Games Programs Play: Obstacles to Data Reuse

Chris Scaffidi
Institute for Software Research Intl.
School of Computer Science
Carnegie Mellon University
cscaffid+isri@cs.cmu.edu

Mary Shaw
Sloan Software Industry Center &
School of Computer Science
Carnegie Mellon University
mary.shaw@cs.cmu.edu

Brad Myers
Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
bam@cs.cmu.edu

ABSTRACT
Information workers often reuse data by taking it from an existing representation, recombining it to create new data, and storing the new data in another representation. The sources and destinations include databases, spreadsheets, web sites, text documents, and emails. Recombination activities are similarly diverse and include copy/pasting, concatenating, visual reformatting, arithmetic/calculating, and so forth. Yet many obstacles impede such reuse. In this paper, we summarize the problems that users face as well as some strategies for overcoming these problems.

Author Keywords – data, reuse, software, interoperability

ACM Classification Keywords
H.3.5. Online Information Services: Data sharing.

OBSTACLES USERS HAVE ENCOUNTERED

We have recently conducted three studies that characterize numerous obstacles impeding effective data reuse by end users, professional programmers, and everyone in between.

First, preliminary analysis of our contextual inquiry of three administrative assistants and five managers at Carnegie Mellon University reveals that much of their work involves manually copying and pasting data among web pages, spreadsheets, and emails. Their work is highly repetitive and ripe for end-user programming—except that they lack suitable tools.

Second, our finished survey of 831 computer-savvy *Information Week* readers asks what software they use, followed by the open-response question, "In what ways has this software 'gotten in the way' of doing work in the past year?" [5] Of the 527 people who list problems in response, 25% mention obstacles related to data reuse, especially data incompatibility. (By comparison, only 15% mention bugs, glitches, or other software reliability problems.)

Third, preliminary analysis of telephone interviews with six people involved in creating Hurricane Katrina "person-locator" sites suggests that even technically capable people struggle to reuse data. As these sites redundantly proliferated in the weeks after Katrina, three of our respondents helped merge sites into a single whole. Though handcrafted scripts processed over 500,000 records, numerous problems forced volunteers to type in another 100,000 manually.

In general, users may perform the following six steps when reusing data, and obstacles abound at each step. (Below, "CI" refers to our contextual inquiry, "IW" refers to our *Information Week* survey, and "HK" refers to our interviews related to Hurricane Katrina person-locator sites.)

Step 1: Find data sources

Reusing data first requires finding it, which can prove tedious. One IW respondent has expressed unhappiness with his organization's "very fragmented data management environment," while another has complained, "Separate files in separate formats and folders causes [sic] confusion and need for good organizational skills." In fact, our CI reveals that even if users only need a single piece of data to populate a spreadsheet or web form, they may struggle to find the datum using software and instead fall back on manual methods. For example, administrative assistants and managers fill out many expense reports that require a project code for each expense, but looking up codes is slow, usually involving scrolling through long lists onscreen, sending emails, or phoning peers. To overcome this obstacle, workers collaborate to maintain a "cheat sheet" (in Excel) which they each print and keep on a stand next to their monitors.

Step 2: Access data sources

Once workers locate data, accessing it may be hard. For instance, some HK site creators have refused to let aggregators access backend databases, so aggregators have resorted to using "screen scrapers." As a second example, in order to analyze data in the accounting database, CI managers must first export the data to a file on their desktop computer; this export function is only accessible from browsers running on Windows XP. Our CI also reveals other access issues, some requiring intervention by technical staff.

Step 3: Vet and repair data quality

Ensuring data quality is a problem in any dataset, but even more so when humans generate the data. To deal with this,

HK aggregators have promulgated an XML standard for structuring data. This standard includes fields that help data users evaluate data's reliability so they know what data might need filtering or repair; for example, fields include the record's creation date and the contact information of the record's creator. However, data quality problems are not limited to hurricane-devastated areas but can be endemic to office environments. As one IW respondent has reported, poor data quality "leaves a lot of database cleaning to be done before the information can be used for intended purposes."

Step 4: Cope with incompatibility

After finding, accessing, and vetting data sources, users seek to combine data. Unfortunately, syntactic (meaning-free) incompatibility may interfere with combining data, often due to incompatibility in data layout or encoding. For example, HK data aggregation involves converting data from a rows-and-columns database representation into a hierarchical XML format, with its nested angle-bracket tags and rules for encoding many characters.

Other incompatibility occurs at a subtle, semantic level, where two apparently compatible data representations in fact have incompatible meanings. For example, end users of HK sites often have used the wrong web forms to enter data (e.g.: acting as if data about lost pets is semantically equivalent to data about lost humans, and then using the "missing persons" form to enter data about missing pets).

This problem's dual occurs when different systems interpret the same data in different ways. Formatting incompatibility is a particular case: Many IW respondents complain that different applications render data in different ways. For instance, Firefox and Internet Explorer render HTML differently, and WordPerfect and Microsoft Word render rich text differently. One IW user dislikes needing to "spend to [sic] much time making something look pretty," a sentiment shared by some CI spreadsheet users.

After coping with data incompatibility, users can combine the data by copy/pasting, concatenating, visual reformatting, arithmetic/calculating, and so forth.

Step 5: Store new data

Software limitations hamper storing new data due to performance, capacity, or access problems. For example, one HK interviewee notes the lack of scalability in Access for storing large data sets; similarly, several IW respondents have noted, "Excel can't handle much data."

Step 6: Publish new data

Users' ultimate goal is to publish new data, but helping others to find it can prove challenging. For many HK site creators, the main challenge has been getting the media to report sites' existence to the world. Data exposure is also a problem in offices; one IW reader has complained about the "limited ability for automated report distribution," while several CI users must print out documents and distribute them manually due to insufficient workflow automation.

TOOLS FOR FINDING / ACCESSING / REPAIRING DATA

End users often find data using commercial search tools whose main function is to draw together numerous scattered data sources into one index. Such tools are valuable because users still store and publish data via largely application-specific, decentralized, ad hoc mechanisms such as copying files to a web server or sending emails.

Researchers have recently focused on providing tools to help end users access and repair data. For example, tools exist that allow users to automate retrieval and manipulation of web page data [1]; Java-savvy users can even use such tools to populate spreadsheets [2]. Ensuring data quality remains difficult, but researchers have made progress in the web service [3] and spreadsheet [4] domains.

Integrating tools like these with search systems, and extending them to other domains such as databases and emails, may raise new usability and reliability challenges that deserve further exploration. However, our present research agenda centers on data incompatibility, which is the main subject of the following sections.

STRATEGIES FOR COPING WITH INCOMPATIBILITY

Shaw lists strategies to deal with *packaging* incompatibility between executable software components A and B [6]:

1. Replace A's representation with B's representation.
2. Publish an abstraction of A's representation.
3. Transform A on the fly to B's representation.
4. Negotiate to A and B's lowest common denominator.
5. Make B multilingual.
6. Provide B with import/export.
7. Transform A and B to intermediate representation C.
8. Attach a wrapper to A.
9. Maintain parallel consistent versions of A and B.

Some of these have natural analogues for coping with *data* incompatibility. For example, a user can combine data from spreadsheet A and web page B by running COM-based scripts on both documents (strategy 2), or by exporting the spreadsheet to HTML and referencing it in the web page with a <FRAME> tag (strategy 6).

Although existing tools lack support for some strategies, many strategies do prove useful in certain contexts. For example, database federation exemplifies several of these strategies [7]. In particular, federated systems must negotiate common protocols on the fly (strategy 4).

Whereas federation deals with database incompatibility, systems like Citrine deal with office application incompatibility [8]. Citrine transforms clipboard data from one representation to a standardized intermediate representation (strategy 7) so that users can copy/paste structured data among applications.

In terms of software architecture, many of these strategies can most easily be implemented by interposing a mediator component between A and B. For example, Microsoft COM DLLs act as mediators that expose an abstraction of web pages for scripting (strategy 2). Mediators are known

by various names: "converter" (if used in strategies 3 and 7), "broker" (if used in strategy 4), "translator" (if used in strategy 5), and "façade" (if used in strategy 8).

Unfortunately, there are inherent challenges to mediator-based implementation, as discussed below. Moreover, all nine strategies' practical utility is limited, as no existing tool supports the full range of users' data representations in database tables, groups of spreadsheet cells, web pages, documents, and emails.

TACTICS FOR SUCCESSFUL MEDIATION

Effective mediation ideally requires the mediator to recognize the *details* of the source and destination's layout, encoding, and semantics. For example, Excel can export spreadsheets to a certain XML schema, but this serves no purpose if the user needs to import the data into a system that uses a slightly different XML schema than Excel does. This sensitivity to a representation's details leads to two challenges for making mediator-based strategies successful.

First, in order to be cost-effective, any mediator implemented by a professional should ideally recognize *multiple* detailed representations. (Professionals are typically too expensive to have them create one mediator per detailed representation.) There are several tactics for achieving this:

1. Let the end user customize mediators' behavior.
2. Let the end user (rather than a professional) create mediators in the first place.
3. Let the end user share customized / created mediators with other users (permitting further customization).
4. Let mediators automatically customize their own behavior when faced with new data representations.

Second, mediators are often not robust to evolution of representations, thus provoking manual reprogramming to prevent subtle semantic bugs from jeopardizing data quality. Researchers have worked toward automatic detection of evolution in web service semantics [3]; generalizing this tactic to other representations would be extremely valuable.

Tactics like these are essential to making mediator-based strategies successful, but some mediators are more amenable than others to these tactics.

FUTURE WORK: ENHANCEMENTS FOR CITRINE

In the future, we hope to apply several of the tactics and strategies listed above to produce an end user programming environment that supports a variety of data sources and a variety of ways to combine data from those sources. As a start, we will enhance Citrine, a mediator for copy/pasting structured data [8].

Currently, when end users paste data into a new web form that they have never before encountered, they each must train Citrine how to map the data into the form. Essentially, this equates to customizing the mediator's behavior (tactic 1 in the list above). We will evaluate five enhancements that may reduce users' effort:

1. We will enable users to save a capsule containing a form's data so they can reload the capsule and skip the copy/paste step entirely when reusing data in that form.
2. We will automatically save a capsule each time a user completes a web form. Thus, the next time that the user completes similar forms, we may be able to use the user's entries in some form fields to predict what values should go into other fields. This would eliminate manual reloading of capsules.
3. When a user maps data to a form, we will record the structure of this mapping in a central repository so that if other users face a similar situation, Citrine can offer a reasonable default mapping.
4. We will use machine learning to identify the most commonly occurring mappings so that Citrine can perform them automatically.
5. We will explore how visual cues on the page can help Citrine maintain high quality even if the data sources and destinations evolve in structure or semantics.

These enhancements should reduce the effort required to reuse data in web forms and reveal data patterns that may be of benefit as we tackle data reuse in other contexts.

ACKNOWLEDGMENTS

We thank Andrew Ko for his helpful questions and suggestions. This work was funded in part by the EUSES Consortium via NSF (ITR-0325273), by NSF under Grant CCF-0438929, by the Sloan Software Industry Center at Carnegie Mellon, and by the High Dependability Computing Program from NASA Ames cooperative agreement NCC-2-1298. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

1. Elbaum, S., et al. Helping End-Users "Engineer" Dependable Web Applications. *ISSRE'05*, 22-31.
2. Kandogan, E., et al. A1: End-User Programming for Web-based System Administration. *UIST'05*, 211-220.
3. Raz, O., Koopman, P., and Shaw, M. Semantic Anomaly Detection in Online Data Sources. *ICSE'02*, 302-312.
4. Rothermel, G., et al. A Methodology for Testing Spreadsheets. *TOSEM'01*, 110-147.
5. Scaffidi, C., Ko, A., Shaw, M., and Myers, B. Identifying Categories of End Users Based on the Abstractions That They Create, Tech Rpt CMU-ISRI-05-110/CMU-HCII-05-101, Carnegie Mellon University, Pittsburgh PA, 2005.
6. Shaw, M. Architectural Issues in Software Reuse: It's not Just the Functionality, It's the Packaging. *SSR'95*, 1-3.
7. Sheth, A., and Larson, J. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. CSUR 22, 3 (1990), 183-236.
8. Stylos, J., Myers, B., and Faulring, A. Citrine: Providing Intelligent Copy-and-Paste. *UIST'04*, 185-188.