

Aplicaciones JavaScript II

Y ahora, ¿qué?

Tenemos Modelo y Vista

- Todo el mundo está de acuerdo en estos dos puntos
- Datos + presentación
- Todavía falta algo...

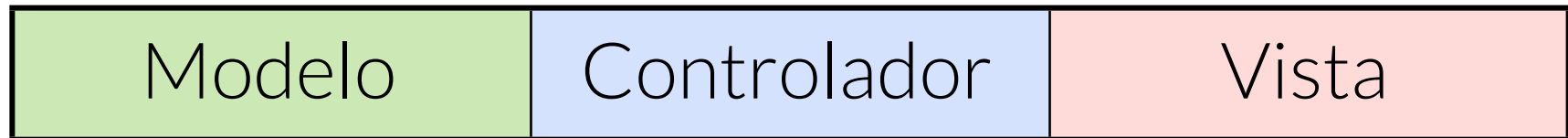
MV*

MV*

- El papel del Controlador no está tan claro
 - Gestionar la interacción?
 - Gestionar los eventos de la vista?
 - Gestionar las rutas de la página?
 - Gestionar al modelo?
 - ...

MV*

La visión tradicional:



MV*

En JavaScript...



MV*

En JavaScript...




Proxy ← ¡Todo lo demás! → Template


Controlador

El modelo “estándar” de Backbone.js

- A pesar de que no se dice explícitamente...
- No hay “Controller”
- La vista propiamente dicha es el template
- “View” gestiona la interacción
- Es decir, hace de controlador

Controlador



```
var VistaTarea = ProJS.MVC.View.extend({  
  events: {  
    "change .toggle": "toggleDone"  
  },  
  template: ProJS.Plantillas.byId('item-template'),  
  tagName: 'li',  
  parent: $('#todo-list'),  
  toggleDone: function(e, target) {  
    var done = this.model.get('completed');  
    this.model.set({'completed': !done});  
  }  
});
```


Controlador

Tareas



¿Qué tienes que hacer?



Pervertir la vistas para hagan de C

Controlador

El código que llevamos hecho:

- <https://bitbucket.org/werelax/projs-material/raw/3fa578baf0cf/tema4/projs-mv.js>
- Tiene algunas mejoras...

Controlador

Siguiente paso:

- Añadir el DSL de eventos a las vistas
- Hacer que funcione el método **toggleDone**
- Añadir un método **destroy** que elimine la tarea

Controlador

¿Cómo gestionar el flow general de la app?

- Debería ser un Mediador
- Pero lo más común es hacer un “controlador general”
- Si las vistas están asumiendo el rol de Controller...

Controlador



```
var listaDeTareas = new ListaDeTareas(),  
    appView = new AppView({el: $("#todoapp"),  
                           list: listaDeTareas});
```

```
listaDeTareas.subscribe('added', function(tarea) {  
    new VistaTarea({model: tarea}).render();  
});
```

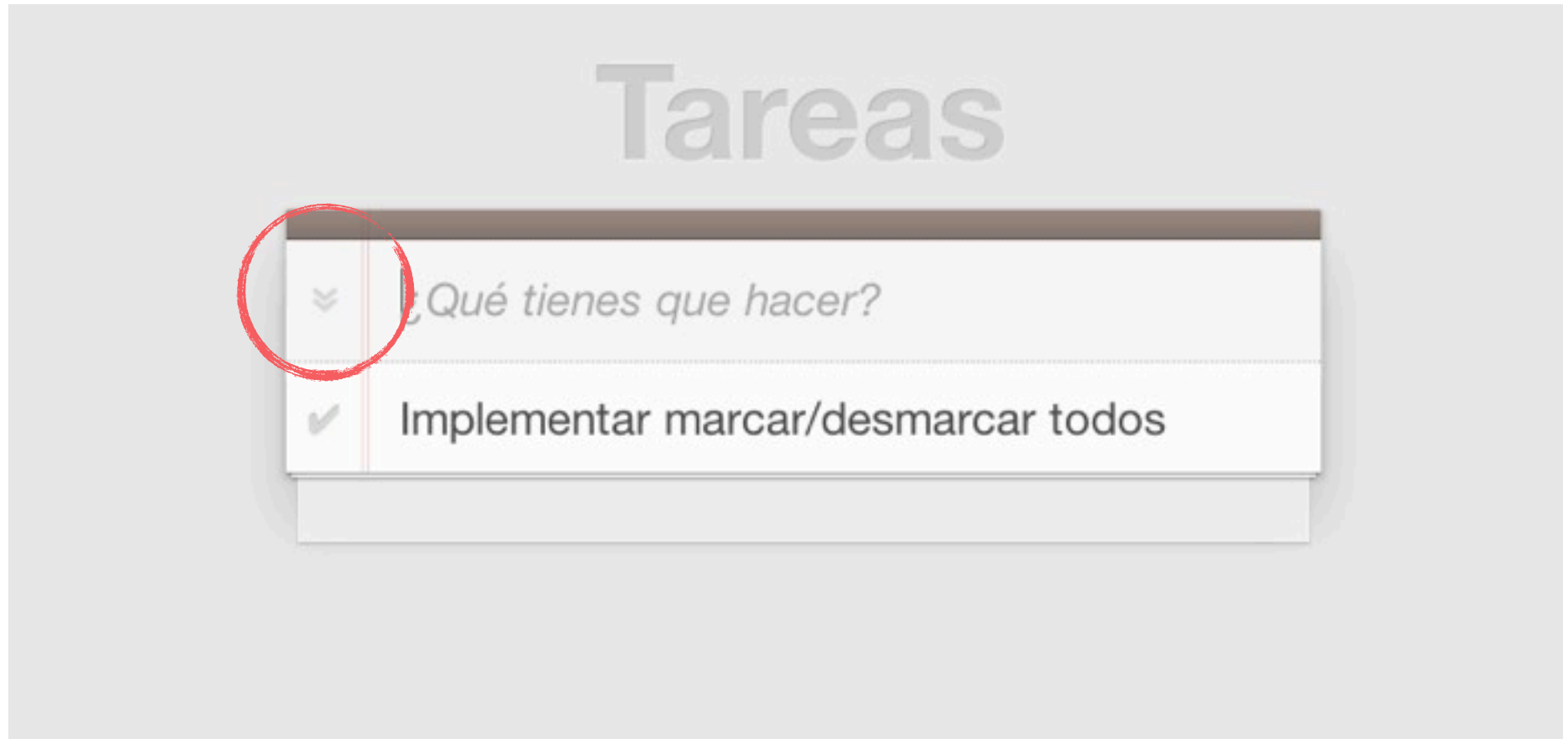
Controlador

Mi solución:

- <http://jsbin.com/akulir/6/edit>

Controlador

Implementar “marcar/desmarcar todos”



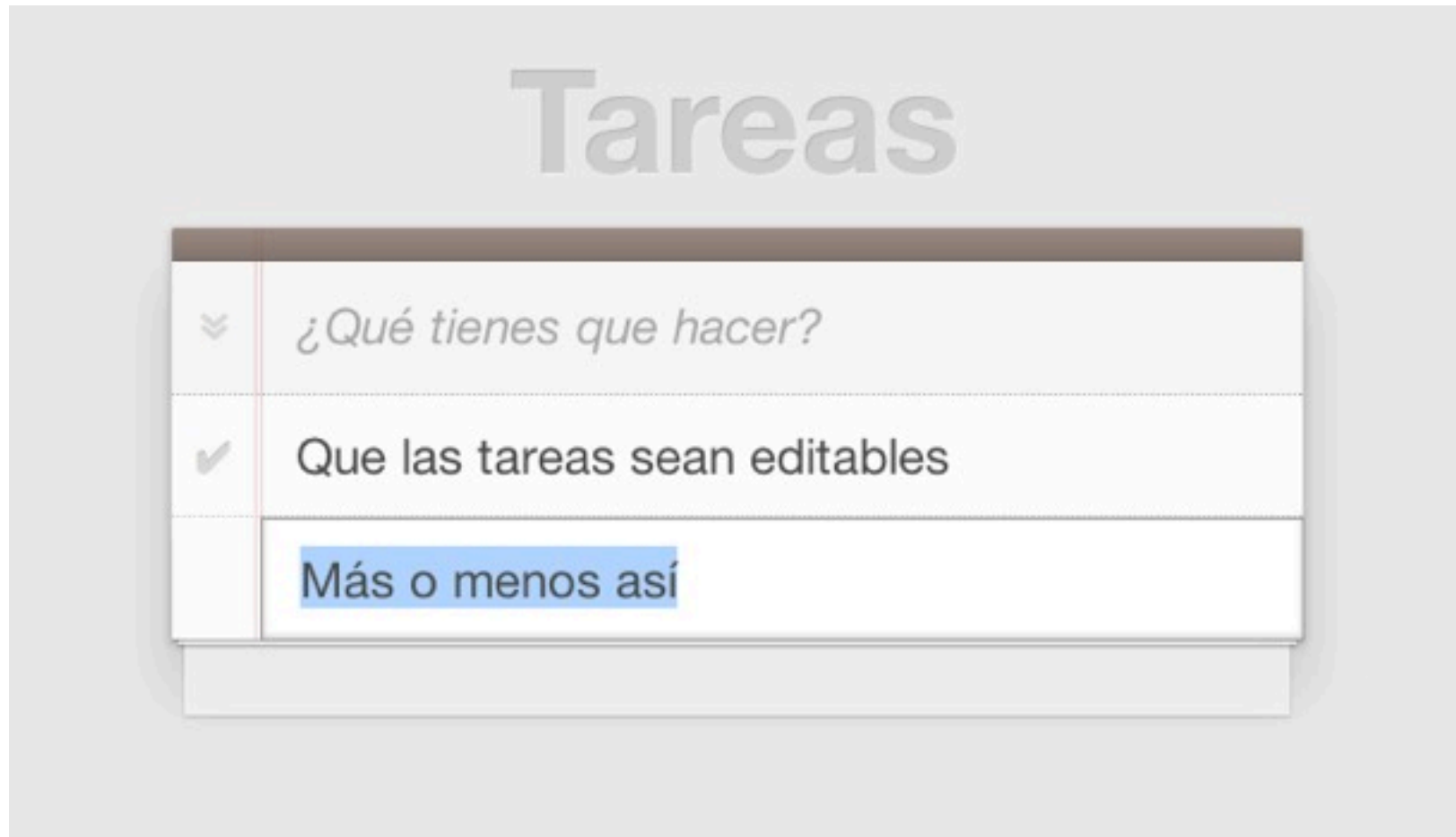
Controlador

Por ejemplo, así:

- <http://jsbin.com/akulir/7/edit>

Controlador

Hacer las tareas editables (doble click)



Controlador

Mi código:

- <http://jsbin.com/akulir/9/edit>

Controlador

Y el final: Filtros y contadores



Controlador

Una posible implementación:

- <http://jsbin.com/akulir/11/edit>

Controlador

Problemas que hemos visto:

- Gran acoplamiento!
- Muy difícil de reutilizar
- Los “controladores” hacen demasiadas cosas
- No se respeta el SRP

Controlador

```
var VistaTarea = ProJS.MVC.View.extend({  
  //...  
  toggleDone: function(target, e) {  
    this.model.toggle();  
  },  
  destroy: function() {  
    this.model.destroy();  
  },  
  edit: function() {  
  },  
  normal: function() {  
  },  
  onKeyDown: function(target, e) {  
  },  
  render: function() {  
  }  
});
```

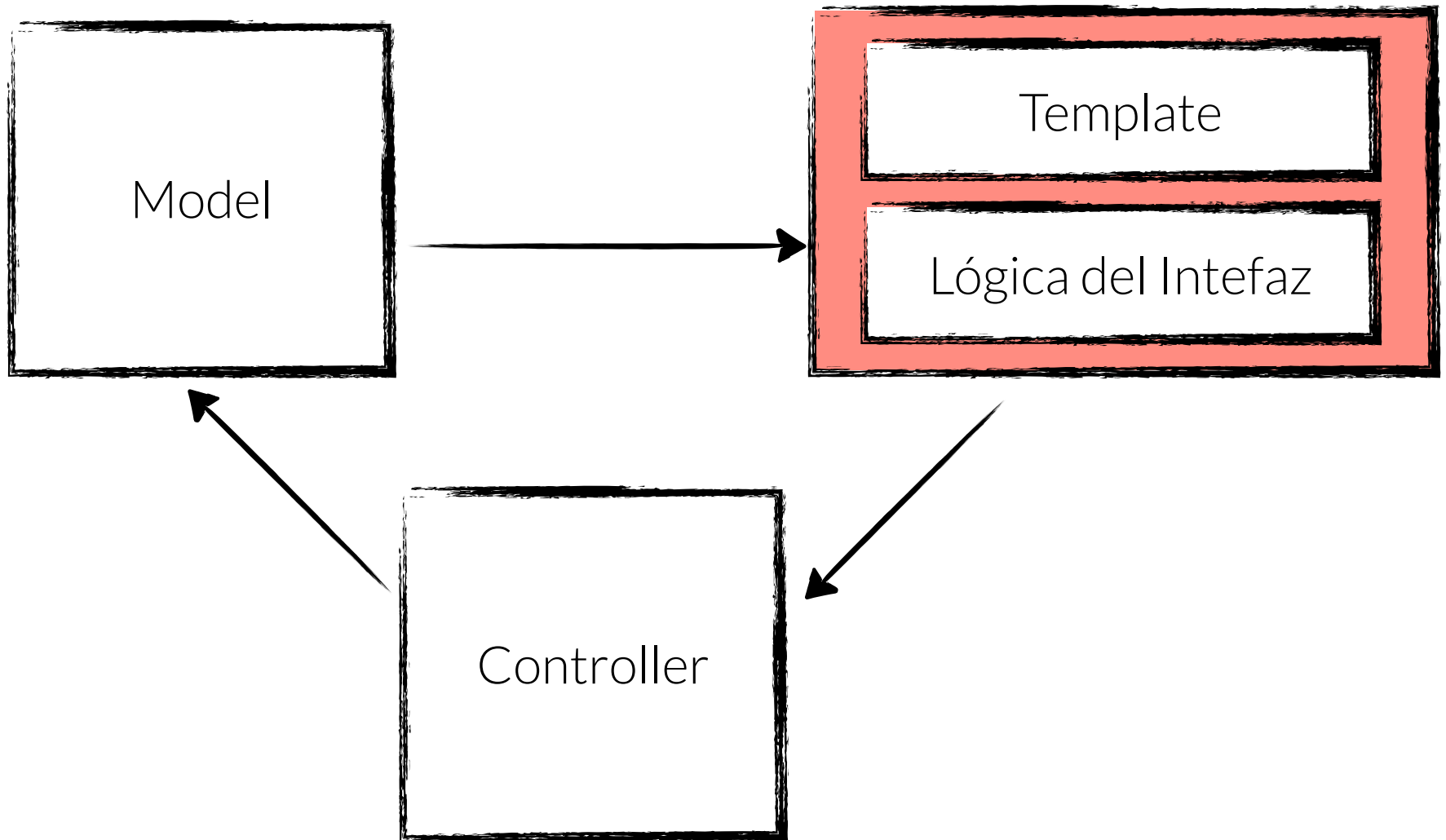


Controlador

```
var VistaTarea = ProJS.MVC.View.extend({  
  //...  
  toggleDone: function(target, e) {  
    this.model.toggle();  
  },  
  destroy: function() {  
    this.model.destroy();  
  },  
  edit: function() {  
  },  
  normal: function() {  
  },  
  onKeyDown: function(target, e) {  
  },  
  render: function() {  
  }  
});
```



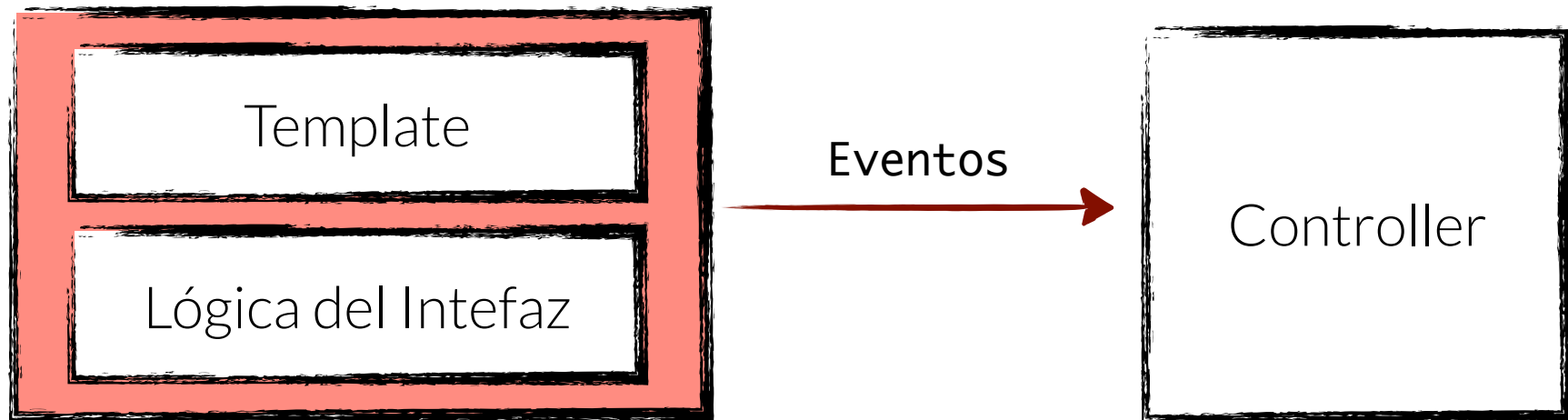
MV*



MV*

```
var ControllerTarea = Class.extend({
  init: function(options) {
    augment(this, pick(options, 'model'));
  },
  toggle: function() {
    this.model.toggle();
  },
  destroy: function() {
    this.model.destroy();
  },
  changeTitle: function(title) {
    this.model.set({title: title}, {silent: true});
  }
});
```

MV*

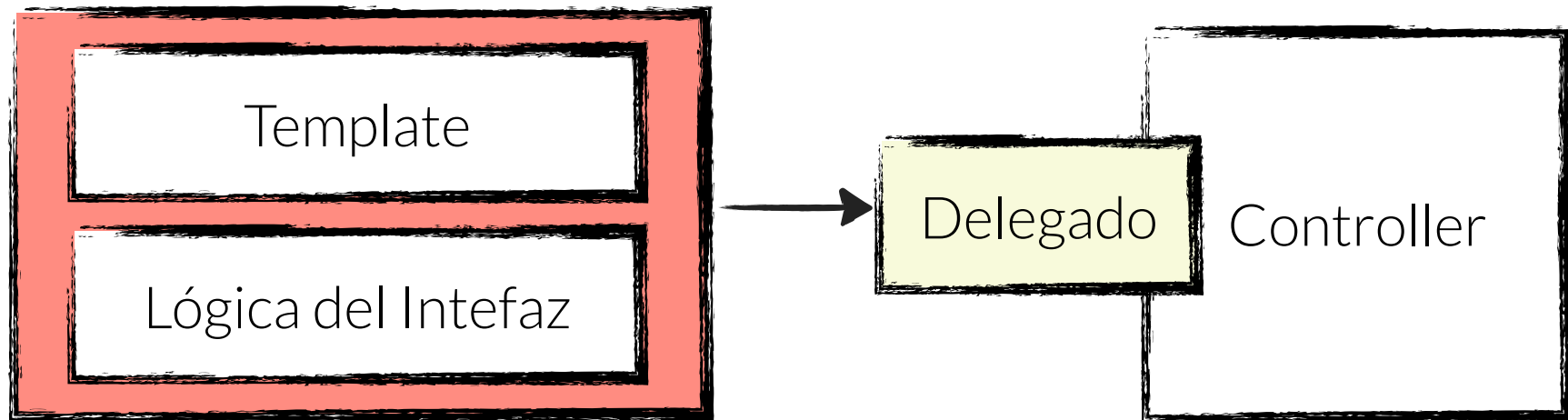


MV*

<http://jsbin.com/akulir/15/edit>

- Muy flexible: las vistas pueden emitir cualquier cosa
- El controlador es fácil de cambiar
- No está muy claro qué eventos emite la vista
- Para vistas complejas es un caos

MV*



Delegados

<http://jsbin.com/akulir/16/edit>

- Más ordenado: contrato explícito
- Apropiado para vistas complejas
- Delegado es una interfaz
- Controller es una implementación del delegado
- Variar la implementación sin modificar la Vista

MV*

La Vista (y su lógica de interfaz)...

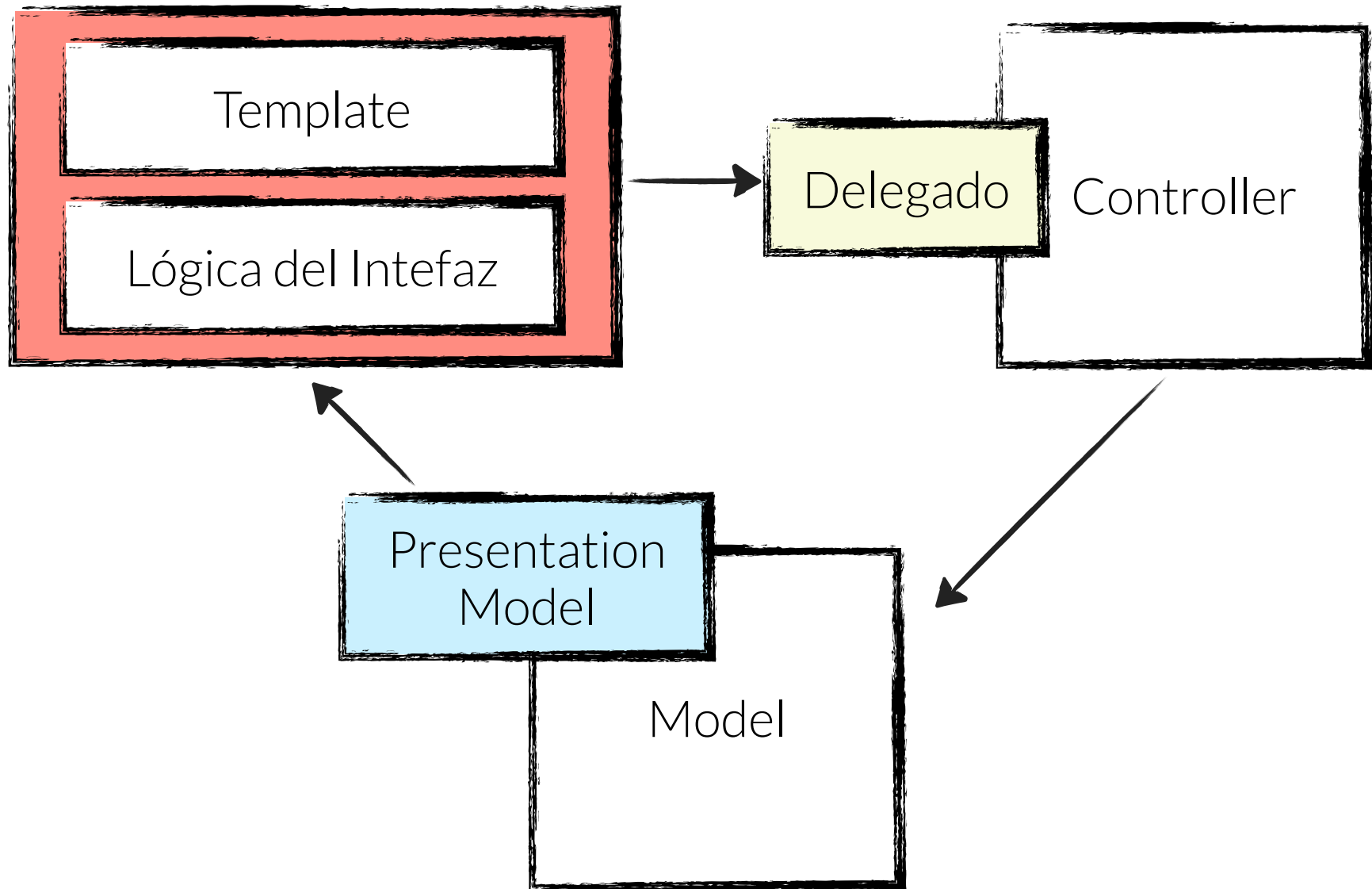
- Suele ser la parte más compleja
- Muchas veces se PODRÍA reutilizar
- Pocas veces se reutiliza
- ¡Acoplamiento!

MV*

La Vista (y su lógica de interfaz)...

- Eventos o Delegados: desacoplar Vista-Controlador
- La Vista sigue demasiado acoplada al Modelo
- Sin necesidad!

MVP



P: Presentation Model

Para desacoplar Vista-Modelo:

- Podemos usar Modelos de Presentación
- Decorador que adapta el Modelo
- A las necesidades de una vista en concreto

P: Presentation Model

Si cambiamos el nombre de las prop. del modelo..

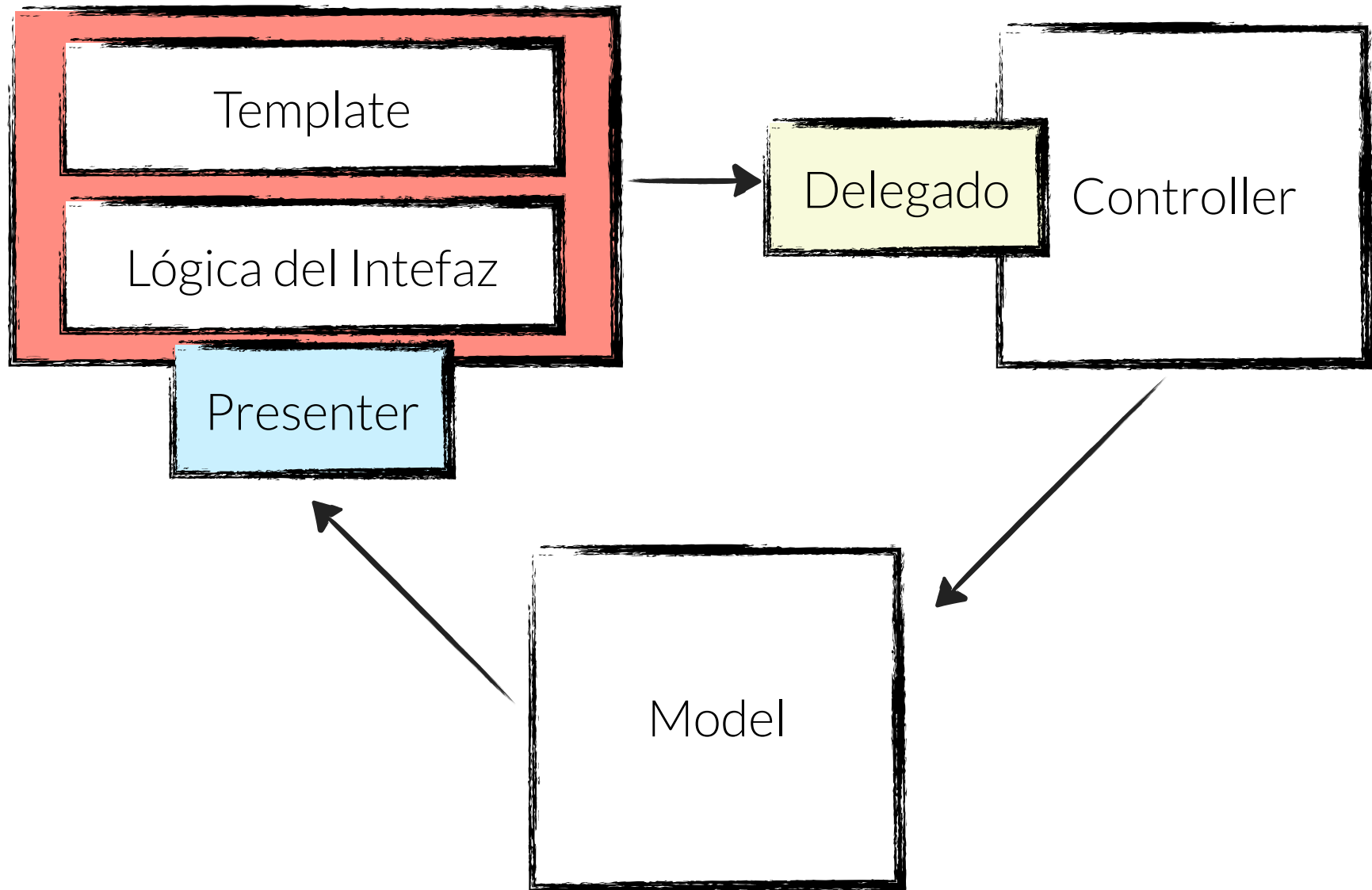
```
var Tarea = ProJS.MVC.Model.extend({  
  defaults: {  
    hecho: false,  
    titulo: "Tarea sin título",  
    visible: true  
  },  
});
```

P: Presentation Model

<http://jsbin.com/akulir/17/edit>

- Bastante sencillo de implementar
- Vista y Modelo pueden variar sin muchos problemas
- Varios Modelos se pueden adaptar a una sola Vista
- Y viceversa

MVP



P: Presenter

El Presenter se sienta entre la Vista y el Modelo

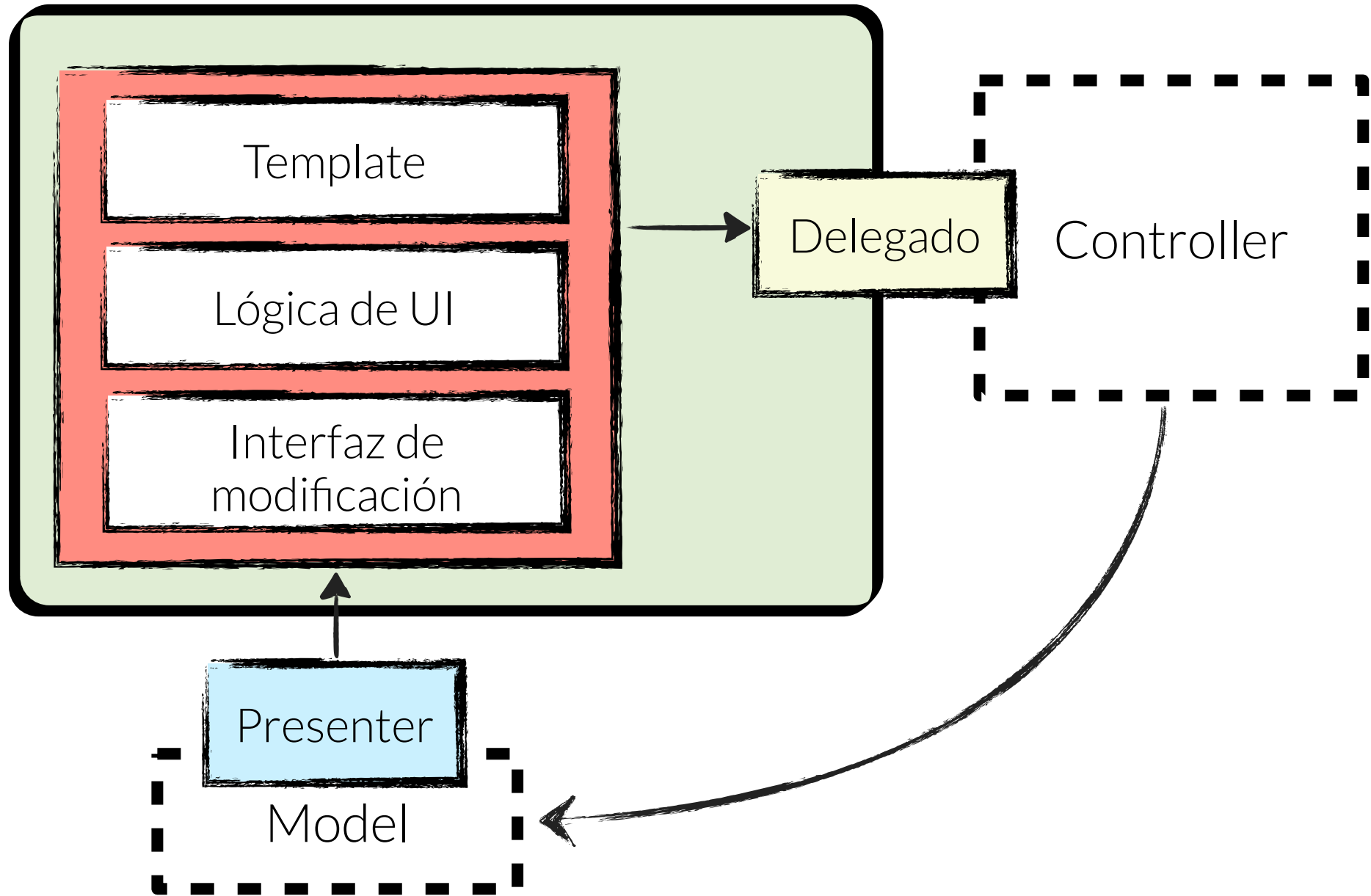
- La Vista se convierte en un interfaz de setters
- El Presenter observa el modelo
- Transforma eventos en operaciones
- Para modificar la Vista

P: Presenter

<http://jsbin.com/akulir/18/edit>

- Control mucho más fino sobre qué se modifica
- No se renderiza todo, se calcula el cambio
- Más laborioso, más código
- Mucho más abstracto!
- Más eficiente

M-(P-V-D)-C



M-(P-V-D)-C

La arquitectura para Apps Grandes

- Los “Componentes” (P-V-D) son reutilizables!
- Para hacer UIs complejos
 - Widgets sofisticados
 - Biblioteca de componentes de interfaz
- Más complejo que el MVC “clásico”
- El acoplamiento entre Modelo y Controlador
 - Es inevitable porque están muy unidos
 - Generalmente no resulta problemático