

DIALOGUE MODELS OF KNOWLEDGE INTENSIVE ASSISTANCE IN SHAPE CONCEPTUALISATION

Jouke Verlinden, Imre Horváth and Peter de Jager

Keywords: Man-machine interaction, conceptual design, computer-aided shape conceptualisation, knowledge-based systems.

1 Introduction

As once noticed in 1963 by Coons, the computer should be able to support design as a full blown assistant, a partner of the designer that actively collaborates [1]. Since the initial introduction of artificial intelligence to the field of computer-aided design and user interface research, many applications have evolved. The advent of wizards, mixed-initiative systems, and decision support systems all made a contribution to the collaborative assistance once mentioned by Coons. Furthermore, new ways to communicate with the computer have emerged, including gestural and speech-based interfaces. The combination of these offer possibilities towards an expressive dialog, similar to our natural human-to-human communication.

In spite of all these advances, the present generation of Computer Aided Design systems offers only a fragment of these capabilities. Although the act of drawing certainly gained an increased level of efficiency and precision, designing itself faces serious limitations in the expressiveness. We assume that the development of natural dialogue and knowledgeable systems will lead to actively collaborating partners as conversational design systems.

This literature study gives an overview of knowledge intensive design systems and the technology involved, in such depth that we can define a framework for knowledge-intensive assistance. In surveying, we have focused on conference and journal articles within the field of Computer-Human Interaction and the Design Sciences. This research has been performed as part of the Integrated Concept Advancement (ICA) project, at the Delft University of Technology.

2 Assistance in Design: supporting the user by anticipating

The following chapter classifies the different modes of assistance, based on an extensive review of Computer-Human Interaction (CHI) related publications. The main issue is to investigate the number of ways to assist the user, as presented in the CHI community. This included literature covering ACM's conferences and symposia of CHI, IUI, DIS, and UIST (respectively Computer-Human Interaction, Intelligent User Interfaces, Designing Interactive Systems, and User Interface Software Technologies). Related journals include Knowledge-Based Systems, User Modelling and User-Adapted Interaction, and the Knowledge

Engineering Review. Over 250 papers were scanned, key literature is referred to in the next sections.

One basic aspect of assistance is its anticipating behaviour, often called 'mixed initiative'. This denotes the capacity of the system to be responsive to a particular situation and to take the initiative to resolve problems - or to teach/give advice- when necessary. In the past years several publications on the exact definition of 'initiative' has been written. This literature has its roots in computational linguistics, the area of spoken dialogue. A broad definition of initiative is defined in [2] as 'control over the flow of conversation'. Flow specifies the movement of the conversation, through a subject or a series of subjects. In [3], a distinction is made between 'task initiative' (plan) and 'dialogue initiative' (conversational).

In classifying the kinds of assistance in communicating designs, we distinguished three types of anticipation (i) model based assistance, (ii) tool based assistance, and (iii) dialogue based assistance. The first type continuously studies the (geometric) model and provides help based on the current state of the design. The second focuses on managing the modelling environment and tries to offer a set of tools that seem to be appropriate at a certain design activity. The last is focused on the interaction between user and application and suggests more effective strategies to finish a task. In the following sections we will discuss characteristics for each of these types of assistance, based on some characteristic examples.

3 Model-based assistance: Design Critiquing

Design critiquing can assist the designer by evaluating the current state of a model based on a rulebase. It encapsulates expert knowledge, which might lead to improved knowledge transfer and quality, while its critique can include procedures to enhance a certain design. This employment of "expert systems" can happen on the fly or in batch processes. As stated in [4], this type of assistance fits in Schön's "reflection-in-action", in which the cycle of synthesis and reflection iterates in short cycles [5].

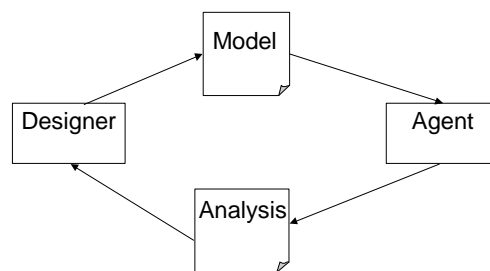


Figure 1. Dialogue between designer and agent in Design Critiquing

One of the earliest examples is a system called HYDRA [6], which focused on kitchen design. It integrated a graphical layout editor with a critiquing environment, able to analyse the drawings and give various types of critique. Specifically, HYDRA included generic, specific and interpretative critiques. Generic critiques are always valid, specific ones within a certain domain, and interpretative critique is based on the designer's input. Each critique was based on a set of rules that were constantly monitored while the designer was interacting with the layout editor.

An large number of critiquing aspects exist, including Correctness, Completeness, Consistency, Optimisation, Evolvability, Presentation, Tool, Experiential, and Organisational [7]. Design critique algorithms use the model representation and scan for the existence of specific conditions, more or less formally defined in a rule base. The basics are conditions and a critique message that will be displayed. Furthermore, an automated procedure or script might be provided to 'cure' a specific error [7]. Such systems are often agent-based, in which each agent scans the design for a specific type of critique. A generic life cycle for critique was proposed by Robbins, called ADAIR. First, a number of agents are activated, then these try to detect opportunities to give criticism. If they detect such opportunities, they will advise the designer. When he or she thinks the critique is correct, the situation will be improved and finally, the decisions are recorded for later use. There might be quite elaborate improvement scenarios, for example the advise and improvement dialog in [8]. A model-based critiquing system for exploratory data analysis is presented, in which statistical operations like regression and clustering are proposed by the system interactively.

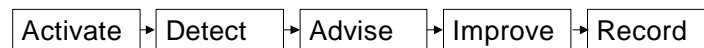


Figure 2. The ADAIR critiquing cycle

Issues concerning design critique include:

Right information at the right time. Although critique systems allow a means of presenting a dynamic evaluation of the model, it is very difficult to determine what information to convey to the designer without blocking the design process itself. The literature often refers to “the right information at the right time”. This strongly depends on the domain, the task, and the individual designer.

Presentation. Displaying the critique messages is a crucial aspect. Most of the critiquing systems fail to do so in an appropriate way and generate a long list of messages that are difficult to handle. An exception to this rule is the dynamic to-do list found in ArgoUML [7]; critiques are dynamically inserted (and removed) to lists with different priorities by a number of critiquing agents.

Explanation. Presenting the user critique on a model might not be enough. The designer might be curious why a particular remark is given ("why do you suggest ...".) This is an issue while the rules are designed.

Inconsistent critique. Some argue that the variety of comments should be unified and contradicting critiques should be harmonised [9]. Others allow this to happen and give the use tools to enable/disable or prioritise certain design critiques.

Detection. Specifying conditions for critique systems might be complex, as the (geometric) model does not contain all information necessary to use an expert system.

4 Tool-based assistance: context switching agents

Another class of assistance is the dynamic adoption of the working environment based on task contexts. As design covers a broad spectrum of activities, it is improbable that one generic interface will be sufficient to support design. Different tools or settings might be used when

different aspects of the design are covered. For example, 2D sketching with rectangles segments might be sufficient to design a floor plan, while detailed design might require a multitude of objects and a finer grid setting. It has been argued by Card and others that humans will segment their physical working environment in the context of the main tasks [10]. To enable context switching, the Rooms project and its 3D Rooms successor can be seen as the first implementations that offered context switching [11]. Similar to a virtual desktop, the user is able to define a number of contexts in which both applications and documents reside. Of course, applications can be shared among the rooms. Typical applications of these systems are found in workflow management systems, which the contexts are predefined based on business processes.

In the case of automatic context switching, a software agent tries to recognise the current design context and then activates its corresponding tools. The "the Right Tool at the Right Moment" project [12] focused on the domain of freehand sketching in architecture. Empirical studies showed that architects share a rather fixed set of contexts in the preliminary phases of design, namely bubble diagram, section, floor plan, numbers and 3D drawing. The main characteristics of these sketch techniques were analysed and captured in a rule base, to be employed in real time recognition of pen-based input. Finally, each context was associated with a number of 'useful' helper applications - (respectively spatial arrangement, image query, view analysis, calculation, and mass study) to be started and stopped by the context-switching agent.

A key characteristic of this class is that the interface is adapted without automatically executing commands, not does it assess the quality of the existing model. It simply tries to fit the working environment as close to the user's task context, still allowing the user to choose what to do. Furthermore, this type of assistance might be reduced to a small part of the working environment, e.g. flexible menu systems [13].

Issues concerning context-switching agents include:

Context definition. The definition of the contexts and its associated tools/settings can be performed in a multitude of ways. First, this might be done by the user or by the system (and its supplier). Second, the definition might dynamically adapt or require explicit changes.

Exclusion. With an ill-defined context, the environment will not offer the appropriate number of tools. It should then be possible to select the correct tool via workaround techniques. This yields for a number of interface techniques that remain static and allow full access to all the application's capabilities.

Initiation. The act of context switching can be initiated by three different actors, i) the user, ii) a personalised agent, iii) the application. The application uses domain-based triggers whereas the agent is more intimately coupled with the user's behaviour.

5 Dialogue-based assistance: programming by example

Programming by Example (PBE, sometimes called programming by demonstration) tries to assist the user in tedious tasks, in essence by generating, suggesting and executing macros [14]. The PBE software agent watches the dialogue between user and application and, based on the interaction history and some more or less advanced inference algorithm, tries to offer the user compound actions. For example, based on a repetitive pattern of copy-paste actions

by the user, the system might infer that she wants to sort the elements in a list. The agent might suggest this to the user and ask to finish the task automatically. The promise of PBE is to enhance the human-computer dialogue by means of creating macros without having to use programming language. This would allow non-programmers (kids, novices, but also designers) to create individualised (and when necessary parameterised) procedures that will lower the burden of pointing and clicking.

An early example of PBE is Eager [15], an Apple Macintosh-based software agent that monitors the behaviour of the user and uses world knowledge (like numeric sequences and days of the week) in order to find patterns of repetitive behaviours. Eager shows his prediction of the next action to the user during monitoring by highlighting it in green. This allows the user to become aware of the monitoring process and to give feedback whether Eager is predicting the right pattern. When Eager is confident that it found the correct pattern, it will show a small dialog box with the option to let Eager take over control. The application of programming by example is quite easily extended towards computer aided design, as defining (and refining) a model involves a lot of compound actions that are used as "recipes". Patrick Girard implemented an exploratory design system [16] and is still involved in this research [17].

The algorithms used in programming by example are basically pattern recognisers. The stream of user input events is continuously studied and generalised to wider applicability. This process of recognition can be bottom-up (like the repetitive action recognition of Eager) or top-down. Some recent advancements in this area are the adoption of 'Version Spaces' and 'Inductive Logic Programming' [18], and algorithms including 'Concept Learning', 'Decision trees' [19].

The most important issues in developing Programming by Example agents are:

Explicit versus implicit training. When a PBE agent has to be activated and set in training mode before offering examples, the notion of collaboration is different than when the agent is always monitoring (like Eager). Explicit training enables the refinement of a previously recorded behaviour (for example by providing more examples or counter-examples)

Generalisation/learning algorithms. The process of generalisation is one of the hardest. As mentioned in [20], the sophistication of the system needs to be balanced with the user's expectations. Assuming too much intelligence will lead to misconceptions that might lead to disastrous effects - e.g. based on prior knowledge, the user thinks that demonstrating the deletion of one file will remove all files in that particular directory; instead, the system will simply remove all files from the hard disk. Myers presents a variety of approaches, from simple rule-based inferencing to more advanced inferencing algorithms. Whereas the first are easier to develop and do not assume too much 'intelligence' to the system's capabilities, the second is capable to deal with complicated behaviour. As a standard reference, Mitchell's book on machine learning is often used [21]. This book describes the basics of AI machine learning and different algorithms.

Felicity conditions. In order to capture the right patterns during learning, only a certain amount of "noise" is allowed. Van Lehn introduces the notion of so-called "felicity conditions", necessary to successfully instruct a system: a) show all steps, b) do all steps correctly, c) make invisible objects and relations visible, and d) introduce at most one new branch per lesson [22]. Although this is a general list, it is often referred to as a starting point.

Initiative. Some systems offer the learnt macros as buttons or menus, allowing the user to activate the behaviour when he thinks it applies. Although this enables the user to manage and filter the behaviours that are necessary at a certain point in time, it leaves the initiative up to the user to recognise one of the existing macros will be applicable and activate these.

Visualisation. Some behaviours might need to be corrected or expanded. One of the most difficult issues is then to visualise the existing behaviour without resorting to some kind of programming language. A common visualisation means in rule-based systems is displaying before-after rules in some iconic forms. For example, Stagecast Creator uses this approach [23]. An attempt to generate written text based on the rules was done in Mondrian [24]. Obviously, these systems become more complex when the learning algorithm is more sophisticated. When the learning algorithm is based on multiple examples or conditional branches, visualisation or natural language generation becomes a big challenge. Similar to Mondrian, the Marquise [25] system offers a textual representation to the user and allows editing. In this system, some help is offered by employing drop-down listboxes instead of free-form text editing. A more advanced macro editing facility was part of Pursuit [26], which presented an interactive graphical account of the examples and its branches.

6 Evaluation

The models mentioned in this paper provide an initial framework in building knowledgeable partners. Each of them have their merits, all can play a considerable role during conceptual design. A further assessment is made in the following table, in which the models are compared in respect to implementation effort, algorithms, initiation methods, shape design, and the impact of the assistance on natural interfaces.

Table 1. Evaluation of the assistance classes

	Model	Tool	Dialogue
Opportunities in conceptual shape design	Quality Offers real-time evaluation of the design, from design rationale to integration of quantitative evaluation systems.	Opportunity Diminishes the cognitive load of knowing the tools, allows fast selection (and learning) of the intended action.	Efficiency Offers automatic execution of repetitive actions, which are paramount in the definition of shapes during conceptual design.
Implementation effort	Considerate, depends on underlying representation	Average, depends on the desired complexity of initiating switching	Considerate, depending on learning algorithms
Preferred algorithm	Rule/Case-based reasoning	Deterministic or statistic	Concept Learning
Possible Weaknesses	Formalised knowledge might be difficult to support all types of design (specific domains might be feasible).	Difficult to determine shape design contexts in advance. This might differ per domain or individual.	The non-deterministic character of design makes it hard to generalise programs with little overhead.
Impact on natural interfaces	Independent, can possibly augment the natural interface by repairing situations (without violating the designer's intent)	The natural dialogue influences the definition of contexts. Conversely, the tool adaptation might empower both novices and experts.	As the natural interface defines the dialogue, there are strong dependencies. Can alleviate the dialogue.

7 Conclusion

The introduction of new interaction techniques yields for new applications of knowledge-intensive systems. This is specifically the case in conceptual design, in which shape design is tightly integrated with other kinds of knowledge. In developing knowledge-intensive assistance schemes, the dialogue component is essential in defining interoperability between man and machine. The three presented classes of assistance are all to be included in a new shape design systems that offer natural means of communication in the conceptual phase of design. Our research will be an expansion of model-based assistance for augmenting the natural dialogue, specifically to repair incorrect interpretation of multimodal input.

References

- [1] Coons, S.A., "An outline of the requirements for a computer aided design system." In AFIPS Spring Joint Computer Conference 23, 1963, pp. 299-304.
- [2] Cohen, R., Allaby, C., Cumbaa, C., Fitzgerald, M., Ho, K., Hui, B., Latulipe, C., Lu, F., Moussa, N., Pooley, D., Qian, A., Siddiqi, S. , " What is Initiative?", User Modeling and User Adapted Interaction (Kluwer Academic Publishers Group) 1998 - volume 8(3-4), pp. 171 - 214.
- [3] Chu-Carroll, J., Brown, M., An Evidential Model for Tracking Initiative in Collaborative Dialogue Interactions", User Modeling and User-Adapted Interaction 1998 - volume 8(3-4), pp. 215 -254.
- [4] Nakakoji, K, Yamamoto, Y., Suzuki, T., Takada, S., Gross, M., "From critiquing to representational talkback: Computer support for revealing features in design", Knowledge-Based Systems (Elsevier), 1998, vol 11, pp. 457-468.
- [5] Schön, D.A., "The Reflective Practitioner: How Professionals Think in Action", Basic Books, New York, 1983.
- [6] Fisher, G., Nakakoji, K., Ostwald, J., Stahl, G., Sumner, T., "Embedding critics in design environments", The Knowledge Engineering Review Journal, Special issue on expert critiquing, vol 8(4), 1993, pp. 285-307.
- [7] Robbins, J., Redmiles, D., "Software architecture critics in the Argo design environment", Knowledge-Based Systems (Elsevier), vol 11, 1998, pp. 47-60.
- [8] Gupta, S., Regli, W., Nau. D. "Integrating DFM with CAD through design critiquing" Concurrent Engineering: Research and Applications, vol 2(2), 1994.
- [9] St Amant, R., Cohen, P., "Interaction with a mixed-initiative system for exploratory data analysis" Knowledge-Based Systems (Elsevier), vol 10, 1998, pp. 265-273.
- [10] Henderson, D. A., and Card, S., "Rooms: The use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface" ACM Transactions on Graphics, Vol 5(3), 1986, pp. 211-243.
- [11] Card, S. K., G. G. Robertson, and J. D. Mackinlay., "The information visualizer, an information workspace" in Proceedings of CHI '91 (New Orleans, Louisiana), 1991, pp. 181-188.
- [12] Do, E. Y. -L., "The Right Tool at the Right Time - Investigation of Freehand Drawing as an Interface to Knowledge Based Design Tools" PhD Thesis, College of Architecture, Atlanta, Georgia Institute of Technology, 1998.

- [13] Kühme, T. Malinowski, U., Foley, J.D., " Facilitating Interactive Tool Selection by Adaptive Prompting" Adjunct Proceedings of INTERACT'93, 1993, pp. 149-150.
- [14] Lieberman, H., Introductory on programming by example, Communications of the ACM, Vol.43(3), 2000, pp. 72 - 74
- [15] Cypher, A., "Eager: programming repetitive tasks by example", in Proceedings of CHI'91, 1991,, pp. 33-39.
- [16] Girard, P., "Environnement de programmation pour non programmeurs et Paramétrage en Conception Assistée par Ordinateur: Le Système Like", Thèse de Doctorat d'Informatique de l'Université de Poitiers, Novembre 1992.
- [17] Girard, P., Pierra, P., Potier, J. "Customizing by Demonstration Generic Systems to Specific Tasks" UI4ALL, 3rd ERCIM Workshop on User Interfaces for All, Ortrrott, France. ERCIM & INRIA Lorraine, 1997, pp. 189-196.
- [18] Lau, T., Weld, D., "Programming by Demonstration: An Inductive Learning Formulation", Proceedings of Intelligent User Interfaces 1999, Redondo Beach (USA), 1999, pp. 145-152.
- [19] Ruvini, J., Dony, C., "APE: Learning User's Habits to Automate Repetitive Tasks" in Proceedings of the Intelligent User Interfaces Conference 2000, New Orleans (USA), 2000, pp. 229-232.
- [20] Myers, B., McDaniel, R., Wolber, D. "Intelligence in Demonstrational Interfaces" Communications of the ACM, March 2000, Vol 43, No. 3, pp 82-89.
- [21] Mitchell, T., "Machine Learning", McGraw Hill, ISBN 0070428077, 1997.
- [22] van Lehn K., "Felicity Conditions for Human Skill Acquisition: Validating an AI-Based Theory" Research Report No. CIS-21, Xerox Palo Alto Research Center, 1983.
- [23] Smith, D.C., "Building Personal Tools by Programming", in Communications of the ACM, Vol. 43(8), 2000, pp. 92-95.
- [24] Myers B., McDaniel, R. and Kosbie, D., "Marquise: Creating Complete User Interfaces by Demonstration," in Proceedings CHI'93(Amsterdam, The Netherlands), 1993, pp. 293-300.
- [25] Lieberman, H., "Mondrian: A Teachable Graphical Editor", ". In Cypher A, ed. Watch What I Do: Programming by Demonstration, MIT Press, Cambridge MA, 1993, pp. 340-358.
- [26] Modugno, F., Myers, B., "Visual Programming in a Visual Shell: A Unified Approach", J. Vis. Lang. Comput., 8, 5/6 (Oct./Dec. 1997), pp. 276-308.

Jouke C. Verlinden

Delft University of Technology, Faculty of Design, Engineering and production, subfaculty Industrial Design Engineering, Jaffalaan 9, 2628 BX, Delft, the Netherlands,
 Tel +31-15-2789 321, Fax: +31 15 278 1839, E-mail: j.c.verlinden@io.tudelft.nl,
<http://www.io.tudelft.nl/research/ica>