27
Mar

Components and SOA for Industrial Web Apps – Part 2

By Mike Bradley, Sr. HyperUA, Industrial Web, Open Source, SOA, Web Services Comments are off

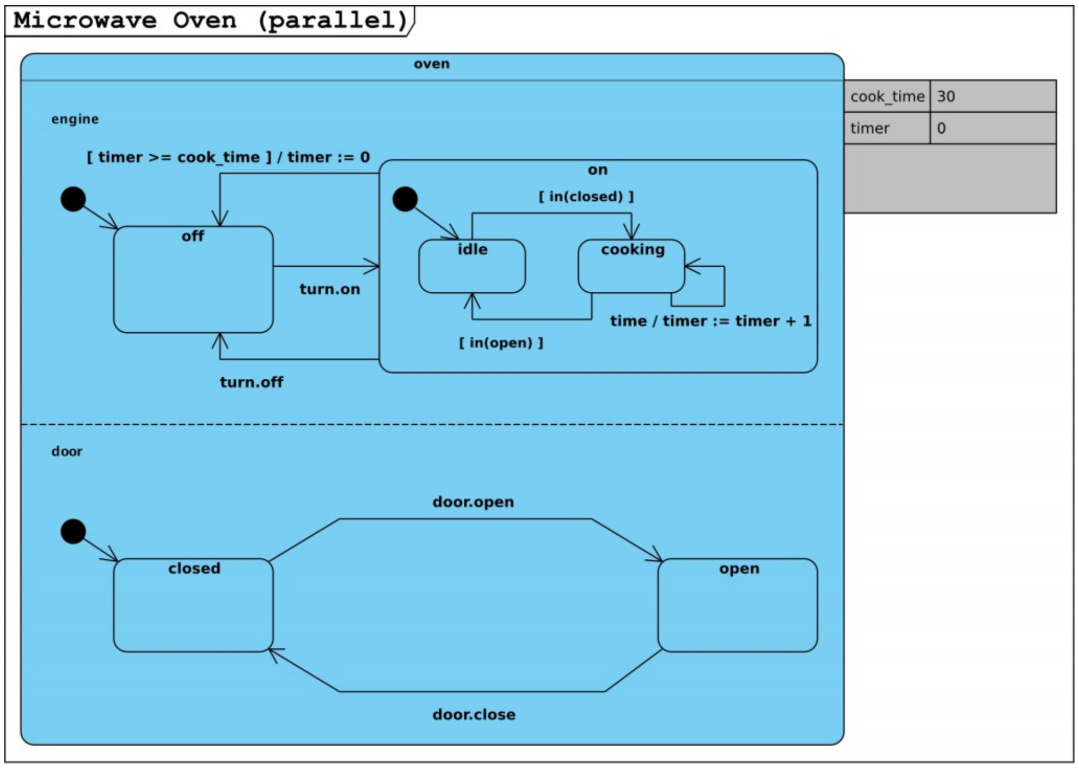
In my previous blog post, I announced that Projexsys is working to leverage Google Polymer as the basis of a new SDK for use by OEMs and integrators.

Our development team's initial forays into Polymer convinced us that Web Components are indeed the future, and hold the brightest promise for Industrial Web apps. The premise is simple: developers extend the vocabulary of HTML with custom elements that cleanly encapsulate templatable, highly reusable bundles of JavaScript and markup. Those custom elements can be individually tested, and then stacked together to create more complex reusable pices, and in turn whole applications. And all while maintaining the option of full (or minimal) customization on per-component and per-app bases.

However, while Polymer blows open the doors to component-driven development, it does little to ease the complexities of developing highly concurrent applications. So our team at Projexsys is taking it further…

Introducing Ply™

Ply is Projexsys' forthcoming "pure Web" (no browser plugins!) SDK for Industrial Web Apps. Ply builds on Google Polymer by embodying a dynamic Statechart interpreter in terms of custom elements, allowing application control flow to be expressed declaratively, along with little bits of imperative "glue" code. Statecharts are a tried-and-true, standardized means of describing concurrent applications. Their origins lie in the innovation of professor and businessman David Harel, who invented them in the 1980s as a tool for avionics engineers to formally model their complex, interconnected systems of hardware and software. We believe that same formalism is well-suited to today's complex Web apps – which interconnect a myriad of users and services – and will offer a much-preferred alternative to hand-rolled imperative control flow.



cook_time	30
timer	0

The statechart above describes the behavior of a simple microwave oven and was created with Visual Paradigm, a popular desktop application for authoring UML diagrams.

Below is HTML + JavaScript source code which describes the same microwave oven, i.e. encodes the same statechart, using the building blocks (custom elements) provided by Ply:

microwave-oven-parallel.html

```
<p-statechart>
  <!-- Immutable hash-maps are used for charts' environments and
       datamodels, with isolated mutable environments passed to
       concurret actions; this allows for robust "big-step"
       semantics in Ply's statechart interpreter -->
  <p-environment>
    <p-datamodel>
      <p-data id="cook_time" expr="30"></p-data>
      <p-data id="timer" expr="0"></p-data>
    </p-datamodel>
  </p-environment>

  <p-parallel id="oven">

    <p-state id="engine">

      <p-state id="off">
        <p-transition event="turn.on" target="on"></p-transition>
      </p-state>

      <p-state id="on">
        <p-transition event="turn.off" target="off"></p-transition>
        <p-transition target="off">
          <!-- condition expression as a child element; a JavaScript
               expression that is cast to true/false according to the
               standard rules of the language -->
          <p-condition>
            get(data, 'timer') >= get(data, 'cook_time');
          </p-condition>
          <p-action>
            mutable.assoc(env, 'data', assoc(data, 'timer', 0));
          </p-action>
        </p-transition>

        <p-state id="idle">
          <!-- condition expression as an element attribute -->
          <p-transition cond="in('closed')" target="cooking">
          </p-transition>
        </p-state>

        <p-state id="cooking">
          <p-transition cond="in('open')" target="idle"></p-transition>
          <p-transition event="time" type="internal">
            <!-- action code is JavaScript, which will be automatically
                 wrapped in a function body and provided with various
                 helper-functions -->
            <p-action>
              mutable.assoc(
                env,
                'data',
                update_in(data, ['timer'], util.inc)
              );
            </p-action>
          </p-transition>
        </p-state>
      </p-state>

    </p-state>

    <p-state id="door">

      <p-state id="closed">
        <p-transition event="door.open" target="open"></p-transition>
      </p-state>

      <p-state id="open">
        <p-transition event="door.closed" target="closed"></p-transition>
      </p-state>

    </p-state>
  </p-parallel>
</p-statechart>
```

Note: the above sample is somewhat simplified, i.e. it does not provide a complete overview of Ply's usage and features, including how such statecharts are embedded within and intended to be "wired to" host components, which render the managed data into visuals and forward events and data changes to the Ply components. Forthcoming code samples and demos will complete the picture, so stay tuned.

Our conventions and interpreter are inspired partly by the SCXML standard, and one of Ply's core aims is providing a straightforward mechanism for consuming and implementing services, whether they are expressed as intra-component APIs, or as Web services using WSDL and SOAP/XML, WADL, or plain REST. The benefit is that .Net and other developers commited to service-oriented architecture (OPC UA is itself billed as an SOA) can allow their services mindset to directly inform their client-side Web apps, and vice versa! As rapid prototyping and iteration with microservices becomes a mainstay in the development of inter-connected applications, the ability to painlessly leverage those same SOAs directly in the browser will become an increasingly big deal.

Modular and OEM-friendly

The core Ply framework will be open source under a business-friendly reciprocal license, with the option of a non-repricocal commercial license. Our OEMs will be able to build proprietary kits and modules around Ply, for sale through or to their vendors and integrators, and Projexsys will provide paid support to commercial licensees. As the core framework solidifes, Projexsys will be looking to builds its own reference suite of modules and advanced tooling — visual authoring, integration testing, profiling and optimization — to enhance development with Ply, which will likewise be available for OEM packaging.

In the next blog post, I will further explore Projexsys' business vision for Ply.

← Components and SOA for Industrial Web Apps – Part 1

Components and SOA for Industrial Web Apps – Part 3 →

PROJEXSYS
Powering the Industrial Web

Our Software OEM & System Builder partners will use HyperUA to tailor apps that enable their End Users and System Integrators to *"Connect OPC directly to the Web!"*

Contact us to schedule a live demo.

Contact Information

Address
211 Crestwood Drive
Johnson City, TN 37601
United States

Phone / Fax
888 960 0743
+1 423 202 9120

Email
info@projexsys.com

OPC News

CC-Link Partner Association announces OPC UA companion specification for "CSP+ for Machine" technology

SAP Leonardo Enterprise Wide IT/OT Integration

OPC Interoperability Workshop, North America

Secure Configuration and Operation of OPC UA

VDMA's Industrie 4.0 Communication Guideline Based on OPC UA

Blog Archive

April 2014
March 2014
June 2013
May 2013
April 2013

Home | About | HyperUA™ | News | Contact