University of Colorado at Boulder

Department of Computer Science

ECOT 7–26 Engineering Center
Campus Box 430
Boulder, Colorado 80309–0430
(303) 492–7514, FAX: (303) 492–2844

Rethinking and Reinventing Artificial Intelligence from the Perspective of Human-Centered Computational Artifacts

Gerhard Fischer

Center for LifeLong Learning & Design (L^3D)
Department of Computer Science and Institute of Cognitive Science
Campus Box 430, University of Colorado, Boulder, CO 80309
gerhard@cs.colorado.edu

In *Proceedings of the Brasilian Symposium on Artificial Intelligence (SBIA)*, Campinas, Brasil, October 1995, pp. 1-11.

Rethinking and Reinventing Artificial Intelligence from the Perspective of Human-Centered Computational Artifacts

Gerhard Fischer
Center for LifeLong Learning and Design (L^3D)
Department of Computer Science and Institute of Cognitive Science
University of Colorado at Boulder

Abstract. Many research efforts in Artificial Intelligence (AI) have focused on replacing rather than augmenting and empowering human beings. We have developed human-computer collaboration environments to demonstrate the power and the possibilities of Intelligence Augmentation (IA) with human-centered computational artifacts.

A focus on IA instead of AI requires different conceptual frameworks and different systems. Our theory is centered around shared representations of context and intent for understanding, mixed-initiative dialogs, management of trouble, and integration of working and learning. Our system building efforts exploit the unique properties of computational media rather than mimicking human capabilities. We have developed (i) specific system architectures including a multifaceted architecture characterizing the components of design environments, and a process architecture for the evolutionary development of such environments, (ii) specific modules such as critiquing systems, and (iii) a variety of design environments for specific application domains.

1. IA: The New Look of AI

Traditionally, the most widely understood goal of Artificial Intelligence has been to understand and build autonomous, intelligent thinking machines. We believe with a number of other researchers that a more important goal is to understand and build interactive knowledge media or collaborative problem solving environments. The idea of human augmentation, beginning with Engelbart (Engelbart and English 1968) has been elaborated in the last 25 years (e.g., (Stefik 1986; Hill 1989; Fischer 1990; Bobrow 1991; Norman 1993; Terveen 1995) and is gaining widespread acceptance as the dominant research paradigm for AI.

In collaborative problem solving systems, users and the system share the problem solving and decision making and different role distributions may be chosen depending on the user' goals, the user's knowledge and the task domain. Any collaborative systems raises two important questions:

- What part of the responsibility has to be exercised by human beings?
- How do we organize things so that the humans can communicate effectively with the computational system?

Collaborative systems can be differentiated from autonomous systems (such as expert systems) along the following dimensions:

- a partial understanding and knowledge of complex task domains is acceptable,
- two agents can achieve more than one, especially by exploiting the asymmetry between agents,
- breakdowns are not as detrimental, especially if the system provides resources for dealing with the unexpected,
- background assumptions do not need to be fully articulated beforehand, but can be incrementally articulated,
- semi-formal system architectures are appropriate, and
- humans enjoy "doing" and "deciding" by being involved in the process.

Collaborative problem-solving approaches do not deny the power of automation (Billings 1991), but they focus our concerns on the "right kind of automation" including interaction mechanisms designed for humans rather than for programs.

2. Requirements for Collaborative Problem Solving Systems

Beyond Intelligent, Deaf, Blind and Paraplegic Agents. Bobrow (Bobrow 1991) has characterized the prevalent style of building AI systems as follows: *"a human agent will formulate the problem in a previously defined language understandable to the computational environment including background knowledge which will then use logical reasoning to achieve the desired goal."* This isolation assumption has led to a deemphasis of collaborative systems' dimensions such as communication, coordination and integration.

Beyond User Interfaces. Effective human-computer collaboration is more than creating attractive displays on a computer screen: it requires providing the computer with a considerable body of knowledge about a task domain, about users and about communication processes. Computational environments need to support *human problem-domain communication* (Fischer and Lemke 1988) by modeling the basic abstractions of a domain (as pursued in efforts in domain modeling) thereby giving designers the feeling that they interact with a domain rather than with low-level computer abstractions. Domain-orientation allows humans to take both the content and context of a problem into account, whereas the strength of formal representations is their independence of specific domains to make domain-independent reasoning methods applicable (Norman 1993).

Mixed-Initiative Dialogs. Despite the fact that communication capabilities such as mixed-initiative dialogs (Carbonell 1970) have been found crucial for intelligent systems, the progress to achieve them has been rather modest. Collaborative systems must support information volunteering by the users as well as by the system. Real users are not just data entry clerks (a role which they are left with in many expert systems), but they must be able to volunteer information (Fischer and Stevens 1991) and integrate new knowledge. On the other hand, humans often learn by receiving answers to questions which they have never posed or which they were unable to pose. Information access techniques must be complemented by information volunteering techniques in which the systems provides information which is relevant to the task at hand (Nakakoji and Fischer 1995).

Shared Understanding. Collaboration is a process in which two or more agents work together to achieve shared goals. In human-centered approaches, these goals originate from the humans, not from the computational artifact. This sets our approaches apart from expert systems and intelligent tutoring systems which are based on system-driven architectures. In our research, a shared understanding is achieved through the domain orientation, the construction and the specification of an artifact(Fischer, Nakakoji et al. 1993).

3. Examples of Human-Centered Computational Artifacts

Domain-Oriented Design Environments. Domain-oriented design environments (DODEs) support collaboration between (1) all stakeholders in a design process and (2) between stakeholders and computational environments. They serve as model for the design of collaborative systems by exploring and supporting different relationships and task responsibilities between humans and computers. Within human-computer collaboration, they are grounded within the complementary approach rather than the emulation or replacement approach by exploiting the strengths and the weaknesses of human and computational agents. They are semi-formal systems that integrate object-oriented hierarchies of domain objects, rule-based critiquing systems, case-based catalog components, and argumentative hypermedia systems. They do limited reasoning and interpretations, trigger breakdowns, deliver information, and support the exploration of the rationale behind the artifact.

By representing models of specific domains, DODEs provide a shared context and ground design with representations supporting mutual education and understanding by all stakeholders. In our research, *DODEs* have emerged as systems serving the integration of working, learning, and collaborating by modeling problem domains. They (1) allow users to focus on their tasks (and not just on the interface), (2) increase the usefulness without sacrificing usability, (3) facilitate human problem-domain interaction, and (4) support short-term and indirect, long-term collaboration. In the context of these research efforts, we have explored topics such as design by composition, design by modification, the integration of problem framing and problem solving, the use of critics to increase the back-talk of situations, and the reconceptualization of breakdowns as sources for creativity.

Critiquing. The critiquing approach (Fischer, Lemke et al. 1991) is an effective way to use computational knowledge bases to aid users in their work and to support learning on demand. In many design situations, artifacts do not speak for themselves. To address this problem, we augmented our environments with computational critics identifying breakdowns, which might have remained unnoticed without them. These breakdowns provide learning opportunities for designers by supporting them to reframe problems, to attempt alternative design solutions, and to explore relevant background knowledge. Critics in design environments "look over the shoulder" of users as they perform tasks in computational environments and signal breakdowns and offer critiques from time to time. Critics compute their advice by using domain knowledge to examine the actions users perform (e.g., information spaces visited) and the products they create (e.g., constructions and specifications). In critiquing, humans select (partial) goals and communicate some of them to the system, attempt to achieve these goals, and retain control of the interactions. Critics detect potential problems and provide information relevant to the identified problems. Users evaluate the critiques and decide how to respond.

Examples of Design Environments. Over the last ten years, we have developed design environments in several domains, including kitchen design (Nakakoji and Fischer 1995), computer network design (Fischer,

Grudin et al. 1992) and voice dialog design (Repenning and Sumner 1992). The Voice Dialog Design Environment (VDDE) illustrates our conceptual framework. Voice dialog interfaces consist of a series of voice prompted menus. Users press buttons on a telephone keypad and the system responds with appropriate voice instructions. Current interface design techniques for voice dialog systems are based on flow charts. It is difficult for designers, customers and end-users (representing the different stakeholders, each suffering from their symmetry of ignorance) of these systems to anticipate what the (audio) interaction will sound by simply looking at a static visual diagram. To experience breakdowns, simulations are needed which can serve as representations for mutual understanding by allowing designers, customers and end-users "experience" the actual audio interface. The VDDE allows domain designers to create graphic specifications. The behavior of the design can be simulated at any time. Design simulation consists of a visual trace of the execution path combined with audio feedback of all prompts and messages encountered. The designer can activate different rule sets for critiquing and determine with the critiquing parameter the "intrusiveness" of the critics. VDDE is implemented in Agentsheets and Hypercard using AppleEvents for the communication between the two systems.

With our prototypical developments, we were able to demonstrate the following learning and collaboration possibilities provided by DODES:

- the creation of a mutual understanding between stakeholders through an external representation of the artifact in a form understandable to all of them,
- the domain-orientation enhances the conversation with the materials of a design situation,
- simulations let stakeholders experience behavior and see the consequences of their assumptions,
- critiquing signals violation of rules and controversial design decisions, and
- argumentation provides the argument behind rules and artifacts to empower designers to disagree.

A Process Model for DODES. Our process model (Fischer, McCall et al. 1994) for continual development of design environments from an initial seed through iterations of growth and reseeding is illustrated in Figure 1:

- the *seeding* process, in which domain designers and environment developers work together to instantiate a domain-oriented design environment seeded with domain knowledge.
- the *evolutionary growth* process, in which domain designers add information to the seed as they use it to create design artifacts.
- the *reseeding* process, in which environment developers help domain designers to reorganize and reformulate information so it can be reused to support future design tasks.

During seeding, environment developers and domain designers collaborate to create a design environment seed. During evolutionary growth, domain designers create artifacts that add new domain knowledge to the seed. In the reseeding phase, environment developers again collaborate with domain designers to organize, formalize, and generalize new knowledge.

The top half shows how DODEs are created through a collaboration between the environment developers and domain designers. The bottom half shows the use of a DODE in the creation of an individual artifact through a collaboration between domain designers and clients. The use of a DODE contributes to its evolution; i.e.,

new knowledge, developed in the context of an individual project, is incorporated into the evolving design environment.

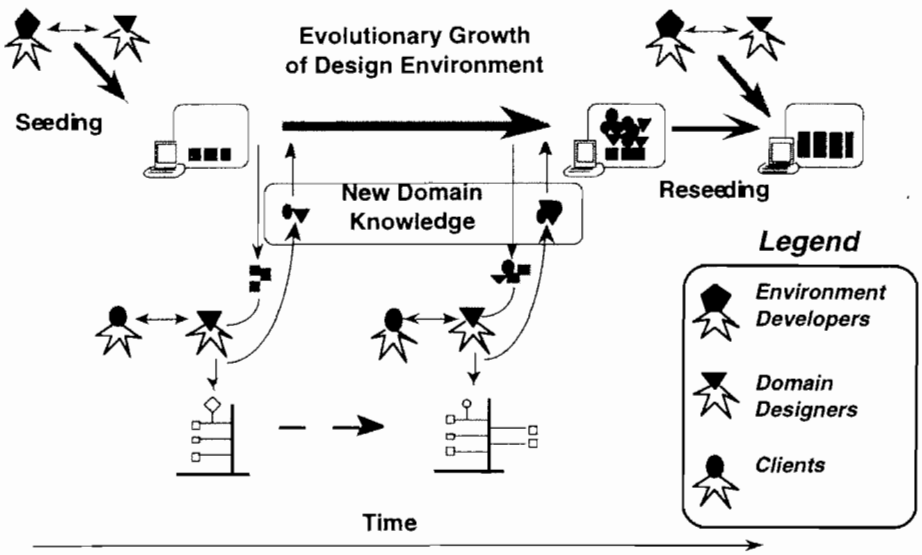

Figure 1: The "Seeding - Evolutionary Growth - Reseeding (SER)" Model:
A Process Model for the Development and Evolution of DODES

The Seeding Process. A seed is built by customizing the domain-independent design environment architecture to a particular domain through a process of knowledge construction. Although the goal is to construct as much knowledge as possible during seed-building, for complex and changing domains complete coverage is not possible. Therefore, the seed is explicitly designed to capture design knowledge during use (Girgensohn 1992).

Domain designers must participate in the seeding process because they have the expertise to determine when a seed can support their work practice. Rather than expecting designers to articulate precise and complete system requirements prior to seed building, we view seed building as knowledge construction (in which knowledge structures and access methods are collaboratively designed and built) rather than as knowledge acquisition (in which knowledge is transferred from an expert to a knowledge engineer and finally expressed in formal rules and procedures). New seed requirements are elicited by constructing and evaluating domain-oriented knowledge structures.

Evolutionary Growth Through Use. During the use phase, each design task has the potential to add to the knowledge contained in the system. New construction kit parts and rules are required to support design in rapidly changing domains. Issue-based information in the seed can also be augmented by each design task as alternative approaches to problems are discovered and recorded. The information accumulated in the information space during this phase is mostly informal because designers either cannot formalize new knowledge or they do not want to be distracted from their design task.

Reseeding. Acquiring design knowledge is of little benefit unless it can be delivered to designers when it is relevant. Periodically, the growing information space must be structured, generalized, and formalized in a reseeding process, which increases the computational support the system is able to provide to designers (Shipman 1993).

The task of reseeding involves environment developers working with domain designers. After a period of use, the information space can be a jumble of annotations, partial designs, and discussions mixed in with the original seed and any modifications performed by the domain designers. To make this information useful, the environment developers work with the domain designers in a process of organizing, generalizing, and formalizing the new information and updating the initial seed.

4. Impact

Knowledge Acquisition versus Domain Construction. Domain-oriented systems offer many potential benefits for users such as more intuitive interfaces, better task support, and knowledge-based assistance. A key challenge for system developers constructing domain-oriented systems is determining what the current domain is and what the future domain should be; i.e. what entities should the system embody and how should they be represented. Determining an appropriate domain model is challenging because domains are not static entities that objectively exist, but instead they are dynamic entities that are constructed over time by a community of practice. New software development models and new computational tools are needed that support these communities to create initial models of the domain and to evolve these models over time to meet changing needs and practices.

The domain modeling approach assumes there exists a common conceptual model of the domain shared by all practitioners and the problem is simply to identify what this model is and codify it. This approach falls into the category of first generation design methods that assume a strict separation between design, use, and implementation phases. In the design phase, these approaches try to identify the domain model through knowledge acquisition practices such as interviewing selected domain experts. As such, these approaches do not acknowledge the situated (Suchman 1987) and tacit (Polanyi 1966) nature of professional expertise. In the implementation phase, these approaches adopt an engineering perspective in that they emphasize domain model representations rooted in computational formalisms rather than representations rooted in work practices. The result is domain models that cannot be inspected or modified by domain practitioners to reflect changing work practices.

The domain construction approach address these shortcomings by explicitly acknowledging that shared domain models do not de facto exist but instead are socially constructed over time by communities of practice. As such, this approach emphasizes the prominent role that domain practitioners must play in constructing an initial model of the domain rooted in work practices and evolving this model over time to suit their changing needs. Computational tools and new models of software development are needed that enable domain practitioners to fully participate in the design and evolution of their domain-oriented system.

The predominant activity in designing complex systems is the participants teaching and instructing each other (Greenbaum and Kyng 1991). Complex problems require more knowledge than any single person possesses, making communication and collaboration among all the involved stakeholders a necessity. Domain designers understand the practice and environment developers know the technology. None of these carriers of

knowledge can guarantee that their knowledge is superior or more complete compared to other people's knowledge. The goal of the seeding process is to activate as much knowledge from as many stakeholders as possible taking into account that system requirements are not so much analytically specified as they are collaboratively evolved through an iterative process of consultation between all stakeholders. This iterative process of consultation requires representations (such as prototypes, mock-ups, sketches, scenarios, or use situations that can be experienced) which are intelligible and can serve as "objects-to-think-with" for all involved stakeholders.

This iterative process is important to support the interrelationship between problem framing and problem solving documented by many design methodologists (Schoen 1983). They argue convincingly that (1) one cannot gather information meaningfully unless one has understood the problem, but one cannot understand the problem without information about it; and (2) professional practice has at least as much to do with defining a problem as with solving a problem. New requirements emerge during development because they cannot be identified until portions of the system have been designed or implemented. The conceptual structures underlying complex software systems are too complicated to be specified accurately in advance, and too complex to be built faultlessly. Specification and implementation have to co-evolve, requiring the owners of the problems to be present in the development.

Supporting Work Practices. The development and evolution (see Figure 1) of computational artifacts requires the collaboration between system developers and domain practitioners raising the following issues:

(1) What software development methodologies are best suited to support the process of system developers and domain practitioners engaging in collaborative domain construction and evolution?

(2) What tools are needed to assist ongoing developer-practitioner communication during system development and extension?

(3) What system architectures are required for developers to create domain-oriented systems in an iterative, yet resource-efficient manner?

(4) What mechanisms can these architectures provide to support domain practitioners to incrementally enrich their tools with computationally interpretable design languages without developer support?

Integrating Working and Learning. Critiquing naturally offers learning opportunities to users. A critic offers alternative perspectives on what a user has done by pointing out potential problems, suggesting additional relevant information to consider, and making reasonable guesses to fill in low-level details. Critiquing brings not just the immediate benefit of improving the current product, but it exposes users to new knowledge.

Critiquing is one way to support learning on demand (Fischer 1991)— learning is contextualized to the task at hand. Critiquing provides support for users who are engaged in understanding, formulating and solving their own problems, rather than solving the problems which the system posed (this is a fundamental difference which sets our approaches apart from intelligent tutoring systems).

Using systems for real work will lead to situations in which the system lacks the knowledge and understanding for a specific problem. End-user modifiability support is needed to allow users to integrate their knowledge into the system (causing the evolutionary growth as illustrated in Figure 1).

Mimicking versus Complementing. Human-computer collaboration can be conceptualized and operationalized from two different perspectives (Terveen 1995):

(1) the *human emulation* approach where the basic assumption is to endow computers with human-like abilities (such as natural language, speech, etc.), and

(2) the *human complementary* approach which exploits the asymmetric abilities of humans and computers and tries to identify the most desirable and most adequate role distributions between humans and computational environments (Billings 1991).

Our work is grounded in the complementary approach. We have explored (depending on the task, the interest and the people involved) different styles of collaboration.

5. Conclusions

The challenge for AI is to augment and empower human intelligence by creating collaborative support systems in which humans can concentrate on the tasks which they can do best. We need systems that are convivial, domain-oriented, open, evolvable and provide a shared understanding for mutual learning and mutual understanding of all stakeholders involved.

Acknowledgments. The author would like to thank the members of the Center for Lifelong Learning and Design at the University of Colorado who have made major contributions to the conceptual framework and systems described in this paper. The research was supported by (1) the National Science Foundation, Grant RED-9253425, (2) the ARPA HCI program, Grant N66001-94-C-6038, (3) Nynex, Science and Technology Center, (4) Software Research Associates (SRA), and (5) PFU.

References

Billings, C. E. (1991). Human-Centered Aircraft Automation: A Concept and Guidelines. NASA Ames Research Center, NASA Technical Memorandum, No. 103885. Moffett Field, CA,

Bobrow, D. G. (1991). Dimensions of Interaction. AI Magazine. 64-80.

Carbonell, J. R. (1970). Mixed-Initiative Man-Computer Instructional Dialogues. BBN Report No. 1971.

Engelbart, D. C. and W. K. English (1968). A Research Center for Augmenting Human Intellect. Proceedings of the AFIPS Fall Joint Computer Conference. Washington, D.C., The Thompson Book Company. 395-410.

Fischer, G. (1991). Supporting Learning on Demand with Design Environments. Proceedings of the International Conference on the Learning Sciences 1991 (Evanston, IL). Charlottesville, VA, Association for the Advancement of Computing in Education. 165-172.

Fischer, G., J. Grudin, et al. (1992). Supporting Indirect, Collaborative Design with Integrated Knowledge-Based Design Environments. Human Computer Interaction, Special Issue on Computer Supported Cooperative Work. 281-314.

Fischer, G. and A. C. Lemke (1988). Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication. Human-Computer Interaction. 179-222.

Fischer, G., A. C. Lemke, et al. (1991). The Role of Critiquing in Cooperative Problem Solving. ACM Transactions on Information Systems. 123-151.

Fischer, G., R. McCall, et al. (1994). Seeding, Evolutionary Growth and Reseeding: Supporting Incremental Development of Design Environments. Human Factors in Computing Systems, CHI'94 Conference Proceedings (Boston, MA). 292-298.

Fischer, G., K. Nakakoji, et al. (1993). Facilitating Collaborative Design through Representations of Context and Intent. Proceedings of AAAI-93 Workshop, AI in Collaborative Design (Washington DC). 293-312.

Fischer, G. and C. Stevens (1991). Information Access in Complex, Poorly Structured Information Spaces. Human Factors in Computing Systems, CHI'91 Conference Proceedings (New Orleans, LA). New York, 63-70.

Girgensohn, A. (1992). End-User Modifiability in Knowledge-Based Design Environments. Ph.D. Dissertation, TechReport CU-CS-595-92. University of Colorado. Boulder, CO.

Greenbaum, J. and M. Kyng (1991). Design at Work: Cooperative Design of Computer Systems. Hillsdale, NJ, Lawrence Erlbaum Associates.

Nakakoji, K. and G. Fischer (1995). Intertwining Knowledge Delivery and Elicitation: A Process Model for Human-Computer Collaboration in Design. Knowledge-Based Systems Journal. Vol. 8, Issue 2-3, 1995. 94-104

Norman, D. A. (1993). Things That Make Us Smart. Reading, MA, Addison-Wesley Publishing Company.

Polanyi, M. (1966). The Tacit Dimension. Garden City, NY, Doubleday.

Repenning, A. and T. Sumner (1992). Using Agentsheets to Create a Voice Dialog Design Environment. Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing. ACM Press. 1199-1207.

Schoen, D. A. (1983). The Reflective Practitioner: How Professionals Think in Action. New York, Basic Books.

Shipman, F. (1993). Supporting Knowledge-Base Evolution with Incremental Formalization. Ph.D. Dissertation, TechReport CU-CS-658-93. University of Colorado. Boulder, CO.

Suchman, L. A. (1987). Plans and Situated Actions. Cambridge, UK, Cambridge University Press.

Terveen, L. G. (1995). An Overivew of Human-Computer Collaboration. Knowledge-Based Systems Journal. Vol. 8, Issue 2-3, 1995