Accessible Rendered Item (ARI) Format

Software Requirements Specification Document

Document History and Distribution

1. Revision History

Revision #	Revision Date	**Description of Change**	**Author**
1.0	6/30/2015	Document Creation	UCLA/CRESST

2. Distribution

Recipient Name	Recipient Organization	**Distribution Method**

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).

1.1 Purpose

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 into an assessment.
- 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 assessment delivery code.
  - Can capture rich item interaction information including data that doesn't contribute to the item score.
- 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.

1.2 Scope

The software will be developed using:

- HTML5
- CSS3
- JavaScript
- jQuery
- Raphaeljs
- g.Raphaeljs
- Polymer
- Python
- Pyramids
- SQLAlchemy.

HTML5
Used in creation of the client side application.

CSS3
Used in styling of client side application.

JavaScript
Used in programming application behavior of the client side application.

jQuery
Abstraction and functions library used on the client side.

Raphaeljs
SVG library used for graphical items.

g.Raphaeljs
Raphael library used for charts.

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.

Python
Used in the backend server.

Pyramids
Lightweight python web framework.

SQLAlchemy
Python database abstraction ORM. Allows the ability to write sql as Python objects and implement it's execution in any database.

The service will allow for delivery of Items to the client, where the client side will render the Item as well as a framework for vendors to create items and incorporate them into the system. The framework will implement items as web components. This will allow for modular loading and creating of items as well as a framework that will preserve the global name space and existing methods for file importing.

1.3 Definitions, Acronyms, and Abbreviations.

HTML - HyperText Markup Language
CSS - Cascading Style Sheets
Web Components - set of standards currently being produced by Google engineers as a W3C specification that allow for the creation of reusable widgets or components in web documents and web applications.
ORM - a programming technique for converting data between incompatible type systems in object-oriented programming languages.
Back end - Serves indirectly in support of the front-end services, usually by being closer to the required resource or having the capability to communicate with the required resource.
Front end - Application users interact with directly. It serves as an interface between the user and the back end.
Controller – The main web component that handles all item functionality, user data, and back end communication.

Assessment Manifest – A JSON file with assessment data such as, location of each item, order of each item, assessment name, etc.
Item Manifest – A JSON file with the item data such as, item type, questions, options, rubric, etc.
CDP - Controller Data Payload; JSON file of user profile data, previous session data, and assessment manifest.
Item Web Component – web component of loaded item.

1.4 References

HTML5 - http://www.w3.org/TR/html5/
CSS - http://www.w3.org/Style/CSS/specs.en.html
JavaScript - http://www.ecma-international.org/publications/standards/Ecma-262.htm
jQuery - https://jquery.com/
Polymer - https://www.polymer-project.org/1.0/
Python - https://www.python.org/
Pyramids - http://www.pylonsproject.org/
SQLAlchemy - http://www.sqlalchemy.org/
RaphaelJS - http://raphaeljs.com/
g.RaphaelJS - http://g.raphaeljs.com/

2. The Overall Description

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.

2.1 Product Perspective

The ARI prototype in its current state is independent of other systems. However, for it's real-life implementation it will need to be tied into multiple systems just as IMDB, user systems, etc.

2.1.1 System Interfaces

The ARI prototype has no existing system interfaces. For the prototype everything is self-contained.

2.1.2 Interfaces

The front end GUI will directly interact with the users who are: students, teachers, administrators, parents, etc. Most of the interaction will be determined by the item itself as well it's appearance.

2.1.3 Hardware Interfaces

Any hardware that can run "modern browsers" will be supported. There might need to be special consideration for items that require more intensive system resources to check the user's hardware to match requirements for optimal use.

2.1.4 Software Interfaces

HTML v5:

- HTML version 5 is used to markup the structure of the client interface. Version 5 in particular is important because it's support for web elements, which is a foundation of this system.

CSS v3:

- CSS will be used to style the structure of the client interface. Certain accessibility features such as color contrast and magnification will be implemented with the use of CSS.

JavaScript v5 and v6:

- JavaScript will be used to control the behavior and functionality of the assessments and items.

jQuery v2.1.4:

- jQuery will be used to abstract browser specific language as well as adding functionality. There are numerous libraries that use jQuery that are also used such as Bootstrap. Although a lot of Bootstrap use might be replaced by Polymer Material library.

Polymer v1.0:

- Used as the framework for the web component. It's rich feature sets aids in data binding and set structure for importing different elements.

RaphaelJS v2.1.2

- SVG library used to create graphical interactions. This is used over other libraries because it is lightweight, SVG's are DOM elements so manipulations and event handling is simple and can be used with jQuery or vanilla JS. SVG's also have wide browser support with certain browsers providing hardware acceleration support.

RaphaelJS v0.5.1

- RaphaelJS library used to extend Raphael's featureset to include graphs used in certain item types.

Python v3.4:

- Used as the backend programming language. Will aid in processing and retrieving all the data needed in the system. Current design of the system will mean the backend is more of a RESTful service.

Pyramids v1.5:

- Python web framework used to run a RESTful service.

SQLAlchemy v1.0:

- Database ORM used to abstract SQL and maintain ability to switch to any database without much, if any, recoding.

Postgresql v9.3:

- Open source database used to store all the data. Subsequent versions will have much stronger support of JSON including querying in JSON which will be of benefit.

2.1.5 Communications Interfaces

Communication between client and server will be implemented through HTTP and/or HTTPS protocols.

Communication between web components will be handled with Polymers. Communications to parent are handled through core-signals functions and communication to children will be handled through HTML web-element attributes.

2.1.6 Memory Constraints

Memory constraints are minimal and related to the users browser's requirements with a nominal 100-200MB on top of that for the ARI system page for most typical item types. Other item types might require more.

2.1.7 Operations

No set operations at this time.

2.1.8 Site Adaptation Requirements

No set site adaptations at this time.

2.2 Product Functions

User retrieval – User logs into system. Their profile is pulled along previous session data and sent the controller with an assigned assessment manifest, this is known as the CDP.

Controller creation – The ARI controller is created in the client's browser and handed the CDP.

Controller initiation – The ARI controller loads the CDP, loads the CDP, and queries CDP for out initial item to load.

Controller item load – Creates a web component of the item and loads the appropriate web component html and hands the item web component the appropriate data.

Item Web Component – The item web component starts loading, sends a "loading" message to the Controller, processes it's data, displays it's payload onto it's web component, and when ready sends a "ready" message to the Controller.

Client Response – The client interacts with the Item Web Component, when they are finished they either signal controller or item itself of intent.

Item Wrap Up – When item receives signal of completion, sends message to Controller that it is "done". Sends data package back to the controller to be sent back to appropriate server.

Item Clean Up – Controller removes the web component.

Controller Next – Item determines if there is another question if so loads it, if not displays end of assessment.

2.3 User Characteristics

Users will primary be students from K-12 background. Secondary consideration should be put administrators, teachers, item developers, and parents.

Consideration needs to be places on student GUI best practice along with accessibility features the item needs to be accommodate based on user profile.

2.4 Constraints

Front-end constraints will be browser requirements along with need to use modern web browsers.

Polymer is still in development and as such they are still developing their feature set. In comparison to more mature frameworks like Angular there are a few constraints.

One such constraint is data binding of HTML is not supported. This presents a problem with dynamic repetitive elements that need to be grouped like radio buttons. In the current SBAC XML format for NL types in particular SimpleChoice elements radio buttons are web elements. By creating radio buttons as elements they lose grouping with their set and is only bound to it's own template. One way to address this is to use native radio html tags with attributes that can then be used to create the desired element.

Another constraint is Polymers loading feature. Some items take a considerable time to load; others won't load until the controller, by direction of the user, has forced a reload. This issue should be addressed in subsequent Polymer and ARI updates.

2.5 Assumptions and Dependencies

Assumptions:

- Client rendering items.
- JSON data payloads.
- Modern browser support.
- Web Component support.
- Polymer framework support.

Dependencies

- User profiles
- Item Database
- Client has access to open internet

2.6 Apportioning of Requirements.

No apportioning or requirements are set at this time.

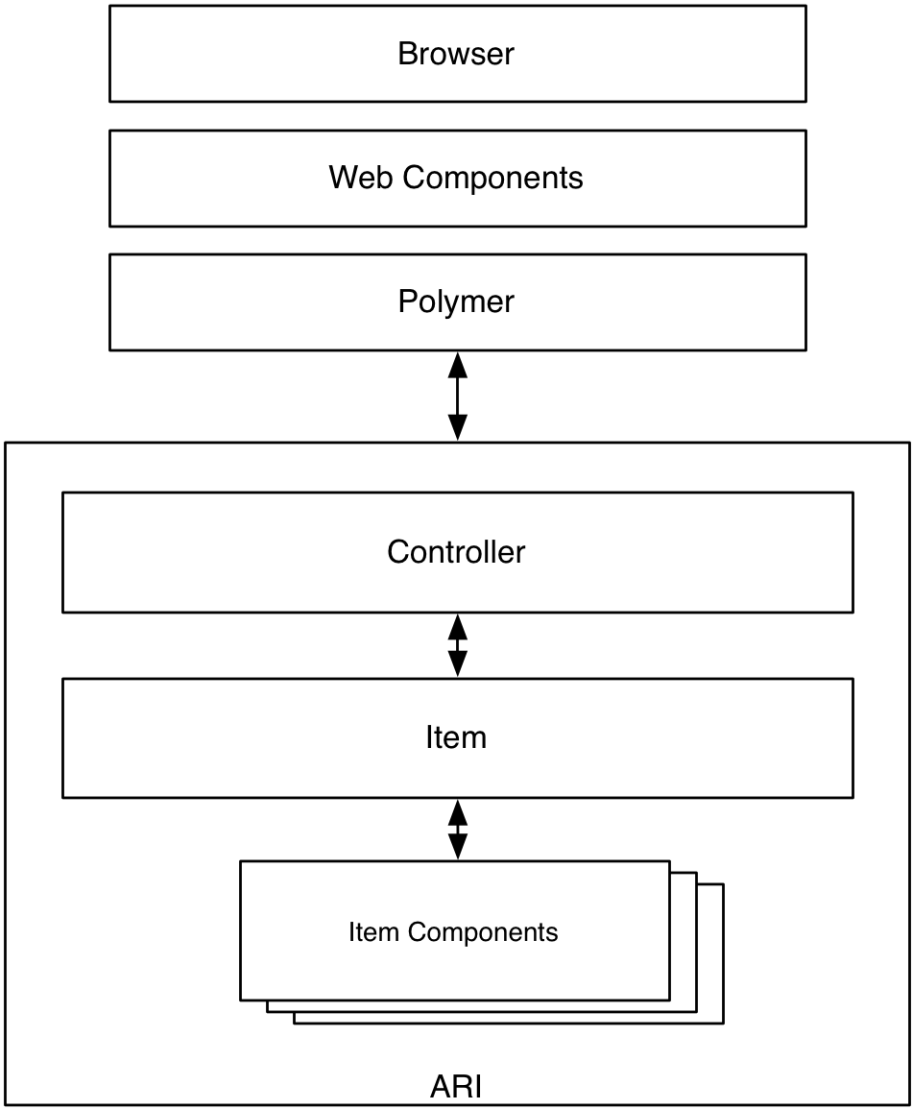
3. Specific Requirements

3.1 External Interfaces

Currently the system is self-contained. At the moment there are no set external interfaces.

3.2 Functions

3.2.1 Overview



3.2.1 Controller

3.2.1.1 Controller will request all data payloads from the server through AJAX calls.

3.2.1.2 Once Controller is loaded it will instantiate the Item Web Component from data in the CDP.

3.2.2 Item

3.2.1.1 When item is loading it will fire off message “loading”

3.2.1.2 Item needs to display it’s content on it’s web component. It will be styled by it’s own CSS and needs to take accessibility into account

3.2.1.1.1 Accessibility data will be handed to the item when instantiated along with other user and item related data.

3.2.1.1.2 Item accessibility needs to be in accordance with Smarter Balanced guidelines for accessibility.

3.2.1.3 When user signals intent, either the item or controller will process that intent. Item needs to have the capability to process the intent by sending message “done” to controller.

3.2.3 Item Component

3.2.1.1 Item component will instantiate with JSON data handed to it as a web component attribute “data”.

3.2.4 Controller

3.2.1.1 The Controller will cleanup the item element.

3.2.1.2 The Controller will load the next item if applicable.

3.2.1.3 User can signal save or finished to the controller.

3.2.2 Functions and Data

Currently all of SBAC’s item sets are in XML format. A requirement for ARI is for item sets to use the JSON format. To accommodate a more efficient transition and conversion ARI handles a straight XML to JSON conversion of existing items.

There were some modifications that needed to be done after the conversion but they were minimal. The modifications are as follows:

- Passages has an array inside the passage object called “items”
- Due to constraints in how items are named the following XML elements have been renamed:
  - itemBody - item-body
  - prompt - prompt-body
  - simpleChoice - simple-choice

3.2.2.1 Functions

Web Component State Functions

The Polymer framework has a loading order of the following:

- `created` callback
- local DOM initialized
- `ready` callback
- `factoryImpl` callback
- `attached` callback

The created callback is always called before ready.
The ready callback is always called before attached.
The ready callback is called on any local DOM children before it's called on the host element.

Core-signals for loading needs to happen on the created function.
Item instantiation needs to occur on the ready function.
When an item is ready or needs to manipulate the DOM, those functions need to reside on the attached function.

Controller Functions

itemStatus(e):

- Input:
  - event object. Any related data should be attached to the object.
- Function that registers and handles core-signals.

previousQuestion(data):

- Input:
  - data: Object.
- Function will load and instantiate the previous question if exists or allowed.
- Saves data object of current item if needed.

nextQuestion():

- Input:
  - data: Object.
- Function will load and instantiate the next question if exists or allowed. Also saves the item data to the server.
- Saves data object of current item if needed.

processAssessmentManifest(json):

- Input:
  - json – Object -Assessment Manifest
- Function will parse though manifest and extract relevant data.

loadQuestion(questionNumber):

- Input:
  - questionNumber – Integer
- Loads item with the corresponding question number

itemSave(data)

- Input:
  - Data – object
- Saves the data object to the server for the current item

Item Functions

In order to maintain extensibility, the only real requirement for an item is to do a core-signal when user has selected an answer. The controller will do the rest. An example of such is:

this.fire('itemStatus', {data: {'status':'done', 'user': answerObject}});

3.2.2.2 Data

Currently the following json variables are required in the ARI system.

Assessment JSON – please see 4.1 for schema example

Name	Type	Desc
name	String	Name of the assessment
items	Array	Array of Items
number	Integer	Order number of this item in assessment
url	String	URL of the web component for the item
element	String	Web Component should have tag names as such "item-element-1448". In that example 1448 is the item id. Element in his case is the item ID.

Passage JSON – please see 4.2 for schema example

Name	Type	Desc
itemrelease	Object	Object of the passage
version	Float	Version number of this passage
id	Integer	Passage ID
Items	Array	Array of the items
order	Integer	Order this specific item will be displayed on passage
json	String	URL of the JSON file for this item
content	Object	Object of the passage content
language	String	Language this specific content is in

stem	String	Text that will appear for the passage
apipAccessibility	Object	Accessibility objects

Item JSON – please see 4.3 for schema example

Name	Type	Desc
itemrelease	Object	Object of the item
version	Float	Version number of this item
id	Integer	Item ID
associatedpassage	Integer	Passage ID of it's parent
attriblist	Object	Object of Attributes
MachineRubric	Object	Object of rubric. Can contain JS that will be sent back to server to score data of this item.
content	Object	Object of the item content
language	String	Language this specific content is in
stem	String	Text that will appear for the passage
apipAccessibility	Object	Accessibility objects

3.3 Performance Requirements

The prototype has no performance requirements yet.

3.4 Logical Database Requirements

The controller will load the CPD.
Item web components will load Item data in the same format as the XML but in JSON data.
Item web components will hand a user data payload to the controller and the controller will send that payload to the server, with a possible response.

3.5 Design Constraints

Controller and items need to be web accessible and compliant to modern web standards.

3.5.1 Standards Compliance

Items need to be cross browser compatible along with modern mobile devices and tablets.
Standards that need to be meet are:

HTML5 - http://www.w3.org/TR/html5/
CSS - http://www.w3.org/Style/CSS/specs.en.html
JavaScript - http://www.ecma-international.org/publications/standards/Ecma-262.htm

3.6 Software System Attributes

3.6.1 Reliability

Item needs to work in a wide range of browsers and not interfere with existing frameworks. Item needs to display the same across all platforms.

3.6.2 Availability

The system needs to be available 24/7

3.6.3 Security

The prototype is security agnostic at this point. Once fully developed and in conjunction with external systems, security guidelines need to be established.

3.6.4 Maintainability

Items need to be self-contained in its web component. It is also required that the item maintain it's interface without interfering with the controller.

3.6.5 Portability

Specify attributes of software that relate to the ease of porting the software to other host machines and/or operating systems. This may include:

- *Percentage of components with host-dependent code*
- *Percentage of code that is host dependent*
- *Use of a proven portable language*
- *Use of a particular compiler or language subset*

- *Use of a particular operating system*

Once the relevant characteristics are selected, a subsection should be written for each, explaining the rationale for including this characteristic and how it will be tested and measured. A chart like this might be used to identify the key characteristics (rating them High or Medium), then identifying which are preferred when trading off design or implementation decisions (with the ID of the preferred one indicated in the chart to the right). The chart below is optional (it can be confusing) and is for demonstrating tradeoff analysis between different non-functional requirements. H/M/L is the relative priority of that non-functional requirement.

The front-end needs to maintain portability across many browsers and platforms. The back-end, for the purpose of the prototype, is an abstraction in it's current form so that will need to still be defined in later stages

ID	Characteristic	H/M/L	1	2	3	4	5	6	7	8	9	10	11	12
1	Correctness													
2	Efficiency													
3	Flexibility													
4	Integrity/Security													
5	Interoperability													
6	Maintainability													
7	Portability													
8	Reliability													
9	Reusability													
10	Testability													
11	Usability													

12	Availability													

Definitions of the quality characteristics not defined in the paragraphs above follow.

- *Correctness - extent to which program satisfies specifications, fulfills user's mission objectives*
- *Efficiency - amount of computing resources and code required to perform function*
- *Flexibility - effort needed to modify operational program*
- *Interoperability - effort needed to couple one system with another*
- *Reliability - extent to which program performs with required precision*
- *Reusability - extent to which it can be reused in another application*
- *Testability - effort needed to test to ensure performs as intended*
- *Usability - effort required to learn, operate, prepare input, and interpret output*

3.7 Organizing the Specific Requirements

3.7.1 System Mode

Developer mode: verbose outputs to console.
User mode: no console outputs.

View – Intended to view instructions.
Gather – Intended to view student input
Evaluate – Intended to evaluate student.

3.7.2 User Class

Developer – Access to all functionality and data.
Student – User primarily in evaluate mode.
Teacher - User in View or Gather mode.
Administrator – User in View or Gather mode.
Parent – User primarily in Gather mode.

3.7.3 Objects

Students – student information is passed to the controller and subsequently to the item.
Items – Item data exists in an item web component as a JSON file.
Assessments – Assessment data is sent to controller which will contain the items needed for the student.

3.7.4 Feature

- Loading of student data.
- Parsing of student accessibility data.
- Loading items on the client browser
- Passing student response back to server for possible grading

3.7.5 Stimulus

Item content and student generated data.

3. 7.6 Response

Item event handling – Item events will be captured and sent to controller.
Controller event handling – Controller will capture response of the event and send it to the server.

3.7.7 Functional Hierarchy

The Controller is in charge of all operations outside of item display and capture. It will instantiate items, save its state, send data to items and server.

3.8 Additional Comments

None.

4. Supporting Information

4.1 Sample assessment JSON file – http://52.11.155.96/static/doc/assessment.json
4.2 Sample passage JSON file - http://52.11.155.96/static/doc/passage.json
4.3 Sample item JSON file - http://52.11.155.96/static/doc/item.json