Transcluded ShadowDOM Components for AngularDart

*Status: (Draft)*

*Authors:* [*misko@hevery.com*](mailto:misko@hevery.com)

*This document is published to the web as part of the public* [*Angular Design Docs*](https://drive.google.com/#folders/0BxgtL8yFJbacUnUxc3l5aTZrbVk) *folder*

# Objective

Be ready for Shadow DOM future without being slow on todays polyfills.

# Background

Currently Angular Components run on top of ShadowDOM. This works on chrome, but relies on platform.js polyfill for non chrome browsers. The only way to emulate ShadowDOM is by intercepting and wrapping all DOM access. This means that a simple presence of platform.js slows down DOM manipulation for any application on the page. This creates a problem until a more efficient polyfill can be rewritten in.

# Prior Art

AngularJS v1.x has components without the use of ShadowDOM.

# Detailed Design

## Declaring a Component

This is the way a component is declared today in AngularDart. See [API](https://docs.angulardart.org/#angular/angular-core-annotation.Component).

|  |
| --- |
| @Component(  selector: 'my-component',  templateUrl: 'my-component.html'  cssUrls: ['my-component.css']  )  class MyComponent {  } |

## ShadowDOM Implementation

Current default implementation of angular creates a ShadowDOM for each instance of <my-component>. Into that ShadowDOM we load the template and the CSS styles. Something like this:

|  |
| --- |
| <my-component>  *<-shadow-root->*  <style>  // contents of my-component.css  </style>  <div>  Some header  <content></content>  </div>  *</-shadow-root->*  <span>My component content.</span>  </my-component> |

## Shadowless Implementation

Notice that the code written by the application developer has no references to the ShadowDOM API, therefore Angular does not have to load the component using ShadowDOM. Instead Angular can be configured to load the component template using normal DOM operations. This is what we refer to as transclusion in AngularJS v1.x. The resulting DOM is as follows:

|  |
| --- |
| <my-component>  <style>  // contents of my-component.css  </style>  <div>  Some header  <content>  <span>My component content.</span>  </content>  </div>  </my-component> |

This is already implemented in Angular using [TranscludingComponentFactory](https://github.com/angular/angular.dart/blob/master/lib/core_dom/transcluding_component_factory.dart) and controlled by the [useShadowDom API](https://docs.angulardart.org/#angular/angular.Component@id_useShadowDom).

### CSS encapsulation

Currently the use of transclusion suffers from lack of CSS encapsulation. To correct this we will have to do the same thing which the platform.js polyfill does, and that is to prefix the CSS with unique attributes to achieve the encapsulation. *(To clarify, template compilation refers to the runtime process by which Angular identifies directives in the template. It is not offline code pre-generation.)*

1. During the template compilation, Angular will add a unique attribute to each element in the template.
2. During <style> insertion, Angular can parse the CSS and look for selectors, which can than be amended with the unique attribute name during compilation.

The advantage we have over platform.js polyfill is that we can add the unique classes exactly once during the template compilation.

### <content> tag emulation

Current [implementation](https://github.com/angular/angular.dart/blob/master/lib/core_dom/transcluding_component_factory.dart) of <content> tag already supports default distribution. Further distribution models using selectors and dynamic updates need to be added to make the <content> tag behave as described in the [ShadowDOM spec](http://www.w3.org/TR/shadow-dom/#the-content-element).

### ShadowRoot injection

It is possible that component can ask for ShadowRoot using the [ShadowRootAware](https://docs.angulardart.org/#angular/angular-core-annotation.ShadowRootAware) interface. For this purpose we have implemented a custom [ShadowRoot](https://github.com/angular/angular.dart/blob/master/lib/core_dom/shadowless_shadow_root.dart) to emulate for the application.

# Caveats

# Security Considerations

# Performance Considerations / Test Strategy

# Work Breakdown

* [DONE] Transcluding Components
* [50%] Transcluding <content> (only support non-selector distribution)
  + Add support for multiple <content> tags in single template
  + Add support for partias distribution using CSS selectors
  + Add support for default content tag.
* [0%] CSS Encapsulation
  + CSS insertion service (since CSS need to be inserted in the head rather than in each element)
  + CSS parser and prefixing
  + Modify compiler phase to insert unique classes for each template.