Chapter 15
The Future of Transclusion

Robert M. Akscyn

15.1 Introduction

Transclusion, a term coined by Ted Nelson [5, 6, 9], is a powerful concept, that like hypertext, offers users considerable benefits. In Ted's words:

> In this system, portions of content are brought in from various sources (local and remote). The content portions thus brought in may remain visibly connected to their origins. This is an important case of transclusion, which we define as the same content knowably in more than one place (For instance, being able to see a quotation or excerpt and its original context in another document.) [8]

Yet, like the term hypertext, the basic concept of transclusion is today so commonly encountered (implicitly, mainly via the web) that it has become as inconspicuous as electricity, or the inner workings of internal combustion engines. As advantageous as inconspicuousness can be (after all, who maintains their own car these days?), this "out-of-sight, out-of-mind" quality of transclusion has an unfortunate flip-side: not thinking much about it leads to not fully appreciating its future potential as much as we might, if only we thought about it more.

The purpose of this note is to make a rapid fly-by of the concept of transclusion, in order to expound opportunities for further capitalizing on its potential.

R.M. Akscyn (✉)
Knowledge Systems, Las Vegas NV 89135, USA
e-mail: rakscyn@gmail.com

D.R. Dechow, D.C. Struppa (eds.), *Intertwingled*, History of Computing,
DOI 10.1007/978-3-319-16925-5_15

15.2 Background

My experience with the concept of transclusion comes mainly from the development of a series of hypertext systems over the past 36 years, beginning with a project in the Computer Science Department at Carnegie-Mellon University in 1978. Those systems were ZOG (CMU 1978–1985) [10], KMS (Knowledge Systems 1981–2006) [1], and Expeditee (University of Waikato 2006–2014) [2].

All three of these systems have transclusion at their core. This is done primarily by chunking the components of an artifact, such as a document or program, into screen-sized portions (called *frames*). Each frame then has a unique name so that it can be referenced by linking.

As a result, incorporating existing material into yet another artifact is merely a matter of creating a link to the existing frame representing the "top of the world" for that material. In general, the interpretation of such links is left to the devices of the link source side so that different treatments of the one-and-the-same transcluded material can be specified in multiple, independent-of-the-source ways.

15.3 Omitting Needless Features

Central to the design philosophy for these systems is simplicity. The primary approach to 'keeping it simple' has been to find data model, user interface and programming concepts that are not necessary for the sake of knowledge work—and then, to the extent practical, get rid of them. Thus, ironically, the ZOG/KMS/Expeditee systems are defined as much by what they're not, as by what they are.

15.3.1 Data Model Omissions

The following concepts are omitted in the Expeditee conceptual data model:

Desktop, Dialogue Boxes, Files, Folders, Layers, Menu bars

In short, the functionality of all container types is provided by a single concept, the concept of a frame. A frame is a named, screen-sized graphics space, capable of containing any spatial arrangement of text, graphics, images, and sound objects (including overlapping) that users wish.

15.3.2 User Interface Omissions

The following concepts are omitted in the Expeditee user interface:

- Dialog Boxes
- Editors (there is no separate editing mode)

- Focus (focus always follow cursor position)
- Icons
- Menus (no menu bars, pull-down/pop-up menus)
- Naming (names are system-generated)
- Saving (saving is navigation-triggered)
- Scrolling
- Selection operation (operand scope is defaulted to whole items)
- Text cursor (no explicit focus representation)
- User interface (widget-ish objects are merely contents like any other)

Some central aspects of the Expeditee user interface are:

- All methods for existing objects (move, copy, delete, scale) are identical across all object types.
- All methods are single point-and-click (there are no conventional widgets).
- Execution time for methods is one second on average.
- System response time is one-twentieth of a second on average.
- Rate of interaction is typically hundreds of atomic methods per hour.

15.3.3 Programming Language Omissions

Some of the programming concepts omitted in the Expeditee scripting language are:

- Call-by-value
- Classes (and objects, as well as over a hundred other concepts of object-oriented programming)
- Declarations (variables are self-typing)
- Functions (there are only procedures)
- Global variables (all variables are local to their parent procedure)
- Goto (all transfers of control are calls-with-return)
- Keywords (no reserved terms)
- Programs (there is no program level, execution can begin at any block)
- Punctuation (commas, semi-colons, parentheses, braces)
- Symbol-controlled calling (all calls are done by links)
- User-defined data types (all data types—including images—are system-provided)

The only programming concepts in the ZOG/KMS/Expeditee trajectory are statements, variables, and "blocks" (as in "block- structured" languages), with some blocks being procedures with parameters. Since scripts—which are also represented in frames—can be associated with any object, virtually all programmatic functionality occurs via stateless widgets, whereby all content types (points, lines, polygons, text, images) are "clickable" for the sake of invoking functionality.

15.3.4 Results of Omissions Trajectory

In short, the over-40 years of ZOG/KMS/Expeditee trajectory has been an embarrassingly slow "de-learning" of many of the concepts computer science has used to advance the state of the art over the past five decades. Thus, from the outside, these systems appear as though little or nothing has been developed, which is precisely what was intended so that, akin to a car windshield, users are rarely aware they are using a system. Instead, they are able to keep their attention on the task at hand.

On the other hand, there are some benefits that stem from using direct manipulation with spatial hypermedia to represent knowledge however one wishes: using scripts to post-process the user's representation, for example; and exporting contents to whatever formats downstream applications require.

To offer just one example, software development can be done in which the code is written in a personal programming dialect (which gets translated when exported) within a lattice of frames, each of which may contain whatever graphics, images, sounds, and colors the programmer wishes to collage along with the code (e.g., for comments by self or colleagues, links to related materials). Frames can transclude other frames that serve as layers overlaying widget collections as user interfaces, even on a frame-by-frame basis, so individual users can customize interaction functionality however suits them best. Note that coding inside a holistic environment that one can use for virtually all other knowledge development purposes is highly-unconventional relative to existing development environments and practices.

Ironically, much of what is actually done in such a hypertext-transclusion world—to expedite the production of knowledge artifacts—can't be done at all in most systems of today.

15.4 The Value of Transclusion

As alluded to above, transclusion is used in many systems, including virtually all hypertext systems [3, 4]. This section addresses a number of issues about transclusion such as examples, benefits, costs, trade offs, and opportunities for expanding the functionality of the concept beyond mere inclusion of original material. Existing examples of transclusion include:

- Images, scripts, and style sheets for web pages (invariably these are not in the base page of the webpage)
- Include files (and/or classes) for programs
- Layers in graphics programs
- Components stored in and dynamically accessible from a database (many web pages are database-based)

15.4.1 Benefits and Costs of Transclusion

The principal benefits of transclusion are reuse of existing material (versus re-inventing the wheel), and currency (access to the latest, greatest version of material). The principal costs of transclusion include finding desired/appropriate transcludeable material (an instance of "the research problem"), as well as effecting the transclusion, which might involve subsetting existing material, or transforming the transcluded material. To the extent that such subsetting and transforming involves sophisticated knowledge (because the interface mechanisms are complex) the less attractive using transclusion will be.

Benefits of transclusion can also be viewed through a cost-reduction lens, in the sense that transcluding existing material obviates having to create a facsimile from scratch. This principle is central, since, as in many things in life, if the cost of doing something is high, it often isn't done for lack of available time and money resources. In the case of software, the costs of re-developing components that are already stress-tested by time can be extreme (assuming we wish the same quality). Thus transclusion offers the benefits of creating artifacts that otherwise wouldn't be done.

Naturally there is a trade-off between benefits and costs constantly in play—since for material that is small (especially material that is static like prose in published papers)—retyping prose to serve as a quotation will often be far more efficient than trancluding the text. Conversely, re-capitulating the LaTeX markup for a mathematics equation seen in a publication is sufficiently onerous that one would dearly love to transclude the original markup (even if that requires some work), if only such representations were generally available!

Thus simply transcluding the markup text shown in Fig. 15.1 from the author's original to directly achieve the same-as-original-author result shown in Fig. 15.2 would be a valuable expedient for reusing previous results, just as is the reuse of well-worn code.

```
\begin{equation}
   \label{IteratedFunctionSystem}
   R(h,q) =
       \begin{cases}
          0, & \text{if $h < q$} \\
          (h - 1) - \displaystyle\sum_{2 \leq p < q}
          \left ( R(\frac{h}{p}, p) + 1 \right ), & \text{otherwise}
      \end{cases}
\end{equation}
```

Fig. 15.1 LaTeX markup that produces Fig. 15.1

$$R(h,q) = \begin{cases} 0, & \text{if } h < q \\ (h-1) - \sum\limits_{2 \le p < q} \left(R(\frac{h}{p}, p) + 1 \right), & \text{otherwise} \end{cases}$$

Fig. 15.2 Visually complex mathematical equation generated from transcluded LaTeX markup

15.4.2 How Can Opportunities for Transclusion Be Expanded?

Opportunities for transclusion can be expanded via several dimensions:

- Expanding the sources of accessible material that is permissibly transcludable (i.e., does not violate copyright as well as generally accepted norms)
- Expanding the set of transformations that can be performed on the transcluded material
- Decreasing the size of the smallest grain size of transcludeable material
- Making transclusions recursive

15.4.3 Expanding the Source of Accessible Material

Naturally having more to choose from is better than having less. But such a capability is meaningless if the inconvenience of finding and collecting candidates, for repeated side-by-side comparison, is too high. As the saying goes, "In theory, theory is sufficient; in practice it isn't." Thus, the efficiency of every step of the transclusion ecosystem is critical to the degree transcludable material will be used.

Factors such as how well collections of material are organized will likely dramatically affect what actually happens, no matter the richness of the possibilities. Indeed, building on the well-known expression:

If it isn't written, it doesn't exist.

We might add:

If it's not in the library, it doesn't exist. (and)
If it's in the library, but is too hard to find, it doesn't exist.

But in addition to accessibility is the issue of just what can be done with transcludable material. Since the purpose of most knowledge artifacts is to contribute something original, simply reusing existing material, as is, may not provide value sufficient for the task at hand. Thus expanding the set of transformations that can be performed on the transcluded material beyond exact copying—is a way of expanding the design space for transclusion. This in turn admits for higher 'value' peaks that greatly increase the ROI of effort invested.

But in addition to accessibility is the issue of just what can be done with transcludable material. Since the purpose of most knowledge artifacts is to contribute something original, simply reusing existing material, as is, may not provide value sufficient for the task at hand. Thus expanding the set of transformations that can be performed on the transcluded material beyond exact copying—is a way of expanding the design space for transclusion. This in turn allows for higher 'value peaks' that greatly increase the ROI of effort invested.

15.4.4 What Are the Types of Transformation of Transclusions?

Some easily-envisioned transformations of material include:

- Subsetting (e.g., clipping out just a portion of a source image) results in more flexibility to get just the portion you need for the purpose you are striving to attain
- Scaling (up or down, through a wide-range of size increments)
- Re-colorizing (e.g., taking a greyscale image and displaying it using any base color)
- Rotation (e.g., any of the 360°)
- Associating a script to be activated when users interact with the displayed transclusion

One could, for example, create an artistic rendering of a Christmas tree, one with the decorative balls all being the same size and color. But the result might not be as engaging as a smattering of such ornaments with a variety of colors and sizes with all the different renditions transcluded and transformed from a single source image.

More sophisticated transformations might include:

- Translation of prose from one language to another
- Animation by 'tweening' between two different states (of the same scene)
- Speeding up the rate of animation of a GIF image
- Overlaying multiple animations on top of one another

In short, the ability to specify transformations as part of the transclusion reference, turns the source material into an abstraction—one whose generality spans all the combinations of the transformation dimensions. Thus, for example, size (say 50 settings), color (100 settings), and rotation (360 settings) by themselves would provide over 1.5 million possible variations of a single object.

15.4.5 Decreasing the Grain Size Directly Transcludable

Another dimension of expanding the design space is decreasing the grain size of what is directly trancludable—as a way of obviating the cost of purpose-built mechanisms that (in theory) can do the job—but at prohibitive cost. An underlying

motivation for this dimension is that components can only be so big, because at the limit you're simply regurgitating the original message. It will always be better to be able to use just the right-sized building block for the purpose at hand, e.g., only part of an image, text, or code.

Take the example of re-using the LaTeX code for a mathematical expression that was shown in Fig. 15.1 and 15.2. The reuse was significantly streamlined by the fact that the mark-up text in question is in an Expeditee frame of its own (from one of my math papers). As a result, trancluding it in this book chapter, only required copying its title (which in Expeditee auto-links to the frame where it resides) and plunking that linked item down in the relevant frame of the set I'm using to author this chapter. Two such transclusions were made—both to the one-and-only frame—one transclusion for showing the verbatim LaTeX source (Fig. 15.1) and the other to be interpreted by LaTeX to show the graphical end result (Fig. 15.2) it specifies.

By comparison, transcluding such an equation out of a whole-document text file would require significant work—to denote the beginning and ending character positions of the text within the file or, alternatively, cropping a portion of a screenshot. While conceivably the original author might have placed the equation within a file of its own, that's not common practice. Thus in addition to the fact that such upsteam-of-WYSIWIG files are generally not publically-accessible to begin with, the hassle of figuring out how to effect the transclusion would likely deter all but the most persevering of re-users. Indeed, many applications can't even transclude their own handiwork, even from artifacts created by the same user on the same system.

This particular example was chosen because it also illustrates the previous point about transformations, for it is the same Expeditee frame that is transcluded both times: first for the sake of showing the underlying LaTeX code, and then again for the sake of generating the end-view as seen in the original paper. The ability to have "the same content knowably in more than one place," but with a different presentation each time, is a good example of how separating presentation properties from content is a powerful mechanism for reuse (i.e., by avoiding embedded markup, as Ted has argued for decades [7]).

15.4.6 Making Transclusions Recursive

Finally, making the transclusions themselves be constellations of transcludable material further adds to the panoply of material that users can selectively reuse.

15.5 Conclusions

In summary, here are some take-home messages about transclusion:

- Since the fastest way to do something is to not have to do it at all, the ability to reuse an existing, well-crafted, extensively-tested existing artifact (or portion

thereof) is one way to avoid having to spend time and money developing that component.
- Transclusion offers a powerful mechanism for reusing material and thus expanding the volume of material that can be created over time. Such volume is critical to the creation of high value, as it better allows for authors to add value by giving us the best of a lot (which is a lot better than the best of a little).
- Potential reuse of transcludable material is greatly expanded if the mechanism for effecting the transcluding can also transform the original in desired ways, such as subsetting (cropping out portions of images as well as text), scaling, colorizing, and other desired transformations–including forming aggregates in which the same original is collaged in multiple, independent ways. Thus the ability to transform transcludable material within a large design space makes the original material inherently more reusable.
- Potential reuse of transcludable material is further fostered by the ability of transcluded material to contain transclusions of their own. In other words, by having the concept be fully recursive. This enables repositories of transcludable material to themselves be multi-level–enabling users to simply tap into whatever level of aggregation suits their creative purpose (e.g., from individual Christmas tree ornament, to the whole tree, on up to an entire landscape).
- But *potential* reuse is not the same as *practical* reuse. Thus, mechanisms for finding and transcluding material need to be efficient, as generally speaking whatever is inefficient is done much less often and therefore is not as valuable in practice as it sounds in theory.
- A recommendation: the more developers might collaborate to create common models of transclusion, as opposed to the endless one-upmanship now practiced—the more transclusion can come out of the closet, and be seen as a core capability that helps all knowledge workers lift their game across all their work, creating greater value for their organizations and clients.

References

1. Akscyn RM, McCracken DL, Yoder EA (1988) KMS: a distributed hypermedia system for managing knowledge in organization. Commun ACM 31(7):820–835
2. Akscyn RM (2009) The expeditee project: measuring the productivity of knowledge work. In: Proceedings of the New Zealand computer science research students conference. Auckland University, New Zealand
3. Bernstein M (2003) Collage, composites, construction. In: Proceedings of the fourteenth ACM conference on hypertext and hypermedia. ACM, p 122
4. Krottmaier H, Maurer H (2001) Transclusions in the 21st century. J Univ Comput Sci 7(12):1125–1136

5. Nelson TH (1965) A file structure for the complex, the changing and the indeterminate. In: Proceedings of the ACM 20th national conference, pp 84–100
6. Nelson TH (1995) The heart of connection: hypermedia unified transclusion. Commun ACM 38(8):31–33
7. Nelson TH (1997) Embedded markup considered harmful. In XML: principles, tools and techniques. World Wide Web J 2(4):129–134. Sebastopol: O'Reilly. http://www.xml.com/lpt/a/294
8. Nelson TH (2007) Toward a deep electronic literature: the generalization of documents and media. Project Xanadu. http://xanadu.com/XanaduSpace/xuGzn.htm. Accessed 16 Dec 2014
9. Nelson TH, Smith RA, Mallicoat M (2007) Back to the future: hypertext the way it used to be. In: Proceedings of the eighteenth ACM conference on hypertext & hypermedia, pp 227–228
10. Newell A, McCracken D, Robertson G, Akscyn R (1981) ZOG and the USS Carl Vinson. In: Computer science research review. Camegie-Mellon University, pp 95–118