Cognitive Issues in GUI design: constructing Website Maps

Oliver von Klopp Lemon
Dept of Computer Science
Trinity College Dublin
Dublin, Ireland
ojlemon@cs.tcd.ie

Ana von Klopp Lemon
Sun Microsystems
901 San Antonio Road
Palo Alto, CA 94303
avk@eng.sun.com

Abstract

This paper examines site maps (graphical representations of web sites, as employed by e.g. Visual Interdev and WebSphere Studio[1]), assessing their value in gaining an understanding of web sites. We consider the site mapping problem with reference to standard claims about benefits of graphical representations, but also argue that their use might be detrimental. We suggest that while some users find site maps useful, the cognitive support they provide is due to properties not normally thought to be among the main advantages of graphical representations.

Introduction

Using visualization to aid with various tasks is often seen as having some intrinsic merit as a means of facilitating human understanding and reasoning. However, the processes by which graphical representations can aid human cognition are not well understood, and research does not unequivocally support the view that they do, at least unless the visual language is highly constrained with respect to the domain it is to represent (see Scaife & Rogers 1996 for an excellent review).

Our objective is to review some of the claims that have been made about the benefits of graphical representations, as well as some factors that could count against their use, and whether some alternative representation scheme might be better. To make the discussion more concrete, we will examine the claims with reference to site maps, which are visual representations of web sites.

Site Maps

Many software applications intended to assist with the development of web sites make use of some form of site map. They are used for (at least) the following tasks: visualisation of "live" web sites without changing them (e.g. Visual Interdev); identification of files in the development source space for the purpose of initiating some activity, e.g. editing or publishing the resource on the web server (e.g. WebSphere Studio); design of web site from the perspective of pages (e.g. Dreamweaver[2], Denim[3]).

Site maps are a practical use of diagrams for the purposes of understanding and interacting with a domain. In the discussion to follow we will use examples from Visual Interdev and WebSphere Studio.

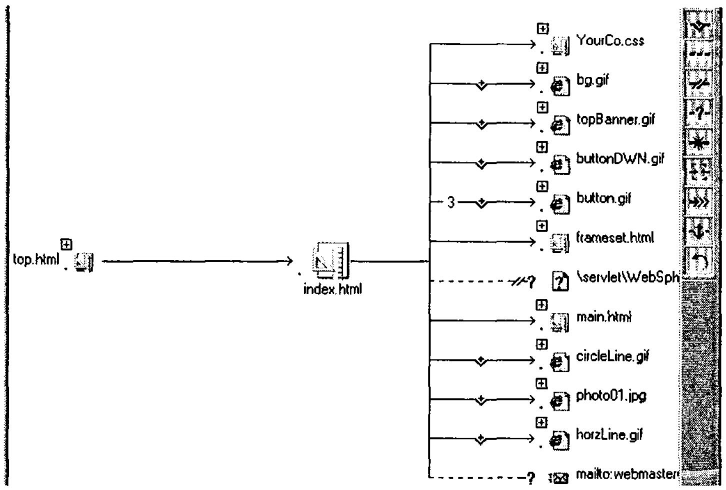

Figure 1: WebSphere Studio Relations View

The paper will focus on how successful site maps are at representing the domain, since this is the foundation of the higher level tasks listed above.[4]

The web site domain

When accessing a web site through a browser, the experience is of a collection of interconnected pages. In the early development of the web, pages were static (one HTML file had to be created for each visible page, one image file per image, and so on), and movement around

[1]Visual Interdev and WebSphere Studio are trademarks of Microsoft and IBM respectively.

[2]Dreamweaver is a trademark of Macromedia Inc.

[3]Lin *et al.* (2000).

[4]We will not discuss the use of site maps for design here—see Lin *et al.* (2000) for an investigation into designers' views of site maps as design tools.

the site was controlled by the user through explicit links. As technology has progressed, pages can now also contain forms with submit buttons which cause user input to be processed on the server, or small interactive applications. Technologies such as CGI scripts, NSAPI, Servlets, JSP, and ASP, allow the content of a visible page to adapt to user input, and data can be maintained in the background as users move around the site.

Dynamic generation of pages adds complexity to the domain, since there is no one-to-one correspondence between pages and files on the server. A program which contributes to the web site's functionality could generate zero, one, or several visible pages, or it could generate text or pictures that will become part of one.

Constructing a site map

Creating a site map involves two steps: generating complete data about the domain state[5], and creating an intelligible two-dimensional graph which adequately represents this data.

Considering the first step, if a web site consists only of static files, with statically defined navigation links and references to other files, then this task simply amounts to parsing HTML files for relevant HTML tags. This is not entirely without complexity. For instance, it involves detecting circularity (a page can have a link to another page which links back to the first one through a chain). Also, most web servers are configured to serve files of particular names as default (usually `index.html` if an incoming request only contains a path name). In that case, `http://theserver/path/` and `http://theserver/path/index.html` refer to the same resource (provided `/path/index.html` exists), which means that the parsing algorithm needs to be able to resolve such ambiguities.

The Visual Interdev parsing algorithm detects neither of these problems, in fact all URLs are treated as referring to a new resource. Most web sites contain more than one link to at least some file, so Visual Interdev graphs almost always contain multiple references to the same file, causing portions of the site to be repeated in the diagram.

If the the web site relies on dynamic generation of pages, however— specifically dynamic generation of the navigation links—then automatic discovery of all dependencies is difficult or even impossible. This is true regardless of whether the parsing algorithm inspects the program files or the generated pages. There are many reasons for this; we will only mention a couple for the purposes of illustration (both examples come from real web sites).

- Navigation from the entry page of the site is often presented to the user as a menu. This menu might be generated by a program that detects the presence of particular files when the user accesses the site. The links that the menu program produces are never present in the program source, so they would be missed if that is what the parsing algorithm inspects. Even when there is no dependency on an external state, links that are calculated on the fly may be difficult to identify for the parser.

  WebSphere Studio, which generates data by parsing source files, fails to find dynamically generated links, although if the user knows about them, they can be entered into the representation afterwards.

- Programs may generate different sets of pages depending on an external state. Say that a web application contains a set of forms which allows the user to administer printing on a remote server. Users can choose to set up, stop or restart the print services; add new printers; remove or delete existing printers. Which of these options make sense, and hence show up on the form, varies depending on what the current state of the system is. This application starts with a menu page with links to the subset of all possible pages corresponding to the tasks that make sense given the current state of the system. The links are references to different programs, each of which generate a form appropriate to the task in hand. If the parsing algorithm inspects a live site, it will only detect the subset of links that is generated for the current state of the remote system. The algorithm would miss several pages, and hence the presence of objects on the web servers.

  Since Visual Interdev parses the live site, it misses pages that are generated this way unless the user ensures that the programs have access to all possible states. However, since the parser cannot detect multiple references to the same resource, it cannot detect when that resource generates alternate pages. While all the alternate pages would be shown, the fact that they correspond to the same URL is not.

The second step in website mapping, creating a a two dimensional graph which represents the domain data, turns out to be highly problematic. Firstly, the domain data might not correspond to a planar graph. This is already true for statically defined navigation links—consider a domain consisting of five pages, each of which has links to the other four—and the domain contains other features which give rise to non-planarity as well, e.g. different types of disjunction (navigation routes and generation choices). Secondly, the number of objects and relations in the domain can be very large, which would require a large and cluttered display. Thirdly, the domain is such that the different types of disjunction need to be represented at the same time. Consider the example of the printer administration forms above. The menu page features disjunction of two types: the user's choices (which of the available tasks/navigation links to follow), and which of the possible tasks were shown given the state of the system. This difference (what alternative pages can be generated, as well as which

[5]Strictly speaking, generation of domain data is not an aspect of graphical representations. This step is required for other representations as well. It is however relevant to the question of whether a graphical representation is the most appropriate interface given the information users are intended to gather from site maps, and so is worth examination.

links are available on each one) should be represented if the diagram is to reflect navigation in the web site.

A graph that represents a whole site is likely to be too complex to either display or interpret, so both Visual Interdev and WebSphere Studio use strategies that allow users to inspect a subset of the domain at a time. These will be discussed in section below.

Some benefits of graphical systems

There are several types of argument for why visual representations could be better than sentential ones. For example, in some graphical representations the activity of constructing a diagram in itself is claimed to help the user reason about a problem domain. This is thought to be the case when a visual language is such that it cannot represent impossible states of the domain; or if the act of representing a set of premises automatically causes all their valid conclusions to be represented (as has been argued for Euler Circles).

With respect to the site maps considered here, users do not themselves construct the diagrams. However, the notion that the diagram might automatically represent all true facts about the domain still has an interpretation. In practice it means that whatever facts the user needs to know about the domain are available by inspecting the diagram—there is no need for them to do any additional inference work.

Another property of graphical representations which potentially confers a cognitive advantage over sentential ones is that they may allow representational objects to participate in two or more semantic relations simultaneously, thus being more "compact" and easier to read than sentential systems.[6] If the layout of the diagram permits, and it is appropriate for the domain, users should also be able to detect patterns of such relations across several objects.

With respect to site maps, these general points translate into some specific possibilities. With regard to compactness, for instance, a site map may simultaneously show all pages that have links to a page P, and all the pages to which P has links (only the latter would be found by inspecting P's source). This can be represented sententially in a format that is as easily accessible as a diagram, but only with reference to resources one link away. To represent the branching at the next level, at least some graphical annotation (e.g. indentation) seems necessary to make the sentential representation as readable as a diagram. The ability of site maps to show several objects and their relations at the same time means that the user can detect where pages exhibit systematic navigation patterns, allowing for better organisation both of the site and of navigation.

[6]Suggested by Kathi Fisler at the First CSLI workshop on Visual Reasoning, 1999.

Possible disadvantages of graphical systems

The possibility that for some tasks, graphical representations may be less suitable than sentential ones, or even detrimental, is rarely raised in the theoretical literature (though see Scaife & Rogers 1996 for cognitive issues, and Lemon & Pratt 1997 for formal arguments). In practice, when it comes to designing real applications, some of these problems are well known, so software designers have developed a set of standard strategies to work around them, which often involve restricting the portion of the domain that can be viewed at any one time.

This section discusses properties of some domains, visual languages, and tasks, which make it difficult to design suitable graphical systems, and may hence count against using diagrams in those circumstances. We also consider the common techniques for working around such problems, and assess what the trade-offs of these are.

Conflict between user assumptions and design reality

It is a feature of some formal systems that they conform to the *closed world assumption*—if a relation is not represented, then its negation holds; if an object is not represented, it does not exist. Natural language is such that neither in production, nor in interpretation do we assume that the closed world assumption applies. However, users of graphical systems may find it natural to adopt the closed world assumption (Levesque 1988) when reasoning with them, which is what many users appear to do.

This factor is relevant in the context of site maps, because as pointed out above, it is not possible to devise a fail-safe algorithm for inferring domain data about web sites (site maps are not unique in this respect, this is also true for e.g. two-way UML applications). In other words, information derived from the site map is not reliable. A user who believes that a site map represents the entire web site may derive erroneous information from it, while a user who is aware that it is incomplete can only use it for guidance—if the information is essential, it has to be confirmed through other means.

At least some users are aware of incompleteness as a generic problem of software which employs graphical representations, but still consider such interfaces valuable. Since incompleteness would seem to cancel many of the supposed benefits of graphical representations. This raises the question, which we return to below, of what *does* make them desirable.

Expressivity and inferential problems

There are formal results which show that some graphical systems are unsuitable for the tasks they were designed for (Lemon & Pratt 1997; von Klopp Lemon 2000). For instance, the system might not be able to represent all the structures required by the application domain, or it might lead the user to generate incorrect inferences. This happens even with common and simple systems, such as Euler Circles for representing statements about sets.

A generic problem with graphical representations is that they are ill-equipped to encode NEGATION,[7] i.e. there is no practical way to express absence other than relying on the closed world assumption. It was pointed out above that absence cannot be reliably interpreted as negation with site maps, since the data they are based on is likely to be incomplete. But this is not the only problem—not being able to represent negation explicitly also means that there are certain types of query that are not easy to answer by referring to diagrammatic representations. While diagrams can be effectively used to answer questions such as *"Which of the HTML files use this style sheet?"*, answers to questions such as *"Do all the HTML files reference this style sheet?"* or *"Do any of the HTML files not reference this style sheet?"* are harder to find. In contrast, results for this type of query can easily be reported and interpreted sententially, perhaps with the additional benefit that the user is less likely to assume that the representation is complete.

The problem of expressing negation can be ameliorated somewhat (though it is by no means solved) by using FILTERS. Both Visual Interdev and WebSphere Studio offer the user the possibility of screening out certain types of object, so that the diagram only shows HTML files for instance. By applying filters, the user may be able to display only objects of the relevant categories, which may enable the absence of tokens to be detected more easily. The risk associated with using filters is that the user fails to spot significant facts that involve objects or relations that have been filtered out.

Generalization problems

The repertoire of symbols that can make up a visual language is subject to restrictions that are not present in sentential languages. There is a natural tendency to categorise the symbols (lines, regions, etc.) and if the language employs too many types from the same category (e.g. different types of line or differently colored squares), they become hard to distinguish. Such distinctions must also be kept in working memory while interpreting the diagrams.

Under the right conditions, this tendency to correlate categories of diagrammatic object with categories in the domain can be exploited to make the diagram easier to understand. For domains which feature many different types of objects and relations, however, that same tendency can be problematic: objects or relations in the domain which should not be categorised together may have to be represented by symbols of similar spatial types. This can lead to unwarranted assumptions about similarity: for example, if A and B are represented by the same type of symbol, but do not belong to the same category in the domain. The converse problem, of failing to detect category membership, can also arise.

Consider for instance a site map which shows both navigation links in a page and inline links to e.g. an image or a program whose output is displayed within the page. These relations are conceptually different: if a resource referred to in an inline reference is missing, the page does not display as intended; if a resource referred to in a navigation link is missing the page displays correctly, but does not function as intended in the sense that the user cannot follow the broken link. The formal properties of both these relations (navigation and inline links) are such that the only sensible graphical representation of them is as lines connecting regions.[8] Thus the graphical types in the representation fail to distinguish different object types (e.g. different kinds of dependencies, or links) in the domain.

Common strategies for avoiding problems arising from the restricted repertoire of visual languages are filtering (discussed above), or by adding some notational convention to the spatial symbol, e.g. symbols such as arrowheads or kinks on lines; color; or even textual annotation. Visual Interdev and WebSphere Studio employ both. Filters are used to categorize objects, and notation is used to distinguish between different types of relations (Visual Interdev uses color; WebSphere Studio symbols) as well as different types of files (which can be recognized by their file extensions).

Adding notational conventions to an already complex diagram is likely to make it more difficult to interpret. It may also require the user to perform additional processing since the annotated objects have a more complex semantics. Stenning & Lemon (2000) explored the possibility that an advantage of at least some types of graphical representations over other types of representation is that users do not have to perform the equivalent of syntactic processing. This would make them easier to use than e.g. sentential representations. Clearly, any such advantage is lost if the visual language is augmented with annotation.

It is also relevant that the user will need to maintain the meanings of the annotations ("blue" means functional navigation link, "red" means broken link, etc.) in working memory. This adds to the effort of processing the diagram. This is an area where graphical representations may involve a type of processing that does not arise with familiar sentential representations, since with the latter it is normally possible to use mnemonic names for the objects and relations.

Size/layout/graph complexity problems

The problems with generating graphs for site maps were surveyed above. To summarise, the graph is likely to contain a large number of tokens, potentially features complex disjunction, and might be non-planar.

Humans are reasonably good at scanning sequential lists, but scanning a two dimensional representation for

[7]The literature also often mentions disjunction, but for the site map domain at least, this is complexity problem, not a notational one.

[8]These issues will be explored in von Klopp Lemon & von Klopp Lemon (2000).

information is harder. For information gathering tasks that would involve scanning an entire diagram for a particular graphical feature, it is therefore likely that a sentential representation would be more effective. Consider the task of identifying and correcting broken links. Both Visual Interdev and WebSphere Studio represent broken links. However, a sentential list consisting only of descriptions of where the problems are (the parsing algorithm has clearly generated this data) would be considerably more helpful to the user.[9]

Non-planar graphs cannot be constructed in two dimensions without assigning *null semantics* to certain graphical relations. For site maps this means that if the diagram represents pages/programs as regions and relations as lines, it is necessary to allow lines to cross each other without the crossing having any meaning in terms of the relations. This obviously makes the diagram harder to interpret since it is more difficult to follow lines that cross each other than ones that do not. Another reason why this might be relevant is that it has been claimed that graphical representations are easier to interpret than sentential ones precisely because every graphical relation corresponds to a relation holding between domain objects, and hence that the user need not expend any effort on determining which aspects of the representation are significant and which are not. Finally, there is some danger if null semantics is implicit. In another domain, crossing lines may be meaningful (consider a map where lines denote paths) so such decisions may not be transparent to the user.

A common technique for avoiding planarity problems is to use a stepping mechanism. This is what WebSphere Studio does: its display is centered around a single object, and shows all objects which depend on it, and all objects on which it depends (see Figure 1). Although this strategy successfully avoids non-planarity, it also severely restricts the number of pages and relations the user can survey at any one time. This means that it does not help the user to detect patterns and cycles in the site as a whole. Moreover, as pointed out above, this amount of information can be represented sententially in a way that is equally accessible as the graphical one.

Visual Interdev does not, as it were, need to employ a strategy to avoid non-planarity. Since it fails to detect when two URLs refer to the same resource, the problem does not arise in the first place. Of course, this feature also makes it unusable for detecting patterns.

Conclusion

Our objective was to survey graphical representations of web sites as user interfaces. The domain is very complex, which makes it difficult both to generate accurate data and to generate a graph that represents it. The tools take shortcuts, and employ simplification strategies, which have the effect that their usefulness for understanding the domain is diminished. Moreover, at least one task (that of identifying broken links) would be better supported by a sentential representation.

Despite these shortcomings, many users still consider site maps to have some merit, even if they are aware of their limitations. This suggests that while the literature describes diagrams in terms of supporting inference, in practice users find other types of benefits, at least in the domain of site maps. We suggest that the main advantages conferred by the use of site maps are firstly, that they provide an external representation of the resource situation (which is useful even though it is likely to be incomplete), and secondly that what is represented is shown in a compact manner (as discussed above). It is worth exploring whether this could be the case with other graphical systems.

[9]Since the displays show only connected diagrams, filters cannot be used to detect broken links.

Acknowledgements

We wish to thank Kathi Fisler, George Fink and Suresh Potiny.

References

Lemon, O., and Pratt, I. 1997. Logical and Diagrammatic Reasoning: the complexity of conceptual space. In Shafto, M., and Langley, P., eds., *19th Annual Conference of the Cognitive Science Society*, 430–435. New Jersey: Lawrence Erlbaum Associates.

Lemon, O.; de Rijke, M.; and Shimojima, A. 1999. Efficacy of Diagrammatic Reasoning (Editorial). *Journal of Logic, Language, and Information* 8(3):265 – 271.

Levesque, H. J. 1988. Logic and the Complexity of Reasoning. *Journal of Philosophical Logic* 17:355–389.

Lin, J.; Newman, M.; Hong, J.; and Landay, J. 2000. Denim: Finding a tighter fit between tools and practice for web site design. In *Proceedings of CHI 2000.* (to appear).

Scaife, M., and Rogers, Y. 1996. External cognition: how do graphical representations work? *International Journal of Human-Computer Studies* 45:185 – 213.

Shin, S.-J., and Lemon, O. 2000. Diagrams. Stanford Encyclopedia of Philosophy. (in preparation).

Stenning, K., and Lemon, O. 2000. Aligning logical and psychological perspectives on Diagrammatic Reasoning. *Artificial Intelligence Review.* (in press).

Stenning, K., and Oberlander, J. 1995. A Cognitive Theory of Graphical and Linguistic Reasoning: Logic and Implementation. *Cognitive Science* 19(1):97 – 140.

von Klopp Lemon, A., and von Klopp Lemon, O. 2000. The constraint matching theory of diagram design. in preparation for submission to Diagrams 2000.

von Klopp Lemon, O. 2000. Comparing the Efficacy of Visual Languages. In Barker-Plummer; Beaver; di Luzio; and van Benthem., eds., *Logic, Language, and Computation, Volume 3.* Stanford: CSLI Publications. (under review).