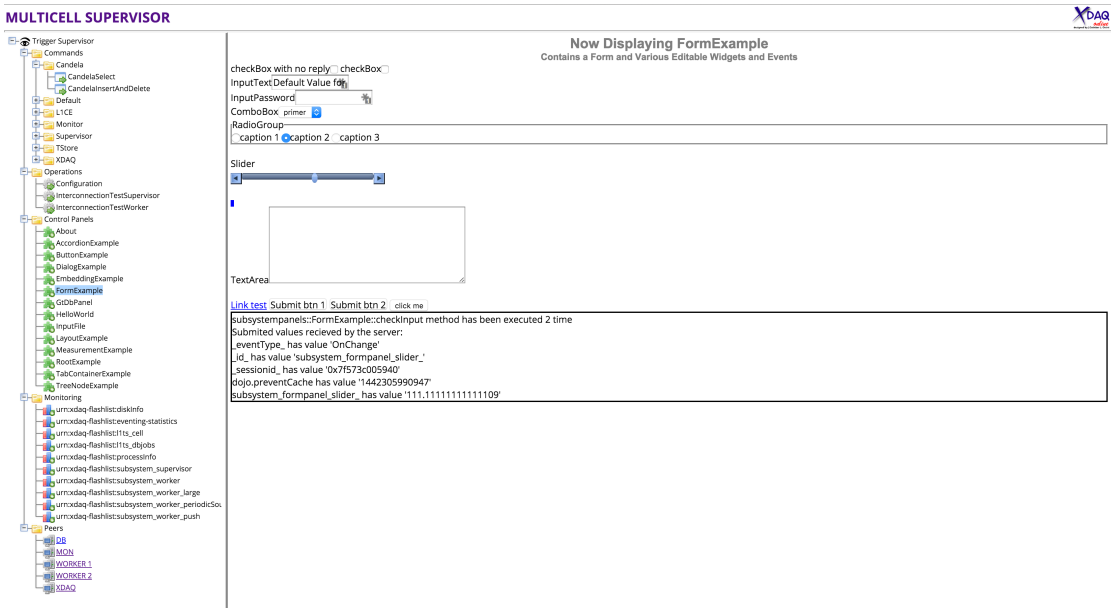
CERN Summer Student Report

The task

As a summer student I was tasked to update the current interface of the Trigger Supervisor. Which looks like this:



The group

- Glenn Dirkx, summer student (developer)
- Christos Lazaridis, Carlos Ghabrous (supervisors)

I would also like to thank Vangelis Paradas for his help, and Allesandro Thea and Joschka Lingemann for their enthusiasm.

Why a new TS GUI?

The human interfaces to Trigger Supervisor applications are built in the form of web applications, which is achieved with Dojo and C++ code. Dojo is a JavaScript library that allows users to define custom widgets (text boxes, buttons, sliders...), thereby extending basic HTML functionality.

The following example creates a slider using nothing but a standard tag and JavaScript:

```
<div dojoType="slider"></div>
```

This has been serving the project for quite some time, but some issues have been identified:

- The Dojo library is very outdated (v0.4, 9 years old) and no longer maintained:
- the risk of non-compatibility between the current TS GUIs and the browsers is very high, regardless whether they are used at the CMS control room or for development in general,
- browsers are starting to put up deprecation notices for old Dojo code. Hence, TS GUIs are at risk of losing support for newer browsers as time passes.
- Current C++ code is quite messy. Widgets can be defined, but the general layout of a page still needs to be generated in C++.
- Mixing CSS, JavaScript, and HTML into C++ makes code difficult to read and maintain and invites bad practices

Code like this is not uncommon:

```
 ajax::PlainHtml* plain25 = new ajax::PlainHtml();
 plain25->getStream() << "left";
 layout25->add(plain25);
 ajax::LayoutElement* layout26 = new ajax::LayoutElement();
 layout26->setPosition(ajax::LayoutElement::Left);
 layout26->set("style","color:white; background-color: black; ");
 layout2->add(layout26);
 ajax::PlainHtml* plain26 = new ajax::PlainHtml();
 plain26->getStream() << "left";
 layout26->add(plain26);
 ajax::LayoutElement* layout27 = new ajax::LayoutElement();
 layout27->setPosition(ajax::LayoutElement::Right);
 layout27->set("style", "color:green; width: 100px; background-
color:brown; ");
 layout2->add(layout27);
```

Explored options

Several alternatives have been explored:

Update Dojo to its latest version

- We cannot just switch to a newer Dojo version, as it underwent a major rewrite, which implies that the current GUIs won't work without rewriting
- We cannot run the old and a new Dojo version concurrently, as they both will try to define a global Dojo object containing all functionality
- The problem of mixing C++ with HTML, CSS, and JavaScript still isn't solved

User another JS solution (jQuery (UI), AngularJS, React, Sencha, ...)

- One of these would be a very good choice if the next option didn't exist
- There is no standard. History might repeat itself and we would be stuck with an old version of the framework again
- These don't work together. You generally can't use two JavaScript frameworks concurrently
- Most of these still don't solve the problem of mixing C++ with HTML, CSS, and JavaScript

Use Polymer, a framework based on Web Components (a HTML5 standard)

- Based on standards. It takes new RFC standards and combines them into a framework
- Works with old Dojo code without adjustments
- Allows us to very cleanly separate HTML, CSS, JavaScript, and C++
- Framework agnostic, it aims to play nice with others

Polymer as a new GUI for the Trigger Supervisor

About Polymer

Web Components consists of several separate technologies. You can think of Web Components as reusable user interface elements that are created using open Web technology.
They are part of the browser and so they do not need external libraries like jQuery or Dojo.

An existing Web Component can be used without writing code, simply by adding an import statement to an HTML page.
Web Components use new or still-developing standard browser capabilities, so a JavaScript Polyfill is needed for older browsers.

With a Web Component you can do pretty much anything that can be done with HTML, CSS and JavaScript, and it can be a portable component that can be re-used easily.

Web Components consists of these four technologies (although each can be used separately):

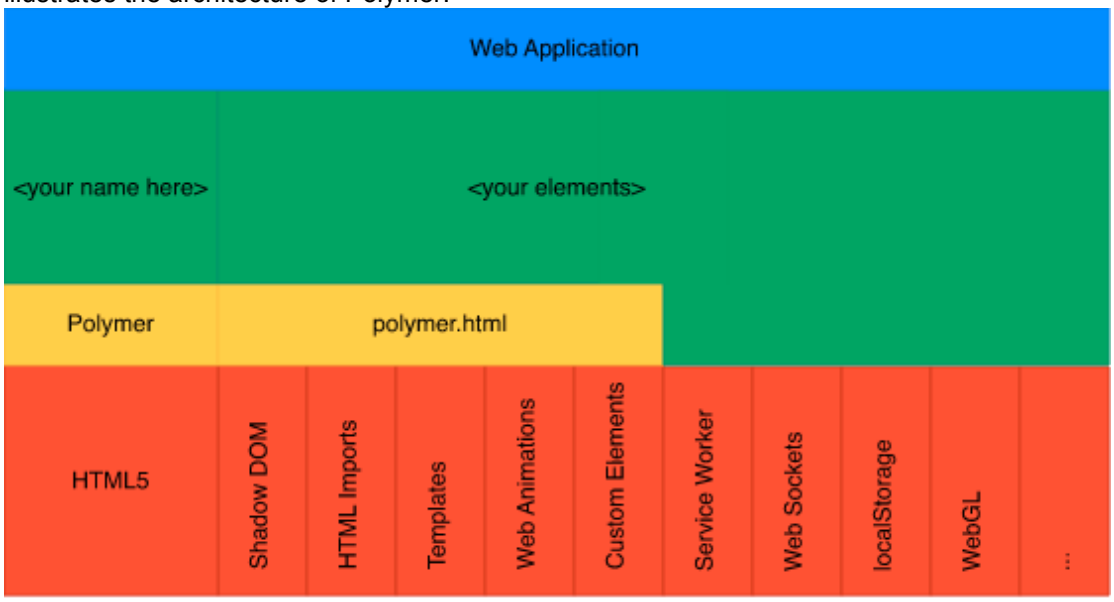
- Custom Elements
- HTML Templates
- Shadow DOM
- HTML Imports

The Web Components specifications define the following items:

- New HTML elements: <template>, <content> and <shadow>
- The associated API interfaces for the new elements: HTMLTemplateElement, HTMLContentElement and HTMLShadowElement
- Extensions to the HTMLLinkElement interface and the <link> element
- An API for registering custom elements, Document.registerElement(), and modifications of Document.createElement() and Document.createElementNS()
- New "lifecycle callbacks" can be added to the prototype that a custom element is based on
- A new CSS pseudo-class to style elements by default, :unresolved.
- The Shadow DOM: ShadowRoot and Element.createShadowRoot(), Element.shadowRoot

- An extension to the Event interface, Event.path
- Extensions to the Document interface
- For styling Web Components:
    - New pseudo-classes: :host , :host() , :host-context()
    - New pseudo-element: ::content

Polymer is a JavaScript library developed by Google and aims to ease the development of Web Components and help developers adhere to best practices. It combines core technologies of Web Components and makes them easier to use. The following figure illustrates the architecture of Polymer:



Combining Shadow DOM, HTML Imports, and Custom Elements in one library allows us to make a Web Component easily.

Consider the following code:

```
<dom-module id="my-element">
  <template>
   <p>I'm a DOM element. This is my local DOM!</p>
  </template>
  <script>    Polymer({      is: "my-element"    });  </script>
 </dom-module>
```

This element is:

- registered as a **custom element** via document.registerElement('my-element');
- The template is defined in the native **<template>** tag and will be injected into the **Shadow DOM** of the element.

Finally when we want to use this element, we use **HTML Imports**

```
<link rel="import" href="my-element.html">
```

Which presents you with the following result

Compatibility

The native support for the Web Components specs are currently as follows (note that we use a JavaScript library that fills all the gaps):

	Chrome / Opera	Firefox	IE	IE Edge	Safari
Templates	≥26	≥22	no	no	≥7.1
HTML Imports	≥36	on hold	no	no	no
Custom Elements	≥33	not enabled	no	no	no
Shadow DOM	≥35	not enabled	no	no	no

More updated information may be found here.

Webcomponents.js

Native browser support currently for these technologies is not enough sometimes. Every major browser has Web Components "in development" but Google Chrome and Opera are currently the only browsers implementing full support. To cope with this situation we use a polyfill, a JavaScript library following the Regressive Enhancement principle called webcomponents.js.
Using this we extend support for Web Components to all CERN supported browsers as well as all evergreen browsers

- Firefox (24)
- Internet Explorer 10+
- Google Chrome
- Safari 8

Technologies in Polymer

Polymer extends the default functionality of Web Components so we can make powerfull Web Applications with it.

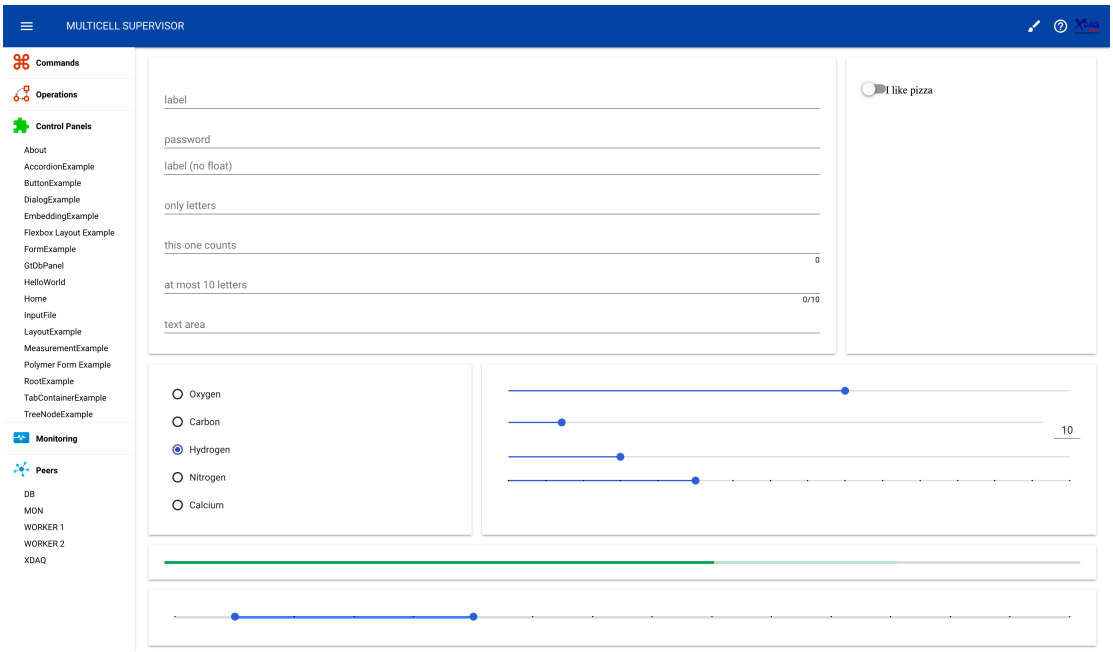
They are:

- properties
- data-binding
- an event system to notify you of changes
- behaviors
- a simplified element lifecycle
- template stamping
- native element extensions
- attribute (de)serialisation
- DOM Distribution
- DOM API
- the Shady DOM to mimic Shadow DOM in older browsers
- other small utilities

The result

You now have

- A much cleaner way to write code
- A much more modern interface
- A way to fix code that was broken because of the old Dojo library


MULTICELL SUPERVISOR
Commands
Operations
Control Panels
About
AccordionExample
ButtonExample
DialogExample
EmbeddingExample
Flexbox Layout Example
FormExample
GtDbPanel
HelloWorld
Home
InputFile
LayoutExample
MeasurementExample
Polymer Form Example
RootExample
TabContainerExample
TreeNodeExample
Monitoring
Peers
DB
MON
WORKER 1
WORKER 2
XDAQ
label
password
label (no float)
only letters
this one counts
0
at most 10 letters
0/10
text area
I like pizza
Oxygen
Carbon
Hydrogen
Nitrogen
Calcium
10