Simile Tools Workshop Summary
MacKenzie Smith, MIT Libraries

Intro

On June 10th and 11th, 2010 a group of Simile Exhibit users, software developers and architects met in Washington D.C. to discuss the current limitations of the tool and how it could be redesigned to meet the changing needs of its user community. The group reviewed a sample of current uses of Exhibit to understand both its value and its limitations, devised a new architecture for the product, and discussed how to insure broad participation in the development and adoption of the new tool. A process was discussed for the reimplementation, but the group stopped short of identifying exactly when and how such a project would begin. For information on the current Exhibit tool see http://simile-widgets.org/exhibit/. The revised tool is referred to below as Exhibit 2.0.

Key New Requirements

To briefly summarize the key decisions on the scope of Exhibit 2.0, it will

- Be scalable up to 1 million triples and 20,000 facet values,
- Provide a server-side option for publishing large collections, and a simple way to move from small collections stored in the browser to large collections stored on the server (ideally with the same UI design process),
- Support incremental updates of views in the UI,
- Provide an API for result set pagination and/or summarization,
- Make it easier to add new views, widgets and facets to the UI,
- Improve embed-ability of Exhibits into external applications and Web pages, and support links to particular Exhibit states,
- Provide support for limited data and UI editing in the Exhibit UI via plug-ins or widgets,
- Provide data export features to enable large-scale data editing in external applications (e.g. Gridworks),
- For complex UI layout editing, provide extension hooks and reference implementations as examples

Requirement Details

1. Scalability

Exhibit was designed as a Web browser widget that stores all of its data in memory, so it can only handle a small number of items (around 500 to 1,000 depending on the complexity of the items). Since many Exhibit users have much larger collections that they want to put in Exhibit, it will be necessary to redesign the tool to support both small and large collections. The scale of collection that we require for Exhibit 2.0 is characterized below:

- Number of items <=100K
- Number of triples <=1m
- Number of bytes/triple <=1K
- Number of facets <=20
- Number of facet values <=2K

OUT OF SCOPE

- Scaling to a larger number of items in a specific view (e.g. map, timeline)
- Number of item types or horizontal scaling (e.g. to enable link sliding)

2. Extensibility (i.e. new views/facets/widgets)

Exhibit was designed to be a data navigation and visualization framework into which new "views" of data (e.g. plot, graph, map, timeline, or a particular facet) could be added. This process is working but with extreme difficulty, partly because it's not well explained how to do it, and partly because the code to support this is complex. We agreed to the following priorities for Exhibit 2.0:

- *Document* the current API for writing new view/facets/widgets with clear examples
- Make it *easier* to connect new views/facets/widgets to the framework
- Add an API to support result set pagination and/or summarization
- Add support for adding/subtracting items from a view/facet/widget *incrementally*

3. Persistence

As a Web browser widget, Exhibit allows users to work with the data in memory and change the local state, for example selecting facets to limit the record set they're viewing, or choosing a particular view of the data. Users want to be able to point to the finished product – the state of the Exhibit that they reached via a set of operations on the data. We agreed that Exhibit 2.0 should:

- Improve permalink (bookmarking) capability for end users to specific views with state; primarily a documentation issue

4. Editing

An ongoing need of Exhibit users, including both data owners who publish their data with Exhibit and end users who interact with data in Exhibit, is to edit the data directly in the tool rather than in a separate program that can open the Exhibit data file stored on the server. We recognize two distinct needs:

- Editing "master" data (usually by the data owner) in a way that keeps the changes permanently.
- Editing a copy of the data for local manipulation (e.g. a student working with a dataset as part of a class assignment) and which does not require persistence to be handled by Exhibit itself.

We concluded that

- Simple edits (e.g. fixing a typo in a single facet value) should be handled by an Exhibit widget for editing-in-place with persistence. The MIT CSAIL Dido project is a prototype for this (see http://projects.csail.mit.edu/exhibit/Dido/).

OUT OF SCOPE

- More extensive editing (e.g. fixing all instances of a facet value across a large dataset) should be done outside of Exhibit, for example using Freebase's Gridworks (see http://code.google.com/p/freebase-gridworks/).
- We did not resolve the question of where or how persistence should be handled for the second case (the data copy), nor how the two cases might interoperate.
- Support for end user *layout editing*, e.g. views or facets, with persistence, is desirable but out-of-scope for now.
- For large dataset editing, Exhibit could support single RESTful writes back to the server but that is not a priority for Exhibit 2.0.

5. Modularity (aka "embed-ability")

Many Exhibit users need to embed Exhibit in a software tool chain and/or a separate system (e.g. VIVO, Sakai, WordPress). This is possible but difficult to do now, so Exhibit 2.0 should provide easier ways to call Exhibit modules and address individual components (e.g. views) in separate web pages.

OUT OF SCOPE
An "Exhibit Builder" tool for end users, or for external systems to build custom Exhibits on-the-fly. NOTE: the issue is the utility of defining a formal representation of Exhibits that could be stored and compiled in other systems (e.g. a way to specify what facets, what properties to display, basic layout, etc.). If that existed, external systems, e.g. VIVO system, could do things like create Exhibits on-the-fly for every distinct *data-type*. We acknowledge that this would be very useful but is beyond the scope of what we want to accomplish for Exhibit 2.0.

6. Data indexability/exportability

Exhibit was designed to be dynamic, importing the data and javascript code to render it from a source outside the Web page. This makes it difficult for external service providers like Google to harvest the data for indexing etc. without extra effort by the Exhibit author. Furthermore, because its current state is only known to the local Web browser, it's difficult to point another user at an Exhibit in a particular state (i.e. offer Permalinks that take you to exactly what the user is experiencing in their current session). Both of these should be supported by Exhibit 2.0. Specifically,

- Exhibit 2.0 should have the capability of exporting the original dataset (in pre-JSON encoding, where relevant) via a URL, such that tool chains can extract data for external processing before or after rendering, and
- Offer a permalink for the current state of the Exhibit (see #3 above).
- Offer a documented way of encoding the data in the HTML that supports indexing

In addition, the documentation should explain the existing ways to support external indexing by search engines (e.g. including the data in a non-displaying HTML tag in the rendered Exhibit).

7. Usability

Usability covers a range of behavior of the software. Here we identify some general goals for Exhibit 2.0 usability in three dimensions:

- Latency will be reasonable, e.g. under 10 seconds to render and include a progress bar
- Exhibit authors will still be able to cut and paste existing HTML to create new Exhibits, and do not need to know programming languages to create simple customizations.
- There will be an easy process to scale from small Exhibits to large ones for the same dataset as it grows over time. The transition from Web-browser only (i.e. the current Exhibit) to the server-side version of Exhibit should be simple and seamless.

8. Exhibit Dependencies

Exhibit relies on an externally-hosted javascript library (http://api.simile-widgets.org/) and two external services, i.e. Babel, Painter (http://service.simile-widgets.org/) and these are not 100% reliable since they're hosted by the MIT Libraries as a convenience to the Exhibit community. It is possible to host copies of these files and services on the Exhibit user's local infrastructure, and we encourage that in cases where Exhibit reliability is important. While we cannot do away with these external dependencies completely, Exhibit 2.0 will

- Provide simple instructions for hosting the API and services locally (i.e. not at MIT),

- Provide a mechanism to distribute load on the externally hosted services, and failover,
- In the new server-side architecture, an API and specifications will be defined for creating a local server rather than using an assumed community-provided server (like MIT's). There will be no centrally-run Exhibit server component.
- Eliminate service dependencies where possible – e.g. Painter may no longer be required due to new Google APIs

REMAINING QUESTION: we need to define how many server environments we will support beyond the reference implementation. We cannot support all possible environments (e.g. operating systems, databases, Web servers, etc.) and this should be settled early on.

Architecture

Exhibit 2.0 will maintain the design goals of the current Exhibit while becoming more scalable and modular to meet emerging requirements (see above). To accomplish this, a client/server architecture will be adopted, following the model implemented by the Backstage prototype (i.e. data manipulation managed on the server, and the UI rendering on the client, using AJAX). The prototype code is available at http://simile.mit.edu/repository/backstage/trunk/ and some discussion by its developer is here: http://www.mail-archive.com/general@simile.mit.edu/msg03135.html

A few specific recommendations include:

- Assume state is kept on the client, in the URL or the Web page.
- Server functionality should include returning data and facet values, up to some limit.
- The query language should use the JSON pattern (like the Backstage prototype) translated on the server side to SPARQL/SQL/etc.
- The server design should permit various back end storage systems: in-memory, file system, database, triplestore, etc.
- Consider Solr for the indexing engine.
- Evaluate the utility of an API abstraction (e.g. Sesame SAILs).
- Evaluate whether the original source database (or a subset of it) could be fed to Exhibit via a simple flat file mechanism.
- Refactor Exhibit client code based on increased maturity of libraries like JQuery.

Community Engagement

Another topic of the workshop was the extent to which Exhibit 2.0 should be conducted as a fully open source software development effort or a centrally-managed development project that releases open source software once a stable, but not yet final, version is complete. There was consensus in the group that we should adopt the latter approach and produce an Exhibit 2.0 beta version with a dedicated development effort in a short time frame, then encourage early adopters to participate in developing the final release. Additionally,

- The software management infrastructure will continue to be Subversion (e.g. at Google Code), and the dispersion (e.g. GitHub) model might be a good fit for this community (not a final recommendation, but a request to consider).
- Remove external service dependencies on MIT-run services (i.e. Babel and Painter).
- Static javascript files should be centrally hosted on at least 3 distinct servers (including, potentially, cloud-based application engines), and adopters should be strongly encouraged to host it locally whenever possible.
- We will maintain the current infrastructure for community participation, e.g. the simile-widgets.org website and the simile-widgets Google Group. We will make every effort to sunset

older documentation sites and wikis, for example, archiving simile.mit.edu. We will consider migrating the documentation wiki back to Google.
- Create an advisory team for the effort, including interested experts from MIT (e.g. the Haystack research group), and companies working in this area (e.g. Zepheira, Metaweb and Talis).
- Also create an Early Adopter Team of software projects with an interest in Exhibit 2.0 (e.g. the VIVO project in the US, the Ensemble project in the UK, the MIT Libraries, Zepheira and Talis).

Process

Finally, the group discussed to process required to develop Exhibit 2.0. We acknowledged that the effort would be a rewrite of the current Exhibit, since the new architecture requires asynchronous communication with the data source and the current Exhibit is entirely synchronous, and it should use jQuery and other post-Exhibit 1.0 advances in Web application development. However we assume that Exhibit 2.0 must be backwards-compatible given the large current adopter base. A few recommendations were:

- The project time frame should be held to one year
- The development process should be open to the community for visibility, but not for active participation (see community engagement discussion above)
- The project must include a documentation effort as a major component, both to clean up older documentation that is still relevant, and to produce good documentation for new aspects of the tool. Both user and developer documentation must be addressed. This work should all be done together, when Exhibit 2.0 is released (i.e. don't fix current doc now and plan to rewrite it again in a year)
- The new APIs will be specified after the stable-but-not-final release of Exhibit 2.0 is made, to avoid over-specifying what is not yet tested. Bring in early adopters to test the new APIs and to integrate and extend them before the final release.

For more information please contact the convener, MacKenzie Smith from the MIT Libraries, via email at 'kenzie at mit.edu'.