What's in a name? Exploring the connections between abstraction and appropriation

M. Cameron Jones,
Michael B. Twidale
Graduate School of Library and Information Science
University of Illinois at Urbana-Champaign
mjones2@uiuc.edu, twidale@uiuc.edu

Abstract. In this position paper, we discuss the role of abstraction in designing for appropriation. We examine the ways in which varying the level of abstraction of tools affects the ability of users to appropriate them. We close with some words about the difficulties of evaluating the appropriability of systems and how they might be addressed in an experimental framework.

1 Introduction

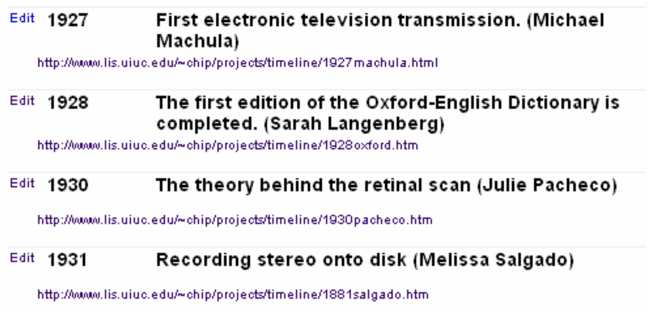
A focus of computer systems design research has been building systems which are capable of being appropriated by users. In order to build systems that explicitly support appropriation, the factors which affect appropriability must first be identified. One such factor is the manner in which software tools are described. This is especially true in component-based software development (CBSD) environments which are designed to enable end users to combine components in order meet their particular needs. These systems often are extremely flexible and powerful. They combine all three aspects of tailoring as described by Mørch (1997); namely customization, integration and extension. Users can configure customizable options for each component, arrange and rearrange components in any number of combinations to create new compound functionalities, and even generate new components through coding or sharing templates.

Based on preliminary observations of a community learning toolkit, it would appear that tailorable tools, when abstractly described to encompass maximal flexibility and customizability, are less appropriable than when provided as a narrower, particular instance. This is a significant problem for researchers and developers of tailorable technologies, as it contradicts the practice of offering the most flexibility and the greatest customizability to the user for the sake of allowing them to fashion whatever they would like.

2 The Story of the Timeline Tool: Anti-Affording Appropriation

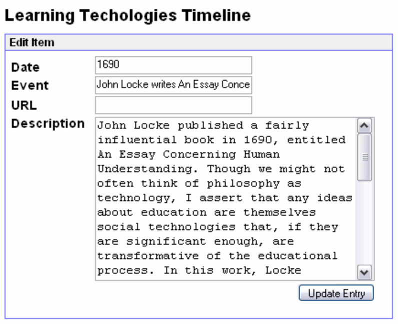
The ILABS system, short for Community Inquiry Labs, is a framework and a set of tools, called bricks, for supporting online communities of inquiry (Bishop, et. al, 2004). The aim of the system is to enable participants to put together a customized environment that will support learning and knowledge sharing for a particular community of users. Examples of the diversity of communities using ILABS include university courses, a Puerto Rican community library project, an African-American women's health network, and a multi-disciplinary research initiative.

One of the first bricks developed for the ILABS project was a timeline tool. The original timeline brick was built around the needs of a professor for one of his classes. The Learning Technologies Timeline existed as a static HTML page, maintained by the professor, which students researched and contributed items to. The initial timeline brick was built specifically to address data of this form, helping the professor update and manage the timeline. The resulting tool had input fields for a date, event, URL and description; the data was sorted in ascending, chronological order.



Edit	1927	First electronic television transmission. (Michael Machula) http://www.lis.uiuc.edu/~chip/projects/timeline/1927machula.html
Edit	1928	The first edition of the Oxford-English Dictionary is completed. (Sarah Langenberg) http://www.lis.uiuc.edu/~chip/projects/timeline/1928oxford.htm
Edit	1930	The theory behind the retinal scan (Julie Pacheco) http://www.lis.uiuc.edu/~chip/projects/timeline/1930pacheco.htm
Edit	1931	Recording stereo onto disk (Melissa Salgado) http://www.lis.uiuc.edu/~chip/projects/timeline/1881salgado.htm

Figure 1



Learning Techologies Timeline

Edit Item

Date: 1690

Event: John Locke writes An Essay Conce

URL:

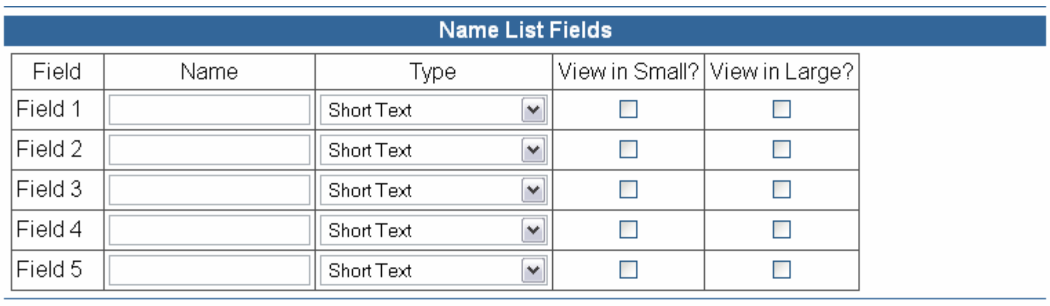
Description: John Locke published a fairly influential book in 1690, entitled An Essay Concerning Human Understanding. Though we might not often think of philosophy as technology, I assert that any ideas about education are themselves social technologies that, if they are significant enough, are transformative of the educational process. In this work, Locke

Figure 2

Figure 1 shows a view of the Timeline tool, almost identical to the original HTML based Learning Technologies Timeline. Figure 2 shows the interface for adding and editing items to the Timeline. Note the specific names of the fields.

Although developed for one professor, the timeline brick was used by other groups. This straight forward tool was easy for users to comprehend – the structure and name of the tool accurately reflected its purpose. The concept of a "Timeline" was familiar and did not need much documentation to explain its purpose and use. Shortly after introducing the tool, we noticed that many people were using the timeline brick for other purposes such as organizing upcoming events, email addresses, daily schedules, and other tasks that involved creating an ordered list of items. We consider these unintended uses to be appropriations of the software – allowing the users to accomplish their goals by creative use of the technology available to them. That is, by looking at examples of timelines created with it, not only were they able to develop their own timelines but also imagine using it for other purposes.

In light of these new uses, we redesigned the tool to be a more generalized key-sorted data table. The new tool was more customizable, allowing users to specify the number, name and data type of the fields, and change the sorting behavior. In order to communicate the breadth of this tool's functionality we renamed it the "Sorted List" to emphasize that it was now a tool from which any number of types of lists could be created, of which timelines were just one example.



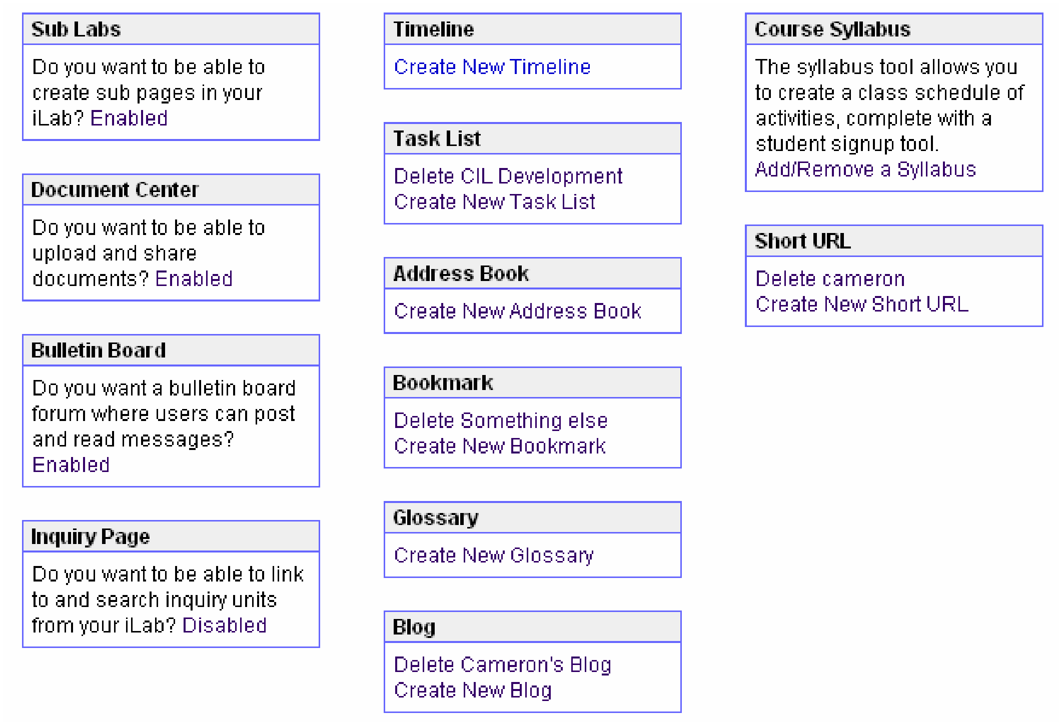
Name List Fields

Field	Name	Type	View in Small?	View in Large?
Field 1		Short Text	☐	☐
Field 2		Short Text	☐	☐
Field 3		Short Text	☐	☐
Field 4		Short Text	☐	☐
Field 5		Short Text	☐	☐

Figure 3: A screenshot of the new Sorted List creation tool, showing the numerous configuration options. Users can define a name, data-type and display options for each field. On subsequent steps of the configuration wizard, users can define sorting behaviours. This very powerful, abstract tool actually led to less appropriation.

Unfortunately we noticed in informal observation and feedback from users that most users did not know what the new, improved Sorted List tool was or why they would want to use it. The Sorted List was described as a tool which enabled users "to create different kinds of sorted lists as content for your iLab". Even when the description was modified to include examples (the addition of the phrase "like bookmarks, contact lists, blogs, etc." to the previous description), users still had troubles understanding what the tool was for. This was very frustrating to us as developers of this tool. We had initially developed a tool that happened to lead to informal spontaneous appropriation. Noticing and valuing that phenomenon, we had put substantial effort into refactoring the design, and created an abstracted functionality far more powerful, adaptable and tailorable than the original, but had ended up with something that was used less and hence adapted and tailored less. Why was this? We suspect that in the process of adding power by abstraction, what was lost was an understandable model which users could grasp; an existing context where they could observe what the different configuration options were for.

To address this paradox we incorporated specific configurations of the list tool as starting points from which users could adapt to their particular needs. The configurations currently supported includes: the timeline, task-list, address book, bookmarks, glossary and blog. These configurations are not merely default settings for the various configuration options; they each represent a distinct conceptual purpose or use-instance of the tool. Not only can users get by with making fewer reconfigurations, but the cognitive overhead of adapting a particular list is less than that of instantiating the more abstract data-type of Sorted List.


Sub Labs
Do you want to be able to create sub pages in your iLab? Enabled

Document Center
Do you want to be able to upload and share documents? Enabled

Bulletin Board
Do you want a bulletin board forum where users can post and read messages? Enabled

Inquiry Page
Do you want to be able to link to and search inquiry units from your iLab? Disabled

Timeline
Create New Timeline

Task List
Delete CIL Development
Create New Task List

Address Book
Create New Address Book

Bookmark
Delete Something else
Create New Bookmark

Glossary
Create New Glossary

Blog
Delete Cameron's Blog
Create New Blog

Course Syllabus
The syllabus tool allows you to create a class schedule of activities, complete with a student signup tool.
Add/Remove a Syllabus

Short URL
Delete cameron
Create New Short URL

Figure 4: A screenshot of the revised iLabs system showing six different configurations of the Sorted List tool including the timeline, presented in the middle column.

From a design perspective this redesign seems a retrograde step – a hack-like inelegant duplication of work, unnecessarily multiplying the number of options that the user has to decide to choose from. From the perspective of a computer scientist no new power has been added and the elegance of the previous Sorted List tool has been corrupted. And yet we have some slight evidence from subsequent use that this change is at least better than our first redesign. The bestpractices advocated in object-oriented design lead programmers to create powerful abstractions to maximize modularity, extensibility and reusability while minimizing redundancy (Alfonseca, 1990). These principals do not seem to enable end users to appropriate and reconfigure in the way that programmers are meant to search through and select from abstract classes and superclasses in Smalltalk. Put this way, it does not sound so surprising, but that is our concern–these design precepts are rarely articulated in sufficient detail for a clear critique to be applied to their suitability for end user appropriation.

3 Using versus Programming

Object-oriented languages like Java and Smalltalk offer programmers vast libraries of classes and class hierarchies from which to select, extend and use in their applications. These libraries are similar to the sets of tools and components made available to users of CBSD systems in that they contain a large amount of pre-defined functionality from which a user must make informed selections. The process by which a programmer chooses a class or module, however, is very different from that of a user. Where a user is driven primarily by picking a component which will do what he or she wants, a programmer must consider other aspects such as memory usage, speed, and flexibility in addition to functionality.

The naming of components in systems designed for end users needs to reflect the users' frame of reference. The classes and modules in programming libraries are named and described by and for programmers. However, adopting the manner of description employed by computer programmers to the description of end user tools can be highly problematic. As evidence, we present some observations from the DATA TO KNOWLEDGE (D2K) toolkit developed at the National Center for Supercomputing Applications (NCSA). The D2K toolkit is a data-mining framework, allowing users to construct data flows from modules. D2K modules have inputs and output; they are assembled in a graphical drag-and-drop interface, and connected by data pipes thus specifying the flow of data through the system. Data is then loaded into the front end of the dataflow and a sequence of operations is executed on the data as it passes from one module to the next. The toolkit serves as both an application for end-users to mine data using the modules provided and a platform for developers to build new algorithms by leveraging an existing code base.

Problems arise when the toolkit is used by non-programmers (who are usually experts in the data they wish to model but have no formal programming experience). The graphical interface is designed to make it possible for these users to apply advanced data-mining software easily. The problem is that the interface is just a graphical representation of the underlying programmers' application programming interface (API). The API, intended for programmers, contains language and descriptions which are targeted at programmers; designed to help them make informed decisions about which modules to use and extend in their application development. Users cannot easily navigate the collection of modules and find ones they might want because they are organized, named and described as they would make sense to a programmer. For example, the machine-learning classifier modules are in different hierarchies for the different authoring organizations (e.g. WEKA vs. NCSA classifiers). This mismatch is indicative of what we feel is a fundamental difference, that abstraction in computer programming languages is different from the kind of abstraction that is helpful to users.

Similarly, the naming and description of the Sorted List brick in ILABS changed with its development. From the perspective of the developer, there was a logical progression; as the tool got more powerful it got more abstract – both conceptually and in name. However, from the perspective of the user, as the tool became more abstract, it became harder to envisage *any* of its different intended uses.

4 Problems and future work

What needs to follow is a more formal and rigorous study of this phenomenon. Much of our observations have been informally gathered through interactions with users and usage log analysis. However, formal comparisons of the appropriability of the various instances of the timeline tool could help in understanding the phenomenon better. We are currently planning user studies designed to measure how significant the abstraction and naming can be in appropriation. A major hurdle is determining how to measure appropriation and appropriability. Even determining an operational definition of appropriation, its different forms and the features that afford it needs more work. We have begun enumerating some of the aspects of technology and software which we feel afford appropriability and might lead to evaluation metrics; this list is by no means complete or necessarily accurate, but has been included as a starting point for further discussion and perhaps to serve as a guidelines for evaluation.

- **At-Handness**: At-hand tools are those which are both physically and cognitively available to the user. At-handness is more than physical availability, because tools may contain features or functionality the user does not know about and thus cannot appropriate. At the same time, there might be a problem with ubiquitous things 'being hidden in plain sight'. Ciborra (1996) describes a similar concept asserting that appropriation happens when a user becomes "intimately familiar with an innovation", ultimately allowing that user to be able to call upon the technology to support day-to-day activities.

- **Granularity**: Clay offers high levels of precision and control in sculpture, but takes a lot of knowledge and skill to mold well. LEGO blocks are a more coarse-grained design resource which can be assembled with lesser skill to produce rough sculptures which approximate the smooth curves attainable with clay. LEGO construction can also be codified precisely such that the exact same sculpture can be reproduced whereas no two clay sculptures can be precisely the same.
- **Playfulness**: The degree to which a tool supports and encourages users to 'play-around', testing variant configurations and learning about how the tool functions. More playful systems could lead to greater discovery of features or generate more ideas about how to use the system in novel ways. This idea is connected more with the notion of serendipitous appropriation as opposed to what we've primarily been discussing – goal-oriented appropriation. We consider serendipitous appropriation to be the uses which arise out of spontaneous creativity – a moment when a user realizes that the tool they have could be used to do something else. This is unlike the goal-oriented appropriation, where a user finds a technology which can help him or her satisfy a need or aid in attaining a specific, defined goal.
- **Sharability**: The degree to which the tool supports sharing customizations and modifications. Tools that have higher sharability would allow users to share appropriations and learn from each other (e.g. Nardi and Miller, 1991).
- **Simplicity**: Tools with complex interfaces might be too difficult to integrate (in the words of Mørch, 1997). Simple things might just be easier for users to understand and learn - thus increasing the at-handness. Also, simple tools might have some atomicity in their functionality, allowing them to be appropriated into ad hoc workflows more easily.

We believe that a consideration of the features that enable appropriation can lead us to the specification of requirements for technologies that can explicitly support appropriation activities by users.

5 References

Alfonseca, M. (1990). Object-Oriented Programming, Tutorial. Conference Proceedings on APL 90: For the Future. May 1990.

Bishop, A. P., Bruce, B. C., Lundsford, K. J., Jones, M. C., Nazarova, M., Linderman, D., Won, M., Heidorn, P. B., Ramprakash, R., Brock, A. (2004) Supporting Community Inquiry with Digital Resources. Journal of Digital Information. 5(3).

Ciborra, C. U. (1996). Introduction: What does Groupware Mean for the Organizations Hosting it? In Ciborra, C. U. (ed.) Groupware and Teamwork: Invisible Aid or Technical Hindrance. Wiley Series in Information Systems. 1-19.

Mørch, A. (1997). Three Levels of End-user Tailoring: Customization, Integration, and Extension. In M. Kyng and L. Mathiassen, editors, Computers and Design in Context. The MIT Press, Cambridge MA.

Nardi, B.A. & Miller, J.R. (1991). Twinkling lights and nested loops: Distributed problem solving and spreadsheet development. International Journal of Man-Machine Studies, 34(2), 161-184