ARI Technical Working Group Report

Meetings held 12-13 Nov 2014
Report 30 January 2015

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).

The ARI Technical Working Group was convened by UCLA CRESST and met at the CRESST offices on November 12-13, 2014. This is a report on the design and recommendations that resulted from those meetings.

Attendees

Meeting participants included the following:

- UCLA CRESST: Alan Koenig, John Lee, Li Cai, Markus Iseli, Frank Masur
- Measured Progress: James Bowen, Chad Rider, Joseph St. George
- AIR: David Lopez de Quintana
- CoreSpring: Evan Eustace
- US Ed: Steve Midgley
- Dan Rehak
- Smarter Balanced: Brandt Redd

Intellectual Property Policy

Participants agreed that we are all seeking an open-licensed solution. Generally speaking, this means that specifications should be released under a Creative Commons License, that code should be released under a permissive open source license, and that patent encumbrances should be avoided. While we didn't sign any IP releases as might be required by IMS Global processes, we agreed in principle to these ideals and there was mutual understanding that if/when the specification is turned over to a standards organization, IP releases may be required of participants.

TWG Recommendations

The Technical Working Group (TWG) began by reviewing design goals – essentially the purpose of the ARI activity. All agreed that the final product should be evaluated against these design goals to

ensure that the new approach represents a valuable improvement over existing standards and technology.

Design Goals

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 runtime 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 Global
    - Unencumbered by intellectual property or patent restrictions.

Technical Requirements

Following the design goals, the team reviewed technical requirements from Smarter Balanced. For the time being, these requirements represent a proxy for overall industry requirements, especially for high-stakes accessible testing. While Smarter Balanced includes summative, interim, and formative assessments; current work is dominated by the summative effort. Therefore, additional input from non-summative efforts such as CoreSpring may be important.

Technical requirements should be additional detail below the design goals. In this case, the TWG reviewed existing item types and accessibility features required by Smarter Balanced. These correspond to the "Supports advanced item types" and "Supports accessibility features" design goals respectively. However, the existing feature lists are a baseline. The design goals extend to more advanced item types and more accessibility features.

Existing Smarter Balanced Item Types

- Evidence-Based Selected Response (two-part multiple choice)
- Multiple Choice
- Multi-Select

- Match Interaction (using a grid rendition)
- Short Answer
- Extended Response
- Equation Response
- Table Interaction
- Graphic Response

Existing Smarter Balanced Embedded Accessibility Features

- Color Contrast
- Pop-Up Glossaries (English and Other Languages plus Audio)
- Translation
- Text-To-Speech
- American Sign Language (for listening passages)
- Print-On-Demand Passages
- Braille Emboss-On-Demand Passages
- Closed Captioning (for listening passages)
- Highlighter
- Strike-Out
- Streamline Layout

In addition to these features, the Smarter Balanced assessment delivery systems follow WCAG and ARIA standards in order to maintain compatibility with third-party assistive devices.

Design Decisions

The core deliverable of the TWG meeting was a set of design decisions. This was seeded by a review of a proof-of-concept by Brandt Redd accompanied by lists of "Questionable Design Decisions" and "Identified Design Issues." The original lists with some notational commentary are included in Appendix A.

The following are design decisions that were made by the TWG after robust discussion that included a lot of insight and contributions from team members.

- All rendering (conversion from transmission form to display form) should be performed on the browser side. No server-side rendering is necessary even to incorporate accessibility features.
- The items should be packaged using IMS Content Packaging
- The ARI format should describe items in the packaged state, mechanisms to obtain consistent client independent rendering, standard item runtime interactions and behaviors and common data models available to an item.
- The entry point to an assessment item is an "item manifest" in JSON format. The manifest should have well-known elements that are the starting point of item use, interaction and rendering.
    - Defined entry points are: render and capture response, score, and render response for review.
- A well-known file, other than the manifest, should store the item metadata.
    - Metadata is defined as data that is needed for management of and selection of items but does not have any impact on how the item itself is rendered, how interaction occurs, or how the item is scored. While metadata should not be necessary for scoring an item, metadata *may* indicate how the item score contributes to an overall test score.

- The manifest lists 1-n JavaScript files that handle rendering and interaction with items.
    - All items listed in the manifest should be included in the item package.
    - The contents of the manifest are made available to the JavaScript at runtime.
    - The JavaScript files may be in common with other assessment items. In certain cases (such as multiple choice) the only content unique to an item may be in the manifest.
    - The content may be differentiated by asset type, e.g., rendering, scoring
- The manifest lists 0-n asset/resource files that are part of the item.
    - The content may be differentiated by asset type, e.g., media, data
- The packaging application (application that creates a package of assessment items) is responsible for managing dependencies.
    - Presumably, the packaging application draws items from a pool and packages them up for distribution. Identifying which assets are required for each item and which common assets are used by multiple items is a function of the packaging application. The data and algorithms needed for this function are an implementation detail outside the scope of the ARI format.
    - All JavaScript and other common assets used by an item must be listed in its manifest.
- The item incorporates the execution and interpretation logic in its JavaScript assets.
    - This includes logic for rendering and interaction for any of the defined entry points as well as incorporating accessibility features.
- The ARI format makes no special browser or user agent requirements.
    - Securing the test environment (e.g. limiting websites the user can visit, preventing unauthorized applications, etc.) is the responsibility of the runtime application and the browser. It is beyond the scope of the ARI format.
- The design should use Web Components as its primary structure – within the limits of available Polyfills.
    - HTML Generation and Entry Point should be designed the way Web Components work
- The manifest can provide interaction configuration data to the item in JSON format.
    - For example the prompts and selectable responses for simple multiple-choice items.
- Model data (in a Model-View context) is pre-loaded before the item is initialized.
- The API design should use loose function binding and self-descriptive data types.

Data Objects Records

In parallel with the design decisions, the team identified certain data records. Mechanisms must be in place to retrieve and transmit certain records between server, browser, and assessment item Other objects exist to manifest API elements to the JavaScript code. The scope of the ARI format is limited to records that need to be accessible to the assessment item JavaScript code and how to exchange those records with the host system.

The following records were identified:

- API Binding Browser Controller – This is a JavaScript object that manages the connection between the assessment item and its host.
- Item Specific Data – This includes the manifest as well as other declarative data that's unique to a particular assessment item instance. The working group was undecided as to whether the manifest and other item-specific data should be combined into one JSON file or whether the manifest and item data should be in separate files.
- Item Rubric Data – Data that is item-specific but is used exclusively for scoring. For high-stakes testing, rubric data is not sent to the browser. In formative testing, rubric data is often used to give immediate feedback.

- Student Response Data – This is the student's response to the assessment item. For a simple multiple-choice it may be a single letter or digit. For an essay it could be multiple paragraphs including markup. For a simulation or graphic item it may be proprietary structured data. Regardless, the student response is a JSON-format document that is opaque to the host system (other standard formats, e.g., HTML, MathML) may be included as part of the opaque content within the JSON.
- Student Score – This is the score earned by the student response. For machine-scored items, it is generated by applying the JavaScript scoring rubric (part of the item identified in the manifest) to the Student Response Data. At least part of the score should be in a format that is well-known and specified by the ARI format. Certain item types will generate multiple scores.
- Student Experience Data – This is optional record records data about the student experience that does not contribute to the student score and, therefore, is not included in the Student Response Data. Certain components of the Student Experience Data, such as time on task, may be well-defined but the record should be extensible to item-proprietary data. Best practices may include insights from the Experience API (xAPI) and the IMS Caliper standard.
- Runtime Context / Student Profile (The name for this record needs to be refined) – Contains data about the context in the item is rendered. This may include accessibility options such as color contrast, translation, glossaries, print-on-demand, braille, and so forth. It may include student information such as a name so that the item can be personalized. And it contains other contextual information. The record should have a well-defined extensibility model to allow for custom options without namespace collisions. Values are likely to be in separate, but related specifications. For example, Smarter Balanced would likely publish an accessibility feature specification specific to its assessment framework.

Item Modes, Status, and Mode Transitions

Considerable discussion was devoted to the modes of an item, the status of an item and how an item transitions between modes. A provocative question was whether the item tells the environment that it's time to move to the next item or whether the environment makes that transition. This is typically manifest by where the "next item" button is located – as part of the item or as part of runtime frame. Ultimately, the team concluded that it's the runtime environment that owns the "next button" or other advancement mechanism. However, the item needs to manifest its status in order to indicate whether it's appropriate to advance.

These modes and status values are exchanged through API calls between the item and the runtime environment.

Modes

The operating environment sets the item's mode. The following item modes were identified and the team agreed that the mode list should be extensible:

- Gather – The item allows user interaction to gather a response from the student.
- View – The item displays a student response but does not allow interaction. If no response is supplied, the item displays its initial state.
- Evaluate – The item displays a student response along with an evaluation of the student's performance.
- Done – The environment is moving to a different item so this item should save its state.

Additional Work: "Done" is an event, not a mode. The workgroup placed it here because the "SetMode" operation is a convenient way to express the "done" event. But it's not parallel with the rest of the construction. The "evaluate" mode might require two forms. In one form, the student

continues to interact with the item, as with the "gather" mode but with feedback being offered continuously. In the other form, the item would be static, as with the "view" mode, except that feedback about student performance is offered in-context.

Status

An item manifests its status to the runtime environment. Status only changes if the item is in "Gather" mode since no other mode allows interaction that would provoke a change in status. The team identified the following status values:

- Empty – The item has not gathered any response from the student (though there may have been some other form of interaction).
- Some Response Received – The item has gathered some response data from the student.
- Complete – The student has provided the minimum amount of input. For example, in a composite item, the student has provided a response to all required parts. Note that an essay item might indicate "complete" even if only one word or letter has been input.
- Finished – The user, thorough interaction with the item, has indicated that they are finished with the item and the environment should advance to the next item.

Deferred Questions

During the discussion, the TWG identified the following design questions and deliberately deferred them for future proposals or discussions.

- What should the item "manifest" be called? Since IMS Content Packaging includes a manifest, having a manifest in the item may be confusing. Choosing a different name for this may be preferable.
- How can security be preserved when loading item JavaScript? How can we tell that no malicious code has been inserted into an item implementation?
- How to handle long-running items? These are items that preserve their state between sessions in which student interaction may span multiple days and sessions.
- How to handle event collection (for xAPI/Caliper info) across multiple sessions with the same item? (E.g. the user answers a question, moves on, and then revisits that question, possibly changing the answer.)
- Must the Student Response Data be entirely opaque? Can there be well-defined data in it? Should the Student Response Data specify a schema so that other applications can determine whether they can interpret the data?
- These vocabulary terms, among others, need to be clarified: Item, Interaction, Resource, Asset, Rendering
- How much "setup" of the item occurs on the server? For example, combining and minifying JavaScript files.
- Must all assets be listed in the item manifest or only those which might be shared among items?

Next Steps

The team agreed that a prototype implementation should precede the writing of a specification. CRESST will assign a developer to build a prototype and that developer should have access to the TWG members to discuss ideas and gain clarity on issues.

The TWG should evaluate the prototype and suggest refinements until it meets the requirements outlined in this report and in the grant and project plan. Once those criteria are met, then a specification can be written based on the prototype implementation.

Appendix A:
Proof of Concept "Questionable Design Decisions" and "Identified Design Issues"

The following questionable design decisions were identified before the meeting. Commentary or conclusions by the team are in parentheses:

- The Proof of Concept introduced a new packaging format. Why not use IMS Content Packaging? (Should use either IMS Content Packaging or Node Package Manager (NPM))
- Metadata is embedded using YAML format. (Should not introduce another format – stick to XML or JSON.)
- No Pre-Selected JavaScript Library such as JQuery. (Provoked a long discussion about library compatibilities, scope, versioning and interaction. Results of that discussion are in the Design Decisions.)
- Server-Side JavaScript for Item Rendering (Seems to be unnecessary, all rendering can be done on the browser side. Server-side JavaScript for scoring remains valuable.)
- No security model for server-side JavaScript to prevent malicious behavior (Should be addressable with an adequate JavaScript sandbox.)

The following design issues were identified before the meeting. To the extent that they were addressed, the results appear in the Design Decisions

- Dependency management (on JavaScript libraries)
- Versioning of common libraries (such as JQuery)
- Browser compatibility and future-proofing
- Multiple items in a single HTML page
    - Library interactions
    - Reference resolution
- CSS Links (must appear in the HTML page header, not inline. However CSS literals can be inlined.)
- Leveraging existing JavaScript solution to problems common between assessment and other web applications.
- Facilitation of student interaction tracking systems such as IMS Caliper and Experience API (xAPI)