Accessible Rendered Item (ARI) Technical Specifications

Prepared by: Ali Abedi, UCLA/CRESST
Last updated: 4/11/2016

Table of Contents

1. Introduction

The Accessible Rendered Item (ARI) format is a new concept for describing computer-administered assessment items. The concept is to provide a standardized mechanism to transform items from their native authoring formats or exchange formats (e.g., IMS QTI, SmarterApp Assessment Item Format) into a common HTML plus JavaScript format used to describe item rendering and behavior. The ARI rendering process and output format is much like how documents can be rendered from their native formats into PDF documents. The ARI format results in reduced implementation complexity for the test delivery system (it is implemented using only common web technologies), more control over how an item renders (provides rendering control not present in item authoring and exchange representations), and increased opportunity for innovation (limited assessment item specific technology is present in the ARI item format, new and extended features can be supported directly with no changes in the core technology).

2. Design Goals

The ARI Technical Working Group was convened by UCLA CRESST and met at the CRESST offices on November 12-13, 2014. The team concluded on the following design goals:

- Common and interoperable.
  - Multiple vendors can deliver assessments in this format.
  - Items in different authoring and exchange formats can be seamlessly integrated
- Supports advanced item types.
  - Constructed responses.
  - Increased Depth of Knowledge (DOK) items.
- Supports accessibility features.
  - Items dynamically customizable according to individual student needs.
- Items render consistently across multiple browser/web client implementations.
- Supports innovation.
  - Allows field-testing of experimental interaction types without changes to
  - Can capture rich item interaction information including data that doesn’t
- Simple and inexpensive client side implementation
- Supports multiple item authoring environments
- Supports convertibility of items from other item formats

- Portability
    - Functions across a variety of devices
    - Resilient to future changes to browsers
    - Student responses are captured in a portable format
    - Machine scoring rubrics can be applied consistently across multiple implementations
- Digestible by a standards body such as IMS
    - Unencumbered by intellectual property or patent restrictions.

3. System Overview

The requirements for this service was to create a framework that will allow for modular item creation, item delivery, and client side rendering of items. One approach to that was to use web components and Polymers. It's an existing set of standards (web components) and framework (Polymers) that allow for modular web apps inside of an HTML page as well as existing standards for importing different CSS/JS/HTML files which each item will have it's own. It also allows for each item to have it's own namespace and not to pollute the global namespace.

3.1 Required Technology Stack

3.1.1 HTML5

Used in creation of the client side application.

3.1.2 CSS3

Used in styling of client side application.

3.1.3 JavaScript

Used in programming application behavior of the client side application.

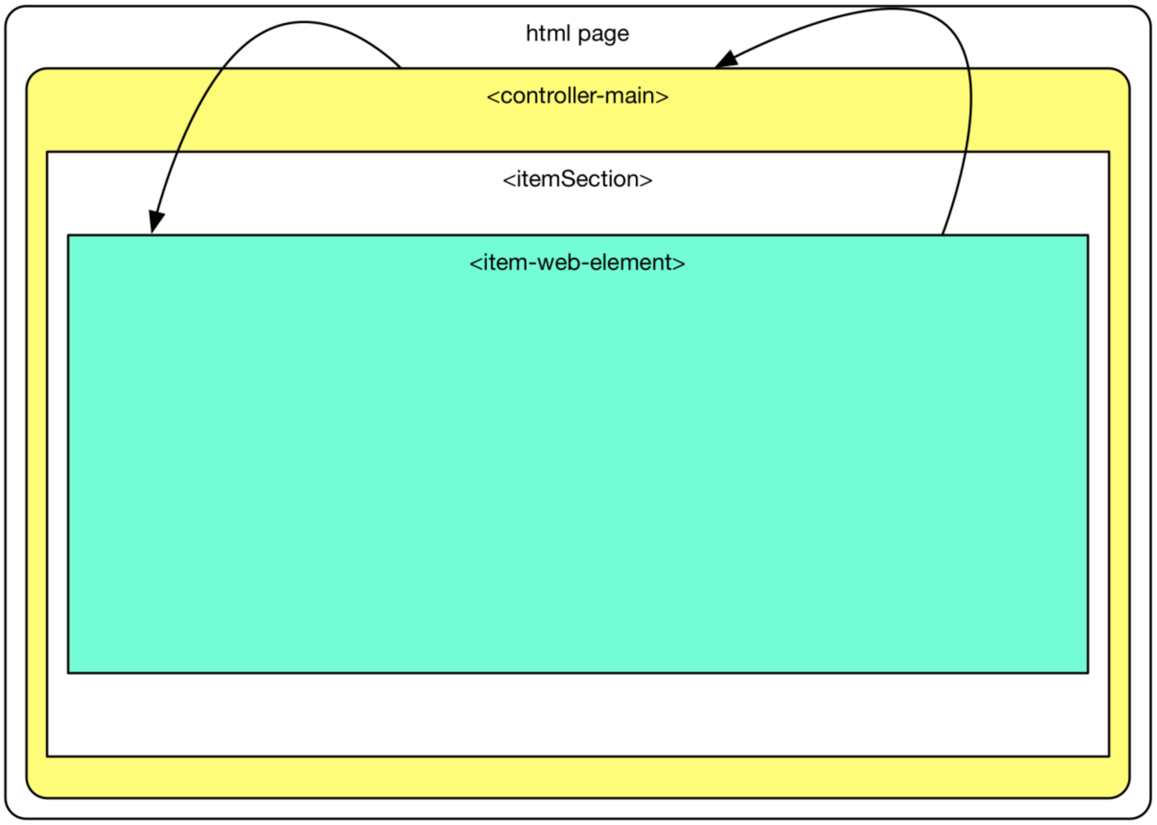
3.1.4 jQuery

Abstraction and functions library used on the client side.

3.1.5 Polymer

Web Components framework. Individual items will be web components. This will allow modular creation and implementation of each item while also preserving global namespace integrity and separation.

3.2 DOM Elements



3.3 Process Overview

3.3.1 Html Page

One of the goals of the system is to keep the base system as minimal as possible. This way any item can use their own frameworks with as minimal conflicts as possible. For that reason the html is loaded with only the polymer and jquery framework.

3.3.2 Controller-main

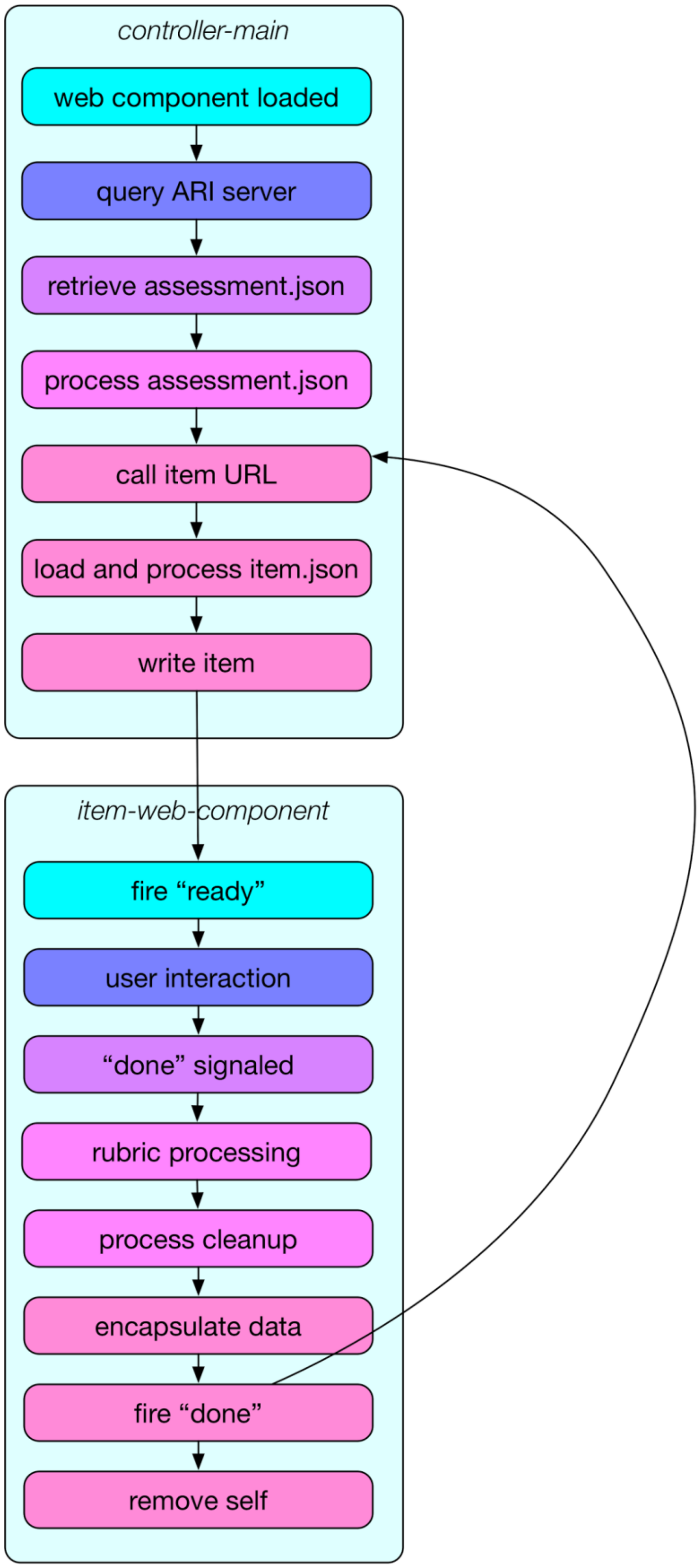
controller-main is the assessment controller. It's role is to load in the assessment information, then with that information write the item web component on the HTML page and control the flow of the assessment, from loading the next question and saving item data such as state and/or student choices.

3.3.3 ItemSection

This is the div DOM element where item-components are loaded. It is located in the controller-main template and subsequently a child of controller-main. When an item is loaded it's item web component URL is written here. When the item component is written polymer will automatically load the polymer element. Once an item is done it will be removed from the DOM from the *controller-main* clean-up process.

3.3.4 Item-web-components

Item-web-components should be abstracts for instantiating items through item.json files. There should be 6 basic item types: simple choice, prompt body, item body, equations, graph, EBSR, HTQ, and NL. Other item types can be added. Instantiation (prompt, questions, directions, etc) of the item types are done through the data from the item.json file (see item.json for more detail). Item.json should contain all the data an item will need and no specific item data should be contained in an item-web-component itself.



3.4 Process Order

3.4.1 web component loaded

The polymer frameworks loads the controller-main item into the DOM. Once loaded into the DOM, controller-main will run through it’s object instantiation such as event listeners (see events section 4.2 for more detail).

3.4.2 query ARI server

Controller-main will query the ARI server to see what assessment to serve up for the user.

3.4.3 retrieve assessment.json

The ARI server will send the assessment.json from a server API call (see assessment.json for more detail). Information there will contain the URL for the item location and assessment name.

3.4.4 process assessment.json

Controller-main will read the assessment.json file and parse all the values from it. Among the values in the json file will be items URL. Items URL will be called to get the next available item.json file and an assessment finished object if there is nothing left.

3.4.5 Call Item URL

Controller-main will now call the item URL and retrieve the next available item.json file and an assessment finished object if there is nothing left.

3.4.6 Load and Process item.json

Controller-main will load the item.json and parse its values. The two important values that will be parsed are ari_url and wc_name.

3.4.7 Write Item

Ari_url will be the location of the item-web-component that will be written in the itemSection and wc_name will be the name the polymer component is already declared as. Controller-main will write the item-component onto the HTML DOM under itemSection. Once this happens Polymer will automatically load the ready() method of the item-web-component.

3.4.8 Fire "ready"

Once the item-web-component has finished everything in the ready() method, the item will fire a "ready" event which will be captured by controller-main.

3.4.9 User Interaction

At this point the user will interact with the item-web-component. Data should be captured during this time and stored internally.

3.4.10 "Done" Signalled

Either the user will initiate a "done" signal by either going to the next/previous item or the controller-main has determined item is done. When this event has been triggered it should begin the finishing the item process.

3.4.11 Rubric Processing

The item-web-component needs to send the student data through it's rubric. Once the rubric processes the student data it will need to return its results to the item-web-component. The item-web-component will need to store this along with the raw data to send back to the server.

3.4.12 Process Cleanup

At this stage the item-web-component will consolidate data and have it ready to be sent back to controller-main as well as clean up any DOM related elements.

3.4.13 Encapsulate Data

Ensure all data is in an object ready to be sent back to the controller-main.

3.4.14 Fire “Done”

Item-web-component at this point will need to fire off “done” event which will be captured by the controller-main. Controller-main will send any data back to the server as well as this event will trigger the call item URL method in the controller-main. If there is a new item, the process starts from there, and if controller-main receives and end message the assessment will terminate.

3.4.15 Remove Self

Item should invoke the polymer remove() method at this point to remove itself from the DOM and unregister any polymer or HTML events.

4. Web-Components

4.1 Controller-Main

The *controller-main* web-component is the assessment controller. It’s in charge of loading the assessment information, i.e. assessment name, item locations, interaction mode, etc. Its other responsibilities include writing the item web-component on the web page, sending and retrieve data to and from the server, going to the next or previous item.

4.1.1 Events

In order to prevent global scope pollution and maintain item isolation, communication between web components will go through polymer custom event listeners. There are a few existing events that all items need to accommodate:

4.1.1.1 ready

The controller-main listens to the “ready” event. Items-web-components need to fire the “ready” event when the item is fully loaded. Actions can be added to the controller-main to handle any actions to be performed when the item is finished loading.

4.1.1.2 done

The controller-main listens to the “done” event. The “done” event will expect a data object which will contain student data for the item. “Done” event signals when an item is done and this begins post-completion events to process from the controller-main. Post-competition events can include, saving data, sending to server, and going to the next item. The item itself is responsible for passing the student data to it’s own rubric. Post rubric processed and raw data should be passed into the “done” event.

4.1.1.3 next

The controller-main listens to the “next” event. The “next” event will expect a data object which will contain student data for the item. “Next” events signals when a user has clicked on the next button, if available, and for for the controller-main to take the user to the next item. Item state for the current data, which has been sent through the data object, can be sent to the server for subsequent load when the user comes back to this item.

4.1.1.4 previous

The controller-main listens to the “previous” event. The “previous” event will expect a data object which will contain student data for the item. The data object can container the item state for the current data, which can be sent to the server for subsequent load when the user comes back to this item. “Previous” events signals when a user has clicked on the next button, if available, and for for the controller-main to take the user to the next item.

4.1.1.5 custom events

Custom events can be added to both items and controller-main.

4.1.1.5.1 examples

4.1.1.5.1.1 *listening*

```
<script>
    Polymer({

      is: 'controller-main',

      registerEvents: function() {
        … other events
        this.addEventListener('EVENT1', {EVENT1_OBJECT: true});
      }

    });

  </script>
```

4.1.1.5.1.2 *firing*

```
<script>
    Polymer({

      is: 'item-main',

      sentEvents: function() {
        this.fire('EVENT1', {data: {'status':'ready'}});
      }

    });

</script>
```

4.2 Methods

4.2.1 Ready

The ready event is called by the polymer framework when the controller-main has loaded. Any init event or functions needing to run must be done here.

4.2.2 registerEvents

All the event firing and listening are called here from the ready method.

4.2.3 loadAssessment

The ready event will call this method. This method will load the assessment.json file.

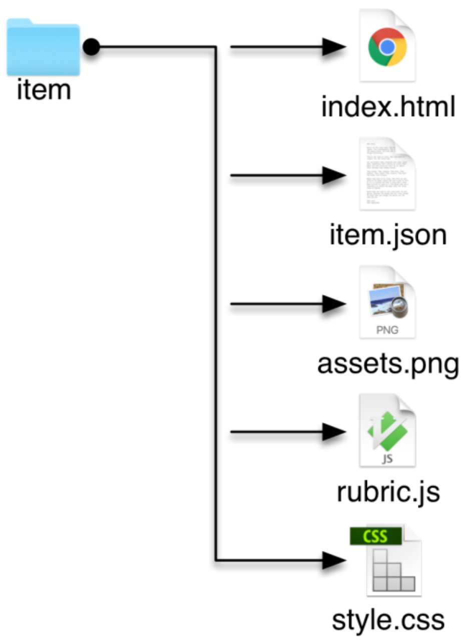
4.2.4 loadItem

Once the assessment is loaded, the loadAssessment will call this method to load the item.json file.

4.2.5 processItem

Once the item.json has been loaded the loadItem will call this method to write the item-web-component to the DOM.

4.3 item-web-component



4.3.1 structure

At a minimum items should be packaged as illustrated. One of the goals set out by the system was portability which this format allows.

4.3.1.1 index.html

Index.html file should include the web-element, along with polymer js code.

4.3.1.2 item.json

This file should contain all the data pertinent to the item itself such as prompt, location of accessibility files, question data, etc.

4.3.1.3 assets.png

Assets such as images, audio, videos should be included here.

4.3.1.4 rubric.js

Rubric.js should be the JS code for the rubric itself. The item will need to know how to access it and pass data to it as well as what to expect from it. The JS code here will be used to process the student data.

4.3.1.5 style.css

Style.css should include all the styling information for the item itself. Accessibility accommodations should also have been made here such as color contrast.

4.3.2 Creating Custom Items

For creating custom items, there are a few requirements.

4.3.2.1 name and id

The first would be to register and obtain a unique id and unique web-component name (wc_name) from the main ARI system framework. The web-component name (wc_name) will be used as the polymer component name. The wc_name will be sent to the controller_main, through the item.json file, and the controller_main will write the web-component from the given name (wc_name) and URL (ari_url).

4.3.2.2 rubric

Item-web-components need to process the student data for scoring. That's done through a rubric. An item will handle it's own scoring and it will send the resulting object back through the "done" fire event, both raw and rubric-processed data.

4.3.2.3 accessibility features

Item-web-component will also need to be able to handle all forms of accessibility. Specific student needs will be sent to the item-web-component from the student_data.isaap object in the assessment.json file. Item-web-component will need to read the data and make appropriate changes to the item rendering either through the item js or item css.

4.3.2.4 Data

4.3.2.4.1 Inputs

Data regarding the assessment, item, and student will be passed to the item-web-component from the controller-main. Item-web-component will need to accept these objects and if required used them.

4.3.2.4.2 Outputs

The item-web-component will need to send both the raw (student entry) and processed (rubric) data from the item to the controller-main through the "done" fire event.

4.3.2.4.2.1 *example*

```
var raw_data = { …. };
var rubric_data = { ….. };
var student_data = {
    raw_data: raw_data,
    rubric_data: rubric_data
};
this.fire('done', {data: student_data});
```

5. Assessment.json

The assessment.json file is sent to the web-component *controller-main*. This is the only input for *controller-main*. *Controller-main* calls the items_URL to retrieve the item.json file. If a termination code is returned then the assessment is finished. If the item.json file contains the location to an item web-component, then the item web-component is retrieved and loaded onto the page.

5.1 Elements

5.2.1 name

5.2.1.1 Description

This is the name of the assessment. Element should be a human readable string. Might be used in titles and reporting.

5.2.1.2 Example

"Smarter Grades 6-8"

5.2.2 items_url

5.2.2.1 Description

URL location of the formative json. Each call to this URL should produce an items.json file for an individual question.

5.2.2.2 Example

http://ari.sbac.com/api/ari/getItem

5.2.3 student_data

5.2.3.1 Description

Student data object which will contain:

5.2.3.2 children objects

5.2.3.2.1 student_id

This will be the unique student identifier

5.2.3.2.2 isaap

The ISAAP data object for the student.

5.2.3.3 Example

See objects section

5.2.4 test_data

5.2.4.1 Description

This is an option object. This object will be sent to the item. Any data pertinent to the assessment or test should be placed here. The system will not use this data.

5.2.4.2 Example

{‘random_data’: ‘lorem ipsum’}

5.3 example

```
{
	"name": "Smarter Grades 6-8",
	"items_URL": "/api/ari/getItem",
	"student_data": {
		"student_id": 1,
		"isaap": {
			"colorContrast": true
		}
	},
	"test_data" : { }
}
```

6. Items.json

The items.json file is what gives an item web-component information to populate the item.

6.1 Requirements

Function: rubric - before the done event is fired item web-component needs to send it's data through it's own rubric. The result of this call is sent to the item-controller.

6.2 Elements

6.2.1 ari_url

6.2.1.1 Description

Location of the item web-component.

6.2.1.2 Example

http://ari.sbac.com/items/q1/index.html

6.2.2 wc_name

6.2.2.1 Description

The name of the item web-component. This needs to be a unique name registered in the ARI system.

6.2.2.2 Example

"Item-2395"

6.2.3 Item

6.2.3.1 Description

This is the Smarter Balanced item data. The item data is a conversion of the existing item XML through the item XML conversion tool that is available at a later date.

6.2.3.2 Example

```
{
  "itemrelease": {
      "format": "mc",
      "version": "10",
      "bankkey": "187",
      "attriblist": {
        "attrib": [
          {
            "attid": "itm_item_id",
            "name": "Item: ITS ID",
            "val": "1448"
          },
          "stem": "<p style=\"\">Coffee costs $2 <span id=\"item_1448_TAG_5\" class=\"its-tag\"
data-tag=\"word\" data-tag-boundary=\"start\" data-word-index=\"1\"></span>per<span class=\"its-
tag\" data-tag-ref=\"item_1448_TAG_5\" data-tag-boundary=\"end\"></span> pound at a coffee shop.
<span id=\"item_1448_TAG_6\" class=\"its-tag\" data-tag=\"word\" data-tag-boundary=\"start\"
data-word-index=\"2\"></span>Which<span class=\"its-tag\" data-tag-ref=\"item_1448_TAG_6\" data-
tag-boundary=\"end\"></span> graph represents this <span id=\"item_1448_TAG_7\" class=\"its-tag\"
data-tag=\"word\" data-tag-boundary=\"start\" data-word-index=\"3\"></span>situation<span
class=\"its-tag\" data-tag-ref=\"item_1448_TAG_7\" data-tag-boundary=\"end\"></span>?</p>",
        "optionlist": {
          "option": [
            {
              "name": "Option A",
              "val": "<p style=\"\"><img id=\"item_1448_graphics1\"
src=\"/static/ari3/items/q1/item_1448_v10_graphics1_png256.png\" width=\"221\" height=\"252\"
style=\"vertical-align:baseline;\" /></p>",
              "feedback": "<p style=\"\">&#xA0;</p>"
            },
            {
              "name": "Option B",
              "val": "<p style=\"\"><img id=\"item_1448_graphics2\"
src=\"/static/ari3/items/q1/item_1448_v10_graphics2_png256.png\" width=\"221\" height=\"252\"
style=\"vertical-align:baseline;\" /></p>",
              "feedback": "<p style=\"\">&#xA0;</p>"
            },
            {
              "name": "Option C",
              "val": "<p style=\"\"><img id=\"item_1448_graphics3\"
src=\"/static/ari3/items/q1/item_1448_v10_graphics3_png256.png\" width=\"221\" height=\"252\"
style=\"vertical-align:baseline;\" /></p>",
              "feedback": "<p style=\"\">&#xA0;</p>"
            },
            {
              "name": "Option D",
              "val": "<p style=\"\"><img id=\"item_1448_graphics4\"
src=\"/static/ari3/items/q1/item_1448_v10_graphics4_png256.png\" width=\"221\" height=\"252\"
style=\"vertical-align:baseline;\" /></p>",
              "feedback": "<p style=\"\">&#xA0;</p>"
            }
          ]
        },
      }
    }
  }
```

6.3 example

```
{
    "ari_url": "/static/ari3/items/q1/index.html",
    "wc_name": "item-2395",
    "item": { … }
}
```