Showing Connection

Jeffrey V. Nickerson, Barbara Tversky and James E. Corter

Introduction

Information systems exist in a space based on connection: Weightless data stream through wires and float through air. Some parts of the system are visible—processors, routers, disks. But the things that flow move so fast they can't be seen.

In system design, there is no site plan, no elevation, no perspective drawing, and so one might expect little visualization and spatial reasoning to supplement symbolic language. But while information systems designers speak words and write code, they also make use of many of the techniques that originate in other design domains [1, 2], first and foremost diagramming [3–6]. What do and what should information systems designers visualize? In order to address this question in manageable parts and discover areas for future research, we will describe the practice of design as an interaction with a set of increasingly abstract spaces.

First, the *geographic space* that we inhabit. Designers can locate parts of a system using GPS coordinates. But only a little is learned about a system from the geographic locations of the components. Instead, systems are commonly described in *network space*, which shows the structural and temporal connections between components. Stepping up a level of abstraction, the design process itself is a network, each design the product of a series of linked decisions. Designs are imagined as points in *design space,* the space of all possible designs. While design space is a place in designers generate alternatives, these alternatives need to be judged in *evaluation space*. The dimensions in this space are design objectives. Creativity involves a shuttling between points in the design space and evaluation space: alternatives are compared against objectives, and new designs are generated by making new decisions that fill out desirable regions of the evaluation space. The designs themselves are often represented as networks, which are mapped into representations on the

J. V. Nickerson (✉)
Stevens Institute of Technology, Hoboken, USA
e-mail: jnickerson@stevens.edu

B. Tversky · J. E. Corter
Teachers College, Columbia University, New York, USA

J. S. Gero (ed.), *Studying Visual and Spatial Reasoning for Design Creativity,*
DOI 10.1007/978-94-017-9297-4_2

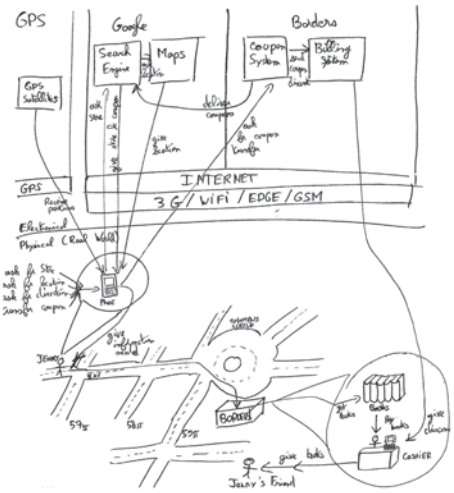

Fig. 1 A novice designer sketch for a system to distribute electronic coupons for a local bookstore. The geographic information at the bottom of the diagram is linked to the topological information at the top of the diagram

page, a part of geographic space. Hence the designer reasons in a variety of different spaces. The movement between spaces in the design process is seldom predetermined, but instead a result of situation, contingency, expertise, and style.

In the rest of the paper, we will discuss each space in more detail. We will point out places where research on visualization and spatial reasoning might further our understanding of design activity.

Geographic Space

When information systems are represented in geographic space, it is usually in order to show the location of particular components of a system: the network routers, the wires, the computers. Such locations are unimportant in many design situations. That is, even after installation, computers and network components can usually be moved without affecting the functions of the system. This movement affects the speed-of-light communication so little that differences in speed are undetectable.

There are, however, some situations in which geography is important. In the design of computer circuit boards, physical distance between wires can affect timing, heat dissipation, and the amount of radio interference. At a higher level, plans for wireless networks often show access points, and indicate with a circle the radio range within which other computers can connect.

Representation of systems in geographic space is usually straightforward, as the components can be placed on maps. An example of a design sketch that incorporates geography is shown in Fig. 1: the designer is showing how a bookstore might provide prospective customers electronic discount coupons. This example illustrates a broader phenomenon: wireless communication is giving renewed importance to

Table 1 Different forms of networks used in designing information systems. The last column, S/D, indicates if the network is meant to represent structure or dynamics

Nodes	Link basis	Representation	S/D
Computers	Wired or wireless network connections	Network diagram	S
People	Frequency of communication	Social network	S
Documents, people	Approval process	Workflow diagram	D
Software components	Calling structure	Call trees	D
Software objects	Inheritance	Class diagram	S
States of objects	Events	State diagram	D
Actors	Messages	Sequence diagram	D
Software, hardware	Software installed on hardware	Deployment diagram	S

location. The visualization conventions for wireless networks are still evolving, and present opportunities for research on both the production and understanding of diagrams addressing mobility.

Network Space

In contrast to geographic space, network space has no coordinates. Instead, a thing is described solely in terms of its connections. These very abstract structures can be used to describe many different phenomena, and with respect to information systems, there are a wide range of network descriptions, both standard ones taught to practitioners [7] and informal ones invented and reinvented to fit particular purposes [8]. These descriptions can be roughly classified into those that describe structure—what connects to what—and those that describe dynamics—the connecting order. Table 1 lists some of the many types of networks used to describe information systems.

Structural networks have as nodes people, computers, and software applications. These nodes have identifiers, and can be located in relation to neighbors, but there are seldom fixed locations. Figure 2 shows a typical sketch: the components are connected by lines that indicate interaction.

Other networks are constructed to indicate dynamics. In such networks, links often indicate the flow of a particular message from place to place. For example sequence diagrams, shown in Fig. 3a, show messages and the order of messages flowing from actor to actor in a system.

The vertical dimension of the diagram in Fig. 3a indicates time, whereas the horizontal dimension shows an instantaneous event, a message being passed. From traces of such messages, it is possible to specify the interfaces of an actor in the system. That is, just as one can specify many of the duties of managers in companies by watching their interactions, the interactions of software components the messages provide guidance on how they should be built. Software tools can help automate this process, and it is normal for design tools to facilitate this. But current tools are

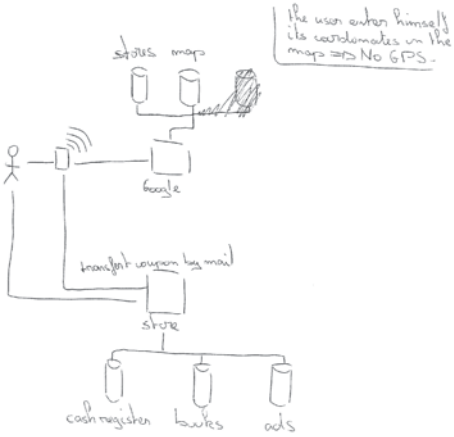

Fig. 2 Another example of the bookstore coupon problem, drawn by a different designer as a network

less useful in the next phase, when designers decide how to structure the connections between components. To what extent can the optimal structure of a network can be automatically inferred from the intended interactions shown in a diagram? One exploratory approach uses techniques from the psychological similarity literature [9]: Fig. 3b shows such an inference [10].

Time can also be combined in a single diagram, as shown in Fig. 4. These diagrams are effective at making inferences about how long a sequence of overlapping tasks will take, as the positions of the nodes in the tree correspond to elapsed time [11]. Are such combined diagrams easier or harder to infer from than a set of diagrams, such as those shown in Fig. 3? On the one hand, multiple diagrams demand designers perform a difficult integration task in the mind, but on the other hand, they avoid confounding structural and temporal aspects of a problem. This issue is important because increasingly computer systems rely on multiple processors

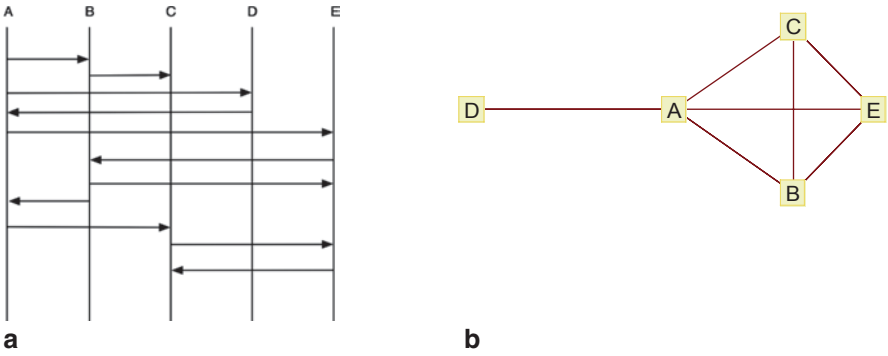

Fig. 3 On the left, **a** a sequence diagram: arrows indicate messages running between the actors of the system, *A*, *B*, *C*, *D* and *E*. On the right, **b** shows a network that is consistent with the sequence diagram, derived automatically using a technique discussed in [9]

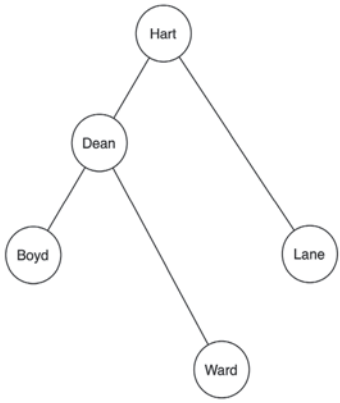

Fig. 4 A representation of a calling tree, in which the locations of the nodes indicate the elapsed time of the calls

running tasks in parallel. Designers need to decide which tasks can be run in such a way, and which must be run sequentially, and these decisions can have large effects on the performance and reliability of a system.

How do we reason about abstract networks? Network diagrams assume designers are capable of disregarding spatial information to focus on only on topological information, but this assumption may be optimistic. Studies of interactions with network diagrams have shown that distance along a line in the diagram is perceived more readily than topological distance [12]. Moreover, errors in enumerating paths on a network correlate with distance [13]. And even the positions of nodes in a network, information that should be arbitrary, are often uniform, based on cultural associations: for example, most designers will show, as in Fig. 1, a network provider above a store [6]. Consequently, we wonder the extent to which Euclidean bias leads to inferior decision making in the design and diagnosis of systems. It may be that training focused on topology will improve design and comprehension of systems, or it may be that we need different representations for systems that take into account our cognitive apparatus.

The tools created in order to facilitate systems design, Computer Aided Software Engineering tools, are not used very much in practice [14]. Programmers stick with a process of informal diagram sketching followed by coding in word processors. Are their tools that are less proscriptive, that can help a programmer without imposing onerous restrictions? Some think that tools that encourage reflective practice [15] will outdo existing tools by providing a gentler kind of guidance [16].

Even as software engineers struggle to find better ways of representing existing systems, emerging technologies present opportunities for new kinds of visualization that combine network and geographic space. For example, roads today can be mapped, using sensor data, to show the average speed on a particular portion of the road. Roads can also be mapped according to their connectivity to wireless access points or cell phone towers. It becomes possible to reason about how to traverse a road system while maintaining connection [17, 18].

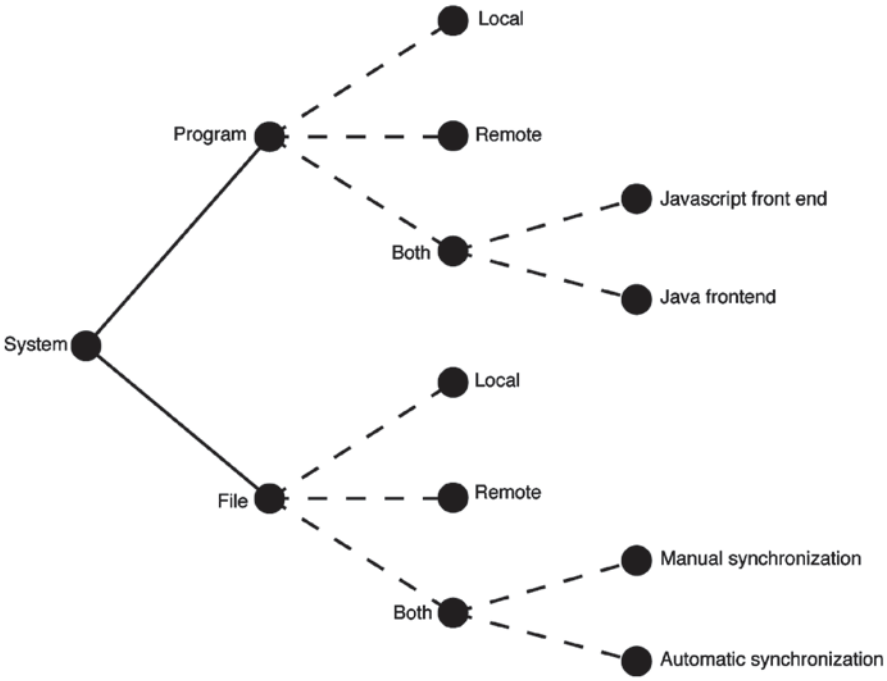

Fig. 5 A design tree for a system (for example a word processing application) that might have both local and cloud components. The tree is based on Brooks [3]: *Solid lines* indicate independent questions that all must be answered, and *dotted lines* indicate a mutually exclusive choice: only one path can be followed

Design Space

Design involves a progressive series of decisions [19]. Understanding the decision tree can be important for two reasons: it can aid in the generation of new designs, and it can help record design decisions, which can be useful for joining members of a design team.

How, then, should the decision tree be imagined? Brooks recommends what he calls a *design tree*, in which design questions are represented as nodes. Nodes send out either independent edges or mutually exclusive edges. All independent edges need to be followed. Mutually exclusive edges force a choice: only one of these several edges should be followed to create a design [3].

For example, imagine creating a design for a new word processor. The problem can be broken up into how to design the program, and how to design the storage of files. The program might reside locally, on the cloud, or both. Likewise, the data may reside locally, on the cloud, or be replicated in both locations. Figure 5 shows the decision tree.

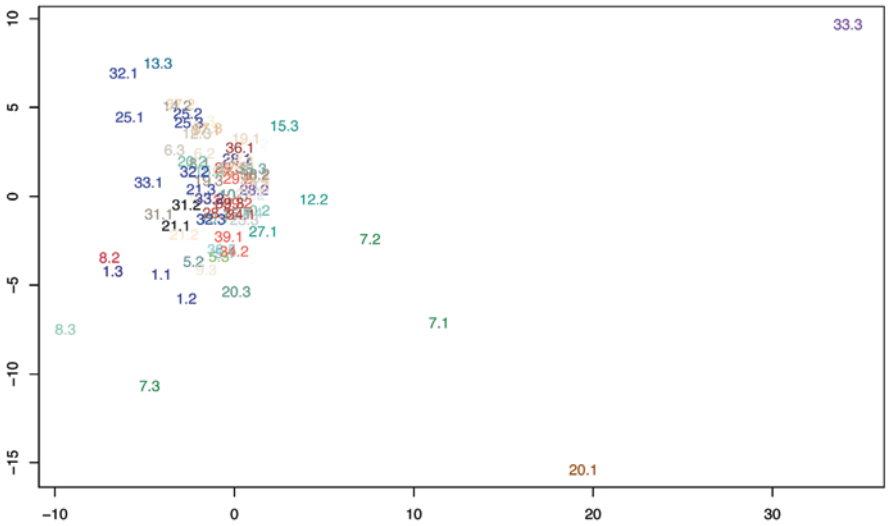

Fig. 6 A set of design alternatives: each individual generated three alternatives: for example, 7.1, 7.2, 7.3. The designs were compared based on their component connections, and the points mapped into the figure using multi-dimensional scaling. More detail on the method is discussed in [6]

The tree becomes a way of exploring the space and documenting decisions: each design is constituted by the nodes traversed from the root on the left to the leaves on the right. Thus, the entire design space, all possible designs, is a tree of trees. That is, there are a finite number of choices that can made in the above tree—there are four ways program can be deployed, and independently four ways data can be deployed, and thus there are 16 possible design alternatives. The tree might be a useful discovery aid in enumerating possible designs.

There are other ways of thinking about exploring the total design space For example, consider all the variations that have already been made. We might be able to explore the space by looking at the relative differences between the alternative designs of an individual designer, or better yet, the designs of a crowd of designers working independently. If a design can be expressed as a network, then the distance between two designs is just the graph edit distance [20]: the number of edges that would need to be either added or subtracted from the set [6]. Then, such a set of distances can be visualized using multidimensional scaling [21]. We show an example of this process in Fig. 6.

The above figure is derived from the designs of a crowd: that is, designers working independently. Each had created three variations, and the graph can be used to see how much individuals vary among themselves, and where the designs seem to converge.

Often design is done collaboratively, with developers modifying other's work. For example, the website Scratch [22, 23] provides a way for children to remix, that is modify, each other's programs, and thereby teach each other programming in

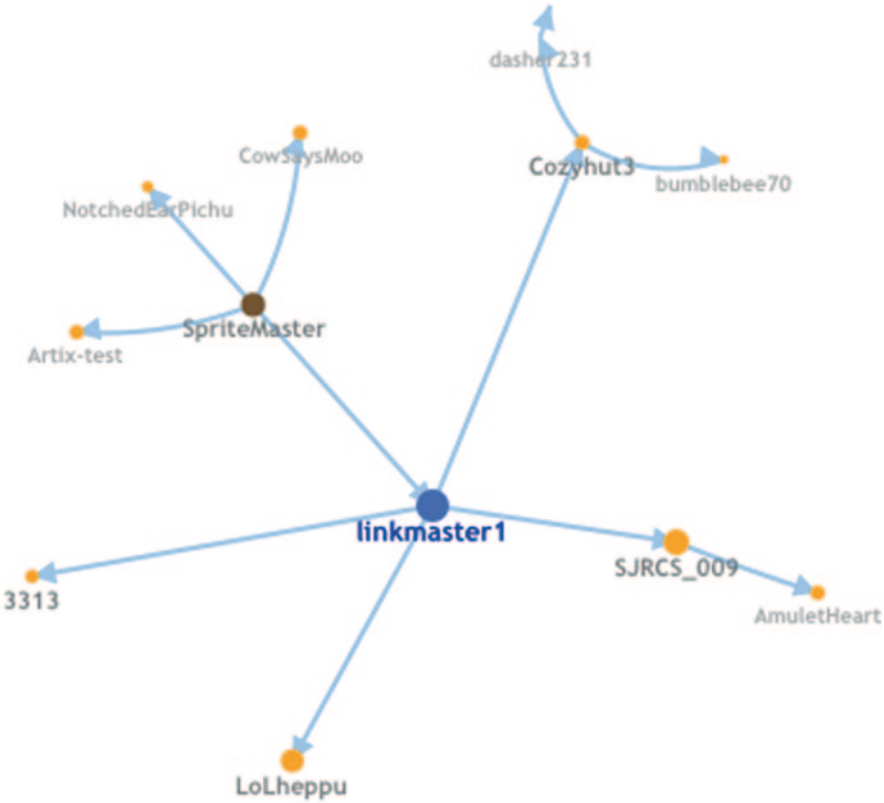

Fig. 7 A visualization showing the chain of programmers that have modified each other's code for a particular set of linked projects on the website Scratch: http://scratch.mit.edu/projects/SpriteMaster/1054710, as of 5/17/2010

the process of creating animations and games. In order to encourage remixing, the developers of Scratch have added a visualization, based on the history of remixes. Clicking on a node recenters the graph and emphasizes the local neighborhood of remixes [24]: the result is shown in Fig. 7. This diagram also gives a sense of both the design tree for a project, and the overall design space, by which variations have led to. One can imagine combining the methods shown in Figs. 6 and 7, by showing not only who modified a project, but how much the project was modified.

Evaluation Space

Designs not only need to be generated, they also need to be evaluated. There are often both a set of requirements with a design activity, and an overall set of criteria that create a space in which each design can be evaluated. For example, a word

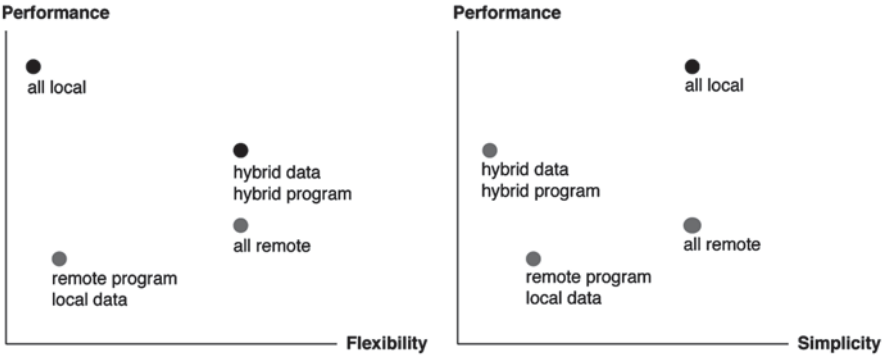

Fig. 8 On the left, **a** an evaluation of four designs from the design space of Fig. 5, based on the criteria of Performance and Flexibility. On the right, **b** these same designs evaluated on Performance and Simplicity

processor will be required to fulfill a long list of requirement relating to editing, file saving, and formatting. Assuming these can be fulfilled, there are set of general criteria that often determine the overall effectiveness of the system. For instance, the 16 designs that can be generated from Fig. 5 each can be evaluated in relation to performance and flexibility; Fig. 8a shows several designs mapped into this evaluation space.

The graph shows there is a tradeoff: local systems will perform better, but are less flexible. They are less flexible because users can't reach over the network and retrieve a file the same way they can with a cloud-based application. The designs shown as gray dots are relatively worse, because they perform no better along any dimension than the systems shown as black dots. In particular, a remote program that works off of a file on my local workstation is worse with respect to both performance and flexibility than an all local system or a hybrid system, in which both data and programming are distributed onto local and remote machines.

Designers fight over the criteria to be used in evaluation [25]. Someone who has learned from past experience that simplicity is an important systems virtue might substitute this criteria for flexibility, and would then choose a local solution, as shown in Fig. 8b. Furthermore, Brooks [3] points out that sometimes new criteria are discovered as part of the design process, and so there is often a shuttling back and forth between evaluation and generation, as ideas are generated, evaluated, and new solutions are sought that fill out parts of the design space. Criteria are sometimes added: for example, a team of programmers may decide that performance, flexibility and simplicity are all important. The designers then alternate between exploring the design space by making different choices in the tree shown in Fig. 5, and evaluating the solution in a three dimensional evaluation space, the two projections of which are shown in Fig. 8.

Not all designers work so rationally, and it is an open question how expert designers who claim they work intuitively successfully find designs that satisfy design criteria. There is, however, a class of designers that are by definition systematic:

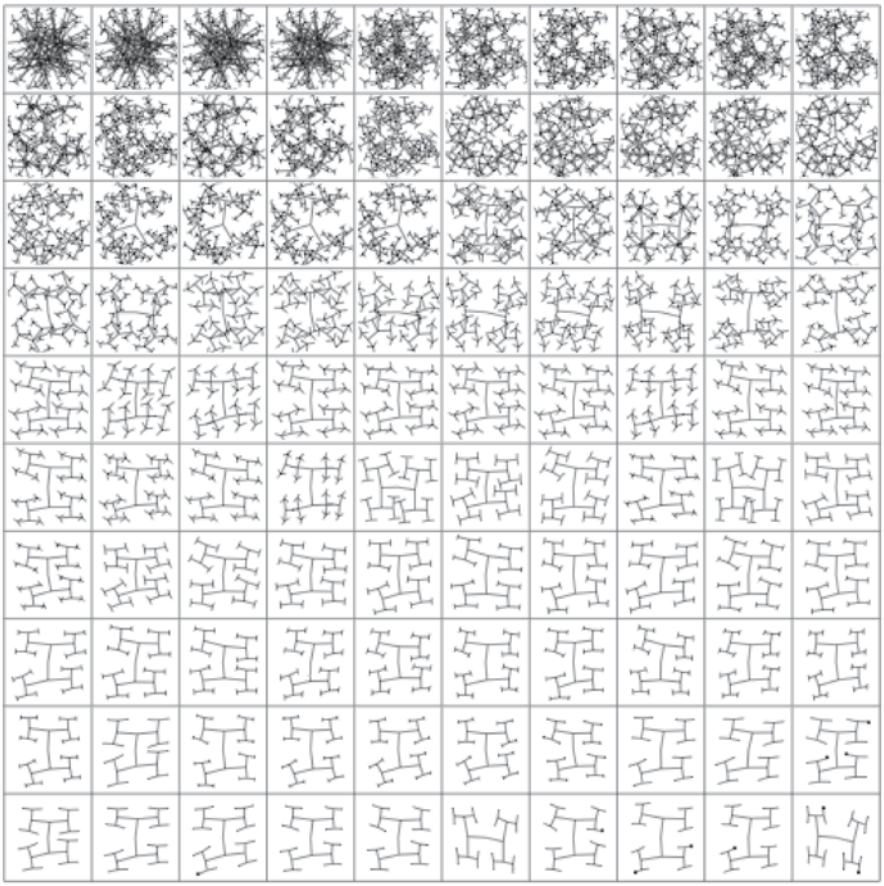

Fig. 9 A genetic algorithms exploration of a sensor network design space. Each cell is a point in the Pareto-Optimal set. The shape that optimizes the material used, in the bottom right corner, is an example of an H-Tree, re-discovered by the algorithm, but previously used in several areas, including VLSI design [31]

machines. In fields from architecture to engineering [26], automated systems explore design spaces, seeking optimal solutions: The output of such systems can look remarkably creative [27]. A range of techniques often used to explore design space are called meta-heuristic multi-objective optimization techniques [28]; for example, genetic algorithms combine solutions to create new ones, and these solutions are then considered in a multi-dimensional evaluation space so that a full range of alternatives are found. Such techniques are sometimes applied to computer programs themselves: that is, new software programs are generated automatically that fulfill specific design objectives [29]. Genetic algorithm techniques have also been applied to software architecture [30]. Figure 9 shows such a technique applied to a sensor network problem. The two criteria were: first, to minimize the cost

of the network, as measured by the total amount of material used, and second, to maximize the amount of coverage. The 100 solutions shown in Fig. 9 are all optimal, meaning that there are no other solutions in the design space better than them, assuming all weights of the two criteria are equally valid.

When problems have a spatial aspect, then the chances of a correspondence between design space and evaluation space are increased. That is, a small change in the design space will generally result in a small change in evaluation space, and finding this correspondence can provide insights into the design process [32]. But in the case of an information systems design that involves a network, small changes in the network will often break the topology. Thus, a more indirect way of encoding the system is needed: for example, mapping each potential network onto a permutation [33]. Such indirect ways are useful in that they guarantee that any network considered will fulfill the systems requirements. However, our ability to reason spatially about the process becomes difficult or impossible, and the exploration of a permutation space may require large amounts of computational power. An open question is whether or not the intuitions of an expert design might suggest other ways of encoding and traversing the highly abstract topological spaces of information systems, perhaps through spatial reasoning. Such reasoning may exist even in the design of abstract systems, because simple spatial structures underlie many common cognitive tasks [34].

There is a non-automated approach that is showing promise. Crowdsourcing marketplaces (for example, [35]) make it possible to divide design tasks into small parts and allow thousands of human participants to engage in design activity. Can groups of independent designers tackle scale design problems? We saw before that humans can be used to generate individual alternatives that together may traverse a swath of the design space [6]. The crowd can also be used at a higher level to establish a correspondence between common situations and common technical mechanisms that are useful in such situations. Figure 10 shows the consensus of 30 designers about which technical mechanisms apply to a set of common situations [36].

When is the crowd better than an automated approach? When is close-knit team of designers better than a crowd? These are areas for exploration. It is possible that difficult problems may yield to a combination of traditional and new approaches to design; for example, close-knit team processes augmented by computational methods that perform evaluation, or crowd-based processes feeding unfinished ideas to expert designers for evaluation and refinement.

Concluding Thoughts

Design of information systems involves grappling with a set of abstract spaces. There is little visible in an information system, and much of the system is dynamic, transient. So the designer needs ways to get a handle on the system. Geography is important in few a situations, but in most situations connections are much more revealing. Therefore designers spend most of their time constructing networks that

Fig. 10 The consensus evaluation of a set of designers asked to match technical mechanisms to common situations

together can describe the behavior of the system so that it can be constructed. Decisions are made, and the end result of the decisions are alternative designs. These many designs together form a design space, and variations in this space can be generated intuitively, or systematically by making different decisions in a design tree. But these choices need to be evaluated: once evaluated, the designer often moves back into generation mode, trying to find new solutions that explore a desirable part of the design space.

The design process is not always systematic, nor is it always conscious. Simon's rational decision approach [37] was critiqued by Winograd and Cross, among others [5, 38]. Still, much of what Simon said still underlies the current conversations about design science [39–41]. What has been tempered in current conversation is the belief in universal approaches and solutions. Domains are distinct, situations are different, and the design process itself is political [42]. We have dampened our enthusiasm for a proscriptive sequence of design activity, because we know that new requirements will be uncovered as the process proceeds [3, 43]. Yet Simon pointed out the importance of visualization, and we are even more convinced today of its importance in design [1, 2]. Fixed diagrammatic conventions, as in [7], are useful because they allow common communication, but we don't fully understanding how

well these representations are understood by practitioners, whether alternative ones would be better, and how completely these representations cover the many abstract spaces that that need visual expression.

We know information systems design is both visual and verbal, and that abstraction is an important prerequisite to the production of novelty—generally, and in information systems design [44]. We know it is easy for novice designers to become confused by even simple abstract diagrams [11–12]. We also know that designers are inventive, creating hybrid representations to apply to particular situations [6, 8]. As a field, we are still in the process of learning how to guide the novice, and augment the expert, by providing appropriate tools and techniques. Looking to the future, new programming languages geared toward children are helping create communities of computationally fluent youngsters [22]. The children's collective community emulates the adult open source community, and both are examples of the growth of peer production in many facets of creative work [45]. Alongside this growth in human capacity is the growth of machine capacity, in clusters, desktops, laptops, tablets, and phones. Thus we anticipate fast increase in our collective cognitive and computational capacity to design information systems. Representations that integrate individual, team, crowd, and machine may be the levers of distributed cognition.

Acknowledgements This work was supported by grants from the National Science Foundation, IIS-0725223, IIS-0855995, and IIS-0968561.

References

1. Tversky B, Suwa M (2009) Thinking with sketches. In: Markman AB, Wood KL (eds) Tools for innovation. Oxford University, Oxford
2. Visser W (2006) The cognitive artifacts of designing. Erlbaum, Mahwah
3. Brooks FP (2010) The design of design: essays from a computer scientist. Addison-Wesley, Upper Saddle River
4. Shaw M, Garlan D (1996) Software architecture: perspectives on an emerging discipline. Prentice Hall, Upper Saddle River
5. Winograd T (1996) Bringing design to software. ACM Addison-Wesley, New York
6. Nickerson JV, Corter JE, Tversky B, Zahner D, Rho YJ (2008) The spatial nature of thought: understanding information systems design through diagrams. Proceedings of the International Conference on Information Systems, Paris, France
7. Booch G, Rumbaugh J, Jacobson I (2005) The unified modeling language user guide. Addison-Wesley, Upper Saddle River
8. Nickerson JV (2005) The meaning of arrows: diagrams and other facets in system sciences literature. 38th Hawaii International Conference on System Sciences
9. Hutchinson J (1989) Netscal: a network scaling algorithm for nonsymmetric proximity data. Psychometrika 54(1):25–51
10. Nickerson JV, Corter JE (2009) Clarity from confusion: using intended interactions to design information systems. Proceedings of the Fifteenth Americas Conference on Information Systems
11. Nickerson JV, Tversky B, Corter JE, Yu L, Mason D (2010) Thinking with networks. Proceedings of the 32nd Annual Conference of the Cognitive Science Society

12. Nickerson JV, Corter JE, Tversky B, Zahner D, Rho YJ (2008) Diagrams as tools in the design of information systems. In: Gero JS, Goel A (eds) Design computing and cognition '08. Springer, Dordrecht
13. Corter JE, Rho Y, Zahner D, Nickerson JV, Tversky B (2009) Bugs and biases: diagnosing misconceptions in the understanding of diagrams. Proceedings of the 31st Annual Conference of the Cognitive Science Society
14. Iivari J (1996) Why are CASE tools not used? Commun ACM 39(10):94–103
15. Schön DA (1983) The reflective practitioner: how professionals think in action. Basic Books, New York
16. Redmiles D, Nakakoji K (2004) Supporting reflective practitioners. Proceedings of the 26th International Conference on Software Engineering (ICSE'04) pp 688–690
17. Cottingham DN (2008) Vehicular wireless communication Ph.D. dissertation, University of Cambridge Computer Laboratory, September 2008
18. Nickerson JV (2005) A concept of communication distance and its application to six situations in mobile environments. IEEE Trans Mob Comput 4(5):409–419
19. Kunz W, Rittel H (1970) Issues as elements of information systems. Working paper No. 131, Studiengruppe für Systemforschung, Heidelberg, Germany, July, 20
20. Sanfeliu A, Fu KS (1983) A distance measure between attributed relational graphs for pattern recognition. IEEE Trans Syst Man Cybern 13:353–362
21. Shepard RN (1962) The analysis of proximities: multidimensional scaling with an unknown distance function. Psychometrika 27:125–140
22. Resnick R, Maloney J, Monroy-Hernández A, Rusk N, Eastmond E, Brennan K, Millner A, Rosenbaum E, Silver J, Silverman B, Kafai Y (2009) Scratch: programming for all. Commun ACM 52(11):60–67
23. Monroy-Hernández A, Resnick M (2008) Empowering kids to create and share programmable media. Interactions 15(2):50–53
24. Chen R, Monroy-Hernandez A (2010) Personal communication.
25. Nickerson JV, Yu L (2010) There's actually a car, perspective taking and evaluation in software-intensive systems design conversations, NSF Workshop on Studying Professional Software Design, Irvine.
26. Gero JS, Maher ML (eds) (1993) Modeling creativity and knowledge-based creative design. Erlbaum, Hillsdale
27. Bentley PJ, Corne DW (eds) (2002) Creative evolutionary systems. Morgan Kaufmann, San Francisco
28. Deb K (2001) Multi-objective optimization using evolutionary algorithms. Wiley, Chichester
29. Iba H, Paul TK, Hasegawa Y (2010) Applied genetic programming and machine learning. CRC, Boca Raton
30. Martens A, Koziolek H, Becker S, Reussner R (2010) Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In Proceedings of the First Joint WOSP/SIPEW international Conference on Performance Engineering
31. Ullman JD (1984) Computational Aspects of VSLI. Computer Science Press, Rockville
32. Mackenzie CA, Gero JS (1987) Learning design rules from decisions and performances. Artif Intell Eng 2(1):2–10
33. Hodge LE, Whitaker RM, Hurley S (2006) Multiple objective optimization of bluetooth scatternets. J Comb Optim 11(1):43–57
34. Kemp C, Tenenbaum JB (2008) The discovery of structural form. Proc Natl Acad Sci 105(31):10687–10692
35. Amazon Mechanical Turk, Artificial Artificial Intelligence. https://www.mturk.com/mturk/welcome.
36. Nickerson JV, Zahner D, Corter JE, Tversky B, Yu L, Rho YJ (2009) Matching mechanisms to situations through the wisdom of the crowd. Proceedings of the International Conference on information systems
37. Simon HA (1969) The sciences of the artificial. MIT, Cambridge

38. Cross N (2006) Designerly ways of knowing. Springer, London
39. Gross MD (2003) How is a piece of software like a building? Toward general design theory and methods, in the National Science Foundation (NSF) Invitational Workshop on Science of Design, Software Intensive Systems, Airlie Center, Virginia
40. Hevner AR, March ST, Park J, Ram S (2004) Design science in information research. MIS Quarterly 28:75–105
41. Lee AS, Nickerson JV (2010) Theory as a case of design: lessons for design from the philosophy of science. Proceedings of the 43rd Annual Hawaii International Conference on systems science
42. Bergman M, King J, Lyytinen K (2002) Large scale requirements analysis revisited: the need for understanding the political ecology of requirements engineering. Requir Eng J 7(3):152–171
43. Gero JS (2003) Situated computing: a new paradigm for design computing. In Choutgrajank A, Charoenslip E, Keatruangkamala K, Nakapan W (eds), CAADRIA03, Rangsit University, Bangkok, pp 579–587
44. Zahner D, Nickerson JV, Tversky B, Corter JE, Ma J (2010) A fix for fixation? Re-representing and abstracting as creative processes in the design of information systems, Artificial Intelligence in Engineering Design, Analysis and Manufacturing, Maher M, Kim YS, Bonnardel N (eds) 24, 2.
45. Benkler Y (2006) Wealth of networks: how social production transforms markets and freedom. Yale University, New Haven