APACHE SLING & FRIENDS TECH MEETUP

2 - 4 SEPTEMBER 2019

Everything is a Component - Site Development with Composum Pages

Ralf Wunsch, IST GmbH Dresden

who am i

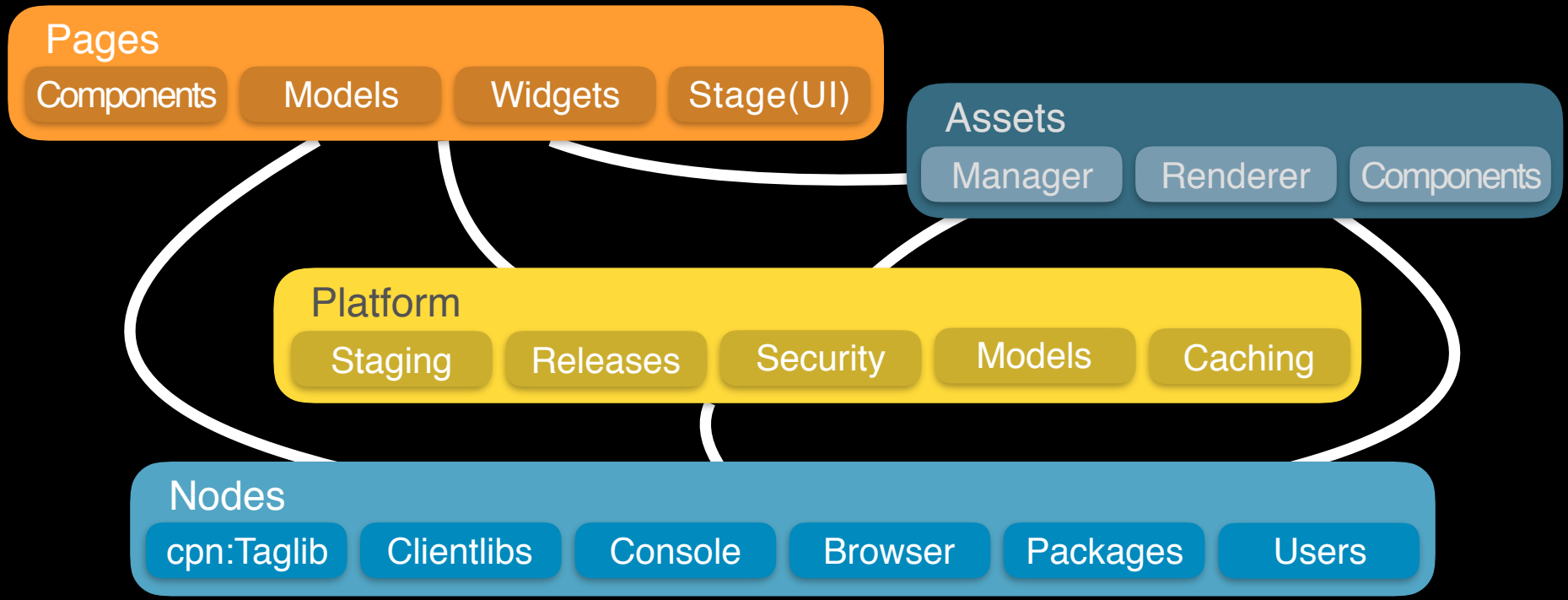
Ralf Wunsch

- Developer, Founder of the IST GmbH
- Sling enthusiast - Sling / AEM since 2009
- 2015 Composum Console - ‘Composum’
  - ‘warm up’ project, part of the Sling starter
- Composum ‘Pages’ / ‘Platform’ / …
  - first ideas 2016 - Release 1.0 2019 just now

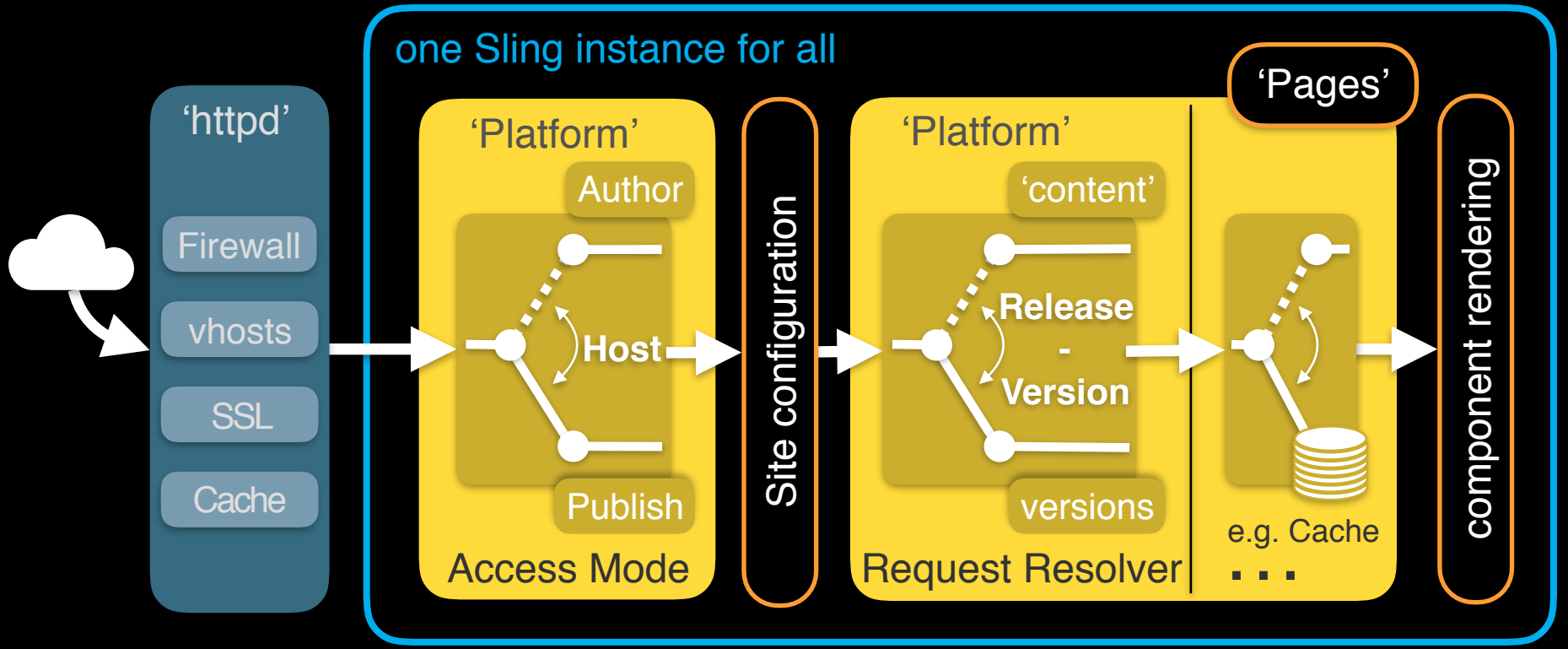
Composum

- ‚Nodes‘ - Browser, Package / User Manager - Core API
- ‚Platform‘ - Release / Access Management
- ‚Pages‘ - Content Management
- ‚Assets‘ - Asset Management (deferred - work in progress)
- see: https://www.composum.com/home.html
  https://www.composum.com/home/cloud.html
  https://cloud.composum.com/

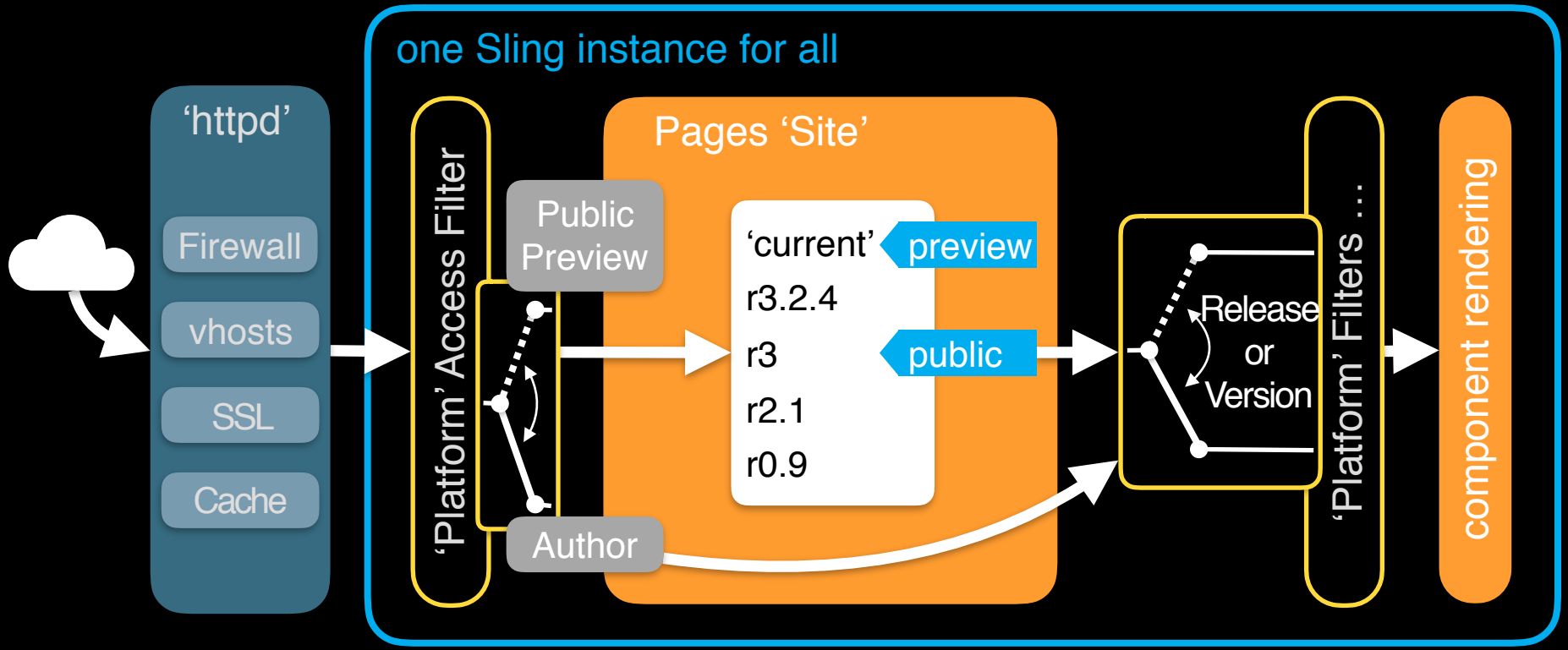
Composum Modules



‘Platform’


one Sling instance for all
‘httpd’
Firewall
vhosts
SSL
Cache
‘Platform’
Author
Host
Publish
Access Mode
Site configuration
‘Platform’
‘content’
Release
-
Version
versions
Request Resolver
‘Pages’
e.g. Cache
. . .
component rendering

‘Pages’ Releases



Pages - Entities

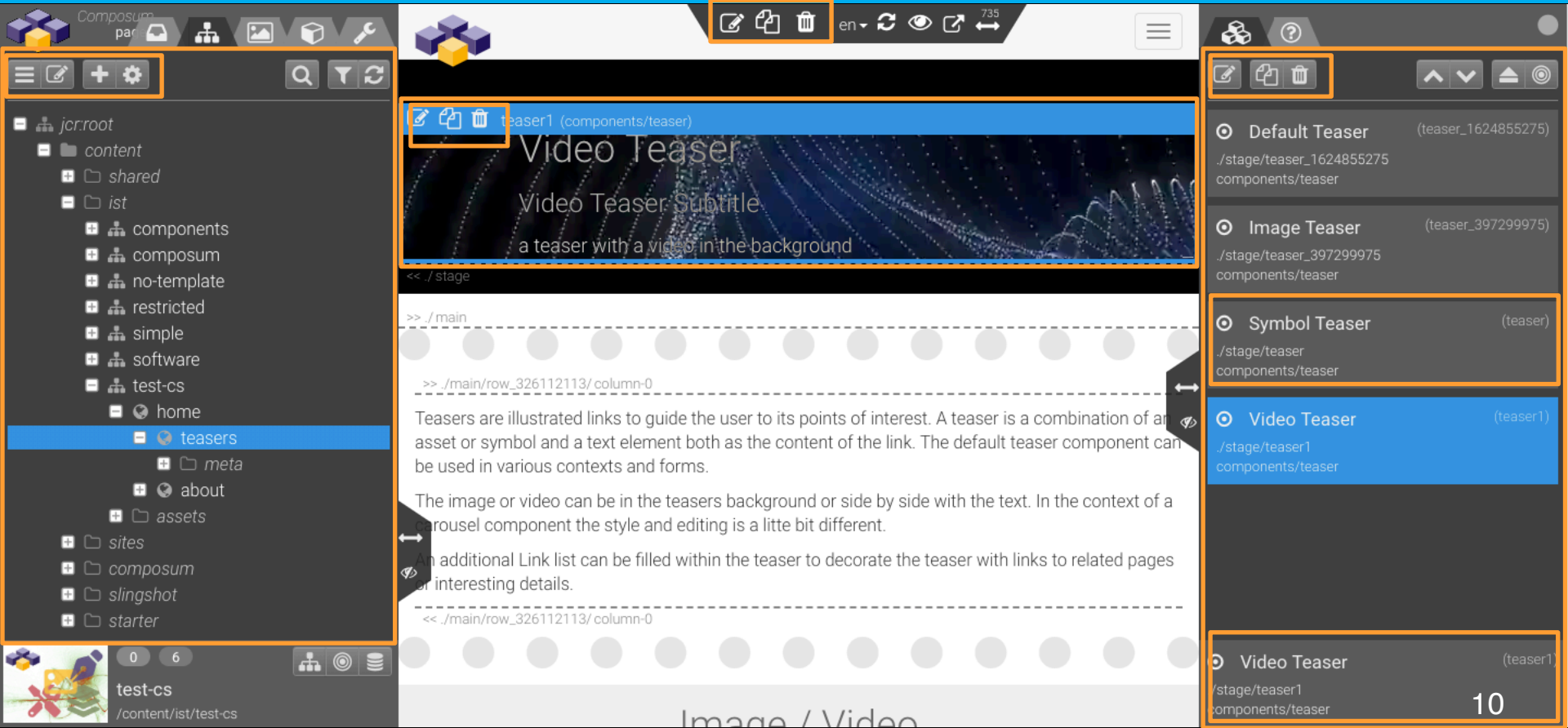
- Site - the whole website - released
  - Page - a single page - various languages
    - Container - arranges a set of elements
      - Element - more or less complex piece
  - Asset - file based resource - part of the site
- Templates - Site / Page - design policies

Pages - Features

- a CMS on top of Composum Nodes / Platform
- content release management and delivery (‘preview’ / ‘public’)
- language support on the property and/or the page level
- a modularised client libraries (CSS, JS)
- content structure templates and page content design rules
- extendable CMS user interface powered by Apache Sling
- standard component and widget set (extendable)
- content components development framework

today: ‘Pages’ components

Pages UI Components



Everything is a (Sling) component

- Pages, Containers, Elements
- Component Tiles / Toolbars
- Pages Tools
  - Navigation (left)
  - Context (right)
- Edit Dialogs / Widgets

Demo

an image component example

the component declaration

```
<jcr:root xmlns:…
  jcr:primaryType="cpp:Component"
  jcr:title="Image (Simple)"
  jcr:description="a single image element"
  category="[Image,Prototype,Simple]"
  sling:resourceSuperType="composum/…/components/element"/>

<jcr:root xmlns:…
  jcr:primaryType="cpp:Component" …
  componentType="cpp:Element"/>
```

the rendering template

```
<cpp:element var="model"
             type="com.composum.pages.commons.model.Image"
             test="@{model.valid||model.editMode}">
    <cpp:dropZone property="imageRef" filter="asset:image">
        <cpn:div test="${model.valid}" class="${modelCSS}_frame">
            <cpn:image class="${modelCSS}_picture"
                       src="${model.src}" alt="${model.alt}"
                       title="${model.title}"/>
        </cpn:div>
    </cpp:dropZone>
</cpp:element>
```

adaptTo()

Yes, a JSP taglib!

- keep component code as simple as possible
- avoid complex layering (edit hooks are embedded in the page)
- a ‘language’ on top of the scripting engine
- JSP as proven ‘old’ standard
- extendable - probably if HTL supports a ‘taglib’ feature
- Thymeleaf? - ideas are welcome
- Questions? - after the talk!

Demo

the various component types

content components / page components

- rendering template
- dialogs: edit [, create, delete]
- component tile and thumbnail
- component actions (toolbar)
- component help (content)
- use Pages Components as template

Demo

dialog widgets

- Sling Component
  - primaryType: cpp:Widget
  - rendering template (for the POST servlet)
  - widget model class (tag attribute handling)
  - CSS / JS scripts if necessary
- use Pages Commons standard widgets as example

Demo

a simple widget (‚confidentiality')

```
<jcr:root …
        jcr:primaryType="cpp:Widget"
        sling:resourceSuperType="composum/pages/commons/widget/select">
    <attributes
            jcr:primaryType="nt:unstructured"
            options="secret,confidential,internal,public"
            default="internal" required="true"/>
</jcr:root>

<cpp:widget label="Confidentiality" property="confidentiality"
            type="confidentiality" required="false"/>
```

Demo

how it works together

- containers and elements
  - tree structure navigation, DnD between tools
- sites and pages
  - current release - activation / deactivation
- releases and publishing
  - markers for ‘preview’ / ‘public’ release

Demo

try it…

getting started

- https://cloud.composum.com (for testing)
  - request your own tenant
  - create your first site, content and component
- for local installation see:
  https://www.composum.com/home/pages/setup.html
- this talk - see:
  https://www.composum.com/home/news/adaptTo.html

adaptTo()

sources on GitHub

- Pages (with Components module)
  - https://github.com/ist-dresden/composum-pages
- more Examples (e.g. the simple image)
  - https://github.com/ist-dresden/composum-prototype

Appendix

Taglib tags - Page tags

```
<html ${currentPage.htmlLangAttribute} ${currentPage.htmlDirAttribute}
      class="${currentPage.htmlClasses}" data-context-path="${slingRequest.contextPath}">
<cpp:head>
  <cpn:clientlib type="css" category="${currentPage.viewClientlibCategory}"/>
  <cpn:clientlib type="css" test="${not empty currentPage.themeClientlibCategory}"
                 category="${currentPage.themeClientlibCategory}"/>
  <cpn:clientlib type="css" test="${currentPage.editMode}"
                 category="${currentPage.editClientlibCategory}"/>
</cpp:head>
<cpp:body>
  <sling:call script="body.jsp"/>
  <cpn:clientlib type="js" category="${currentPage.viewClientlibCategory}"/>
  <cpn:clientlib type="js" test="${currentPage.editMode}"
                 category="${currentPage.editClientlibCategory}"/>
</cpp:body>
</html>
```

Example

adaptTo()

the edit dialog

```
<cpp:editDialog var="model" type="com.composum.pages.commons.model.Image"
                title="Image Properties">
  <cpp:widget label="Image" property="imageRef" type="imagefield"
              hint="the path to the image asset in the repository"/>
  <cpp:widget label="Title" property="title" type="textfield" i18n="true"
              hint="the title text normally shown on mouse over"/>
  <cpp:widget label="Alt Text" name="alt" type="textfield"
              value="${model.altText}" i18n="true"
              hint="the text for a text based view"/>
</cpp:editDialog>
```

Example

toolbar and tile example

```
<cpp:element var="model" type="com.composum.pages.commons.model.GenericModel"
             cssBase="composum-pages-component-tile">
    <sling:call script="_icon.jsp"/>
    <sling:call script="_title.jsp"/>
    <cpn:text value="${model.name}" format="({})" class="${modelCSS}_name"/>
    <cpn:text value="${model.pathHint}${model.name}" class="${modelCSS}_path"/>
    <cpn:text value="${model.component.typeHint}" class="${modelCSS}_type"/>
</cpp:element>
<cpp:editToolbar>
    <div class="composum-pages-tools_button-group btn-group btn-group-sm" role="group">
        <cpp:treeAction icon="edit" label="Edit" title="Edit the selected element"
                        action="window.composum.pages.actions.element.edit"/>
    </div> …
</cpp:editToolbar>
```

Example

content templates

- resolvers search path
- page and site templates
- predefined structure (copied)
- design rules (referenced)
- each template can reference another template
- use Pages Components templates as example

more…