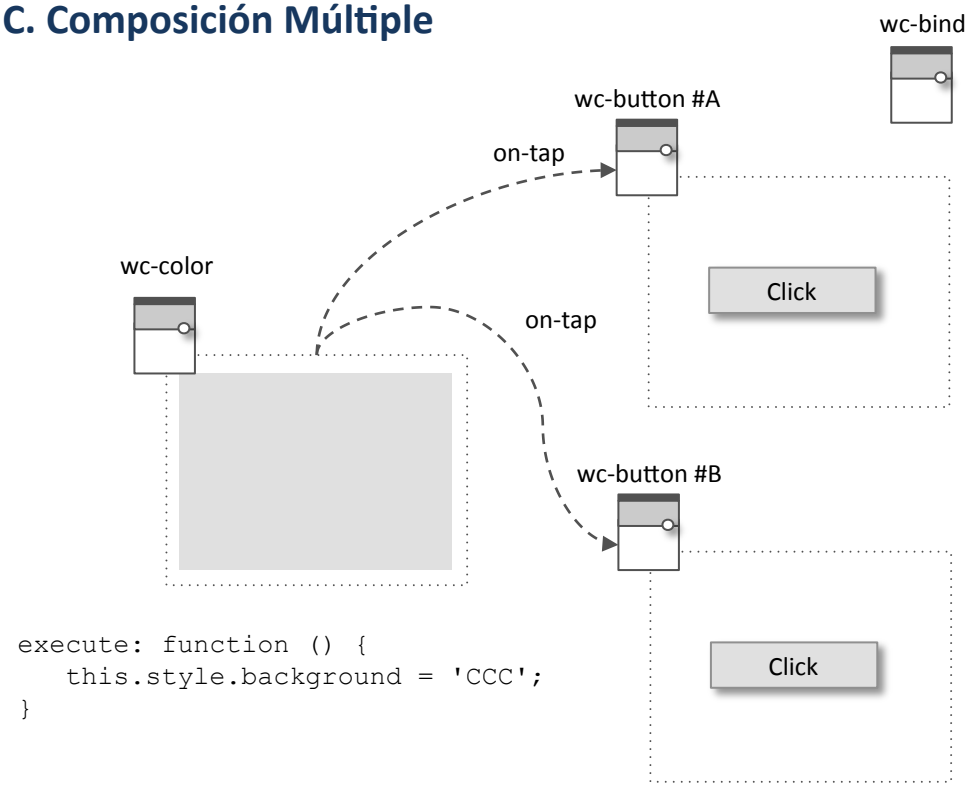

```
execute: function () {
   this.style.background = 'CCC';
}
```

- 2 componentes independientes
- Uno de ellos es composable
- Configurando el binder se consigue la operativa

```
<wc-button id="btn">
</wc-button>

<wc-color>
</wc-color>

<wc-bind for="wc-color">
  <wc-rule event with="#btn" on="tap"/>
</wc-bind>
```

I. Composición Directa por Eventos

C. Composición Múltiple

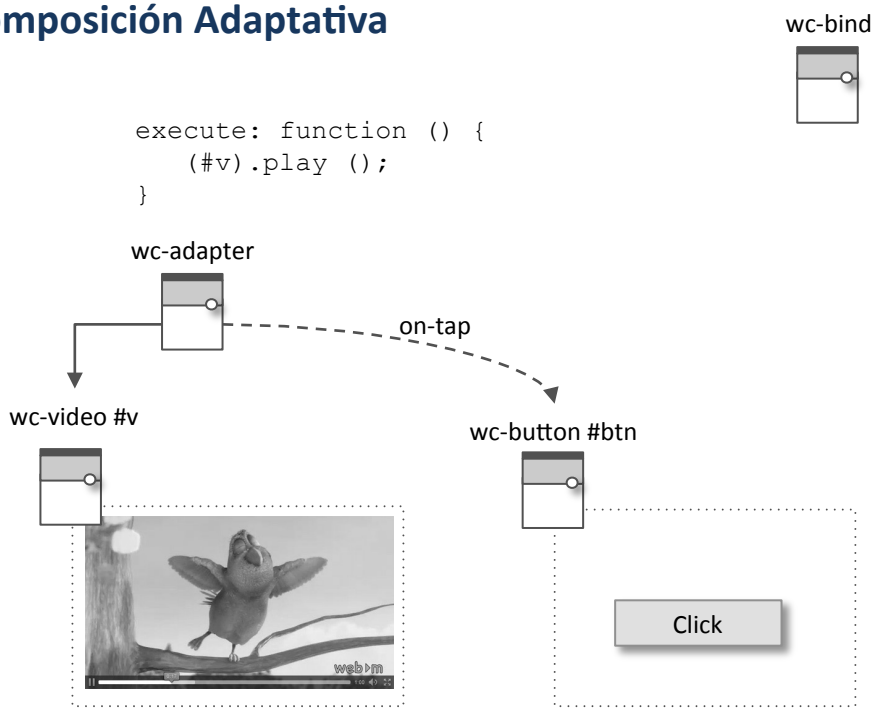


- Varios componentes independientes
- Uno de ellos es composable
- Configurando el binder se consigue la operativa

```
<wc-button id="A">
</wc-button>

<wc-button id="B">
</wc-button>

<wc-color>
</wc-color>

<wc-bind for="wc-color">
  <wc-rule event with="#A" on="tap"/>
  <wc-rule event with="#B" on="tap"/>
</wc-bind>
```

I. Composición Directa por Eventos

D. Composición Adaptativa

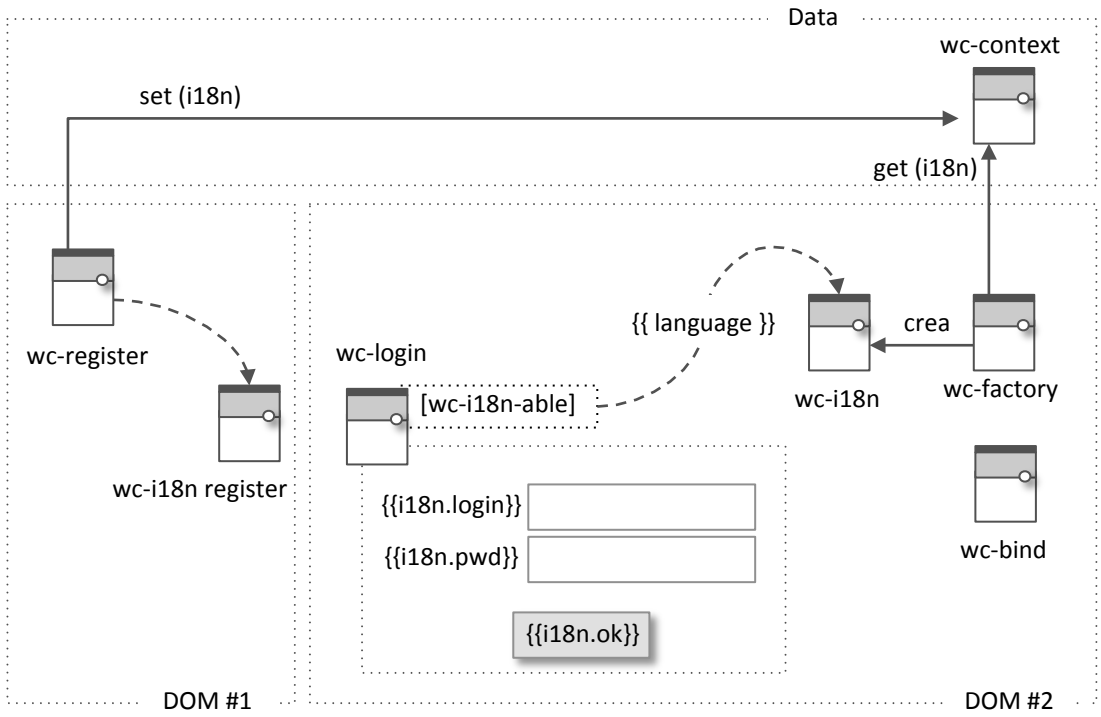


- 2 componentes independientes
- Ninguno de ellos es composable
- Configurando el binder se consigue la operativa

```
<wc-button id="btn">
</wc-button>

<wc-video id="#v">
</wc-video>

<wc-adapter target="#v">
  <wc-rule key="play" as="execute"/>
</wc-adapter>

<wc-bind for="wc-color">
  <wc-rule event with="#btn" on="tap"/>
</wc-bind>
```

II. Composición Indirecta por Datos

A. Inyección Bajo Demanda



[wc-i18nable] :

```
language: 'es'
onLanguage: function (e) {
   this.i18n (getResource(
     this.language
   ));
}
```

- Componentes con un perfil i18n-able
- Composición indirecta a través de memoria
- Binding de datos

DOM #1

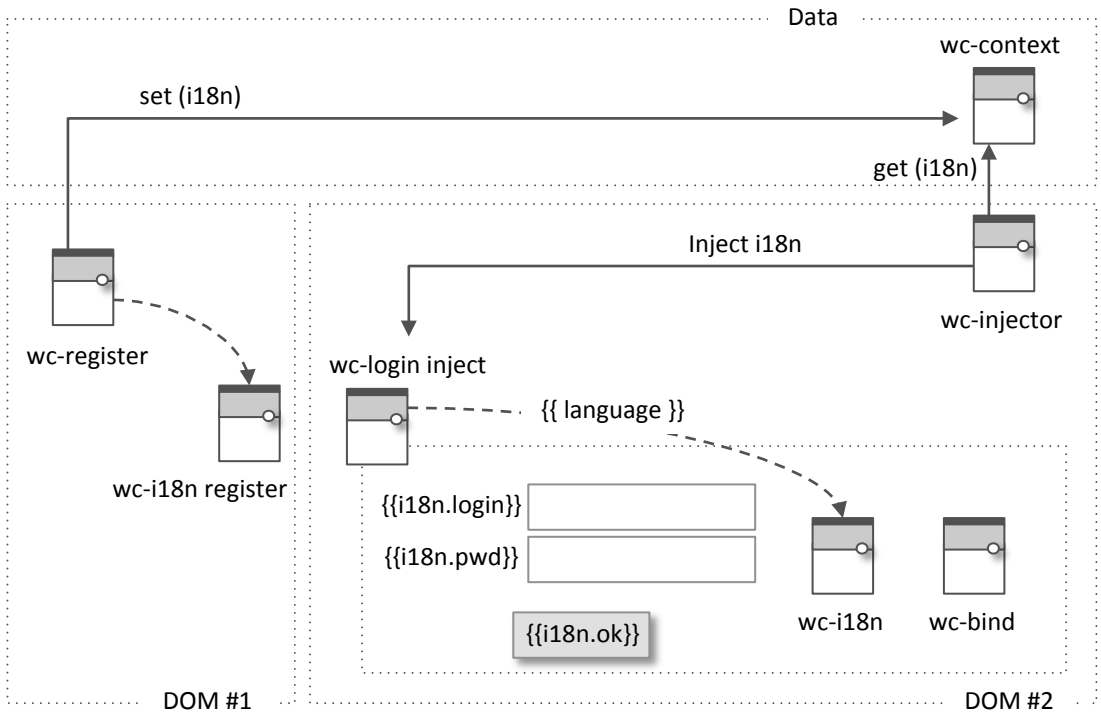
```
<wc-i18n register>
</wc-i18n>

<wc-register>
</wc-register>
```

DOM #2

```
<wc-login>
</wc-login>

<wc-factory>
</wc-factory>

    <wc-i18n>
    </wc-18n>

<wc-bind for="wc-color">
  <wc-rule data
      with="i18n"
      from="language"
      to="language"/>
</wc-bind>
```

II. Composición Indirecta por Datos

B. Inyección Automática



[wc-login] :

```
language: i18n
ready: function () {
   (#bnd).bind (...)
}
onLanguage: function (e) {
   ...
}
```

- Componentes con un perfil i18n-able
- Composición indirecta a través de memoria
- Binding de datos

DOM #1

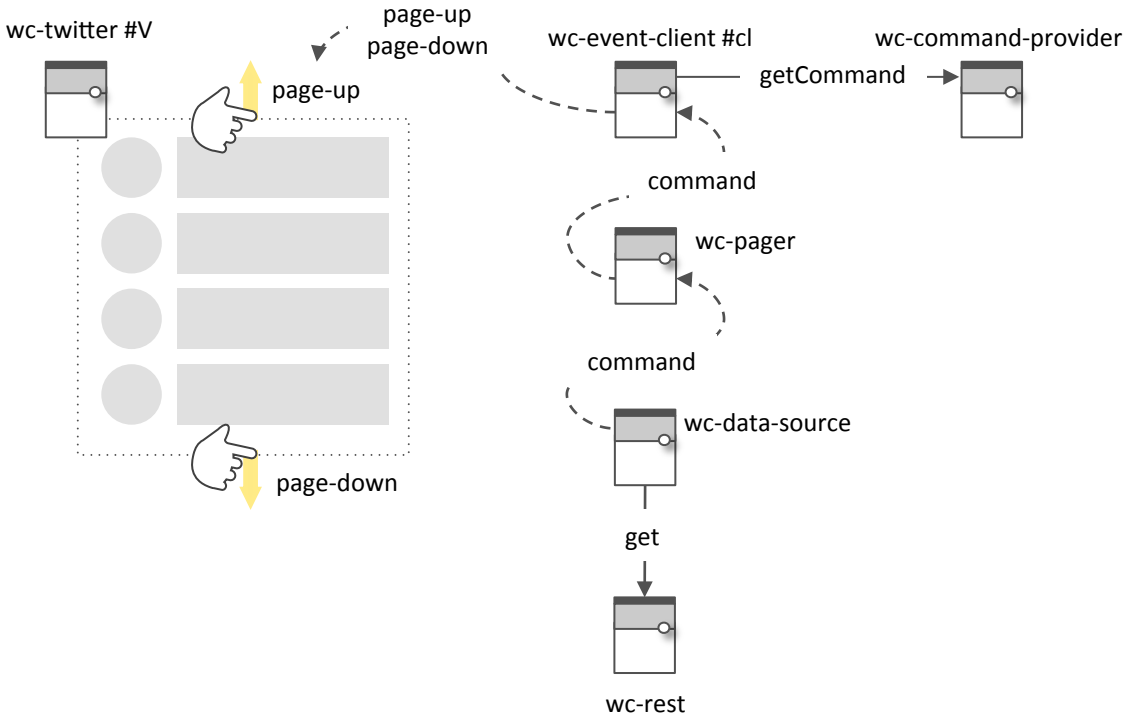
```
<wc-i18n register>
</wc-i18n>

<wc-register>
</wc-register>
```

DOM #2

```
<wc-login inject>
</wc-login>

<wc-injector>
</wc-injector>
```

III. Composición de Flujos de Eventos & Datos

A. Flujo Entrante

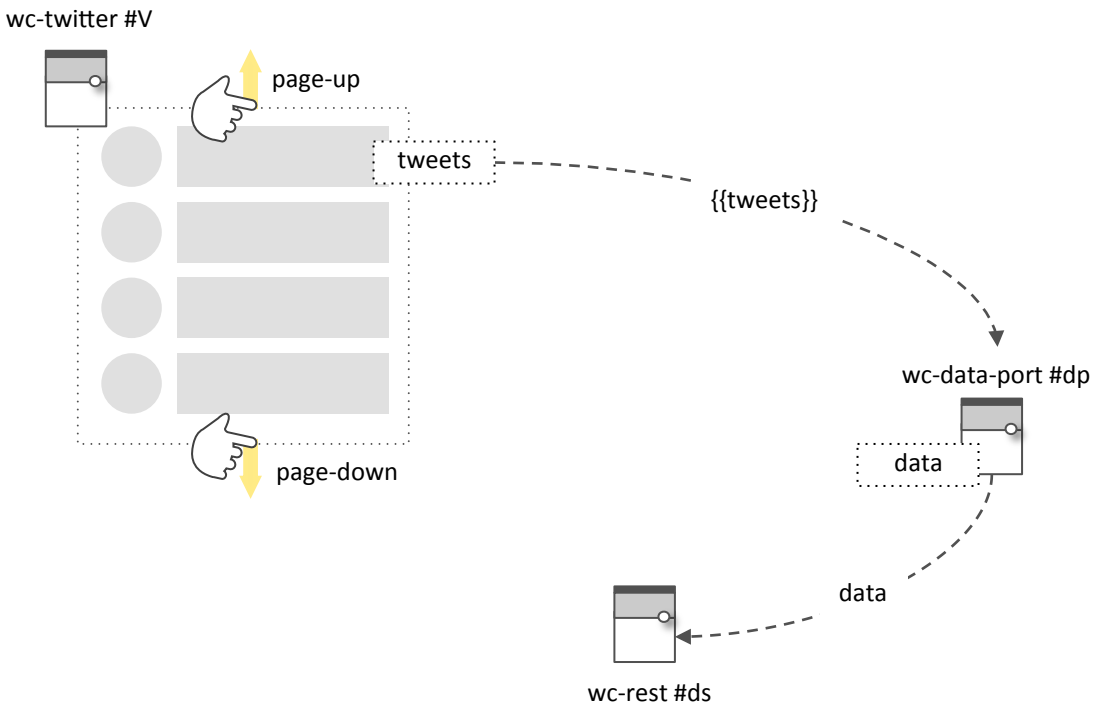


- Una colección de componentes con distinta responsabilidad
- Se encadenan compositivamente por eventos para operar
- Se conectan al componente twitter por eventos

```
<wc-twitter id="V">
</wc-twitter>

<wc-command-provider>
</wc-command-provider>

<wc-rest>
</wc-rest>

<wc-chain binder="#in">
  <wc-even-client id="cl".../>
  <wc-pager.../>
  <wc-data-source.../>
</wc-chain>

<wc-bind id="in">
  <wc-rule event on="command"/>
</wc-bind>

<wc-bind for="#V">
  <wc-rule event
      with="#cl"
      on="page-up"/>
  <wc-rule event
      with="#cl"
      on="page-down"/>
</wc-bind>
```

III. Composición de Flujos de Eventos & Datos

B. Flujo Saliente



- La respuesta de la fuente de datos es asíncrona
- Los resultados se emiten por eventos data
- El puerto de datos recoge los datos de eventos
- Se conecta el puerto de datos con la propiedad {{tweets}} de #V

```
<wc-twitter id="V">
</wc-twitter>

...

<wc-data-port id="#dp">
...
</wc-data-port>

<wc-bind for="#dp">
  <wc-rule event
       with="#ds"
       on="data"/>
</wc-bind>

<wc-bind for="#V">
  <wc-rule data
       with="#dp"
       from="data"
       to="tweets"/>
</wc-bind>
```

Arquitecturas Orientadas a Componentes Web

Patrones de Composición

{codemotion}

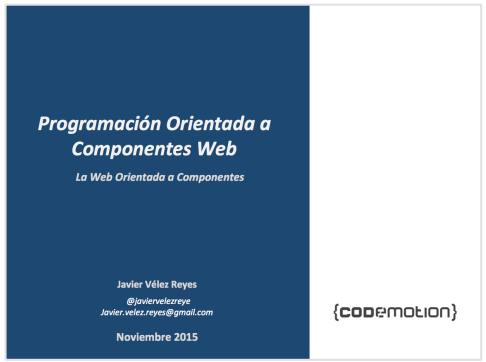
www.javiervelezreyes.com

Referencias

La Web Orientada a Componentes

Componentes Web · Arquitecturas Software · Frontend · JavaScript · Composición · Encapsulación · Reutilización · Buenas Prácticas · Principios de Diseño · Errores comunes

Fecha	Noviembre de 2015
Lugar	Code Motion
Páginas	57
Ref	https://goo.gl/mCxm3w



Principios de Diseño en Componentes Web

Programación Orientada a Componentes · Principios de Diseño · Buenas Prácticas · Recomendaciones de Diseño y Desarrollo · Técnicas de Programación de Componentes Web · Polymer

Fecha	Marzo de 2015
Lugar	Polymer Madrid
Páginas	170
Ref	https://goo.gl/t0GpVS

Referencias

Arquitecturas para la Reutilización en JavaScript

Programación Orientada a Componentes · Componentes · Reutilización · Programación · JavaScript · Modelos Arquitectónicos · Metaprogramación · Adaptación arquitectónica

Fecha	Abril de 2016
Lugar	JS Day
Páginas	43
Ref	https://goo.gl/i9pd03

Metaprogramación Compositiva en JavaScript

Metaprogramación · Programación adaptativa · Programación orientada a Componentes · JavaScript · Adaptación · Alineamiento Arquitectónico · técnicas de metaprogramación · patrones de diseño

Fecha	Mayo de 2015
Lugar	JS Day
Páginas	84
Ref	https://goo.gl/wLllU2

Arquitecturas Orientadas a Componentes Web

Patrones de Composición

Javier Vélez Reyes

@javiervelezreye
Javier.velez.reyes@gmail.com

Noviembre 2016

{codemotion}