Dynamic Design Specification with Chunking

Mihaly Lenart*
Department of Architecture, University of Kassel, Henschelstr. 2
34109 Kassel, Germany; lenartm@scs.fiu.edu

Ana Pasztor†
School of Computer Science, Florida International University
University Park, Miami, Fl. 33199; pasztora@scs.fiu.edu

Abstract

This paper is concerned with problems inherent in the informal specification of design. It describes a domain-independent, generic knowledge-based requirement elicitation tool for design, called DDSC (Design Specification with Chunking). Design specification with DDSC is a dynamic process evolving through continuous participant interaction. The elicitation process is based on the structure of human cognition through the use of recognized cognitive categories.

DDSC takes as input an initial, informal requirement specification for a design object, as well as an extended generic ontology of the design process, and generates a complete, unambiguous, consistent, and implementable requirement specification of the desired design object. It does so by "walking" the Designer and the User Agents through the design process. During this "walk-through," requirements are *chunked* down with the help of a Dialogue Agent and Chunking Agents into implementable bits.

DDSC is a participatory design tool that can be used in any application domain. This sets it apart from most of the existing participatory design tools, which are aimed at system design.

1 Introduction

Design automation is impossible without addressing the problems inherent in the *informal* specification of design.

Informal specifications are the well-known and commonly used specifications in a natural language, such as English, or rather a subset of a natural language used by a given profession. Informal specifications are also codes, guidelines, and regulations as they can be found in the professional literature. Each specification starts out informally. At each stage of the specification process goals, parameters, and constraints can be clarified further indefinitely by elicitation of more and more details. For obvious reasons, however, the elicitation process has to get to a halt at some stage.

Each specification has two parts: a hidden, common knowledge part and an explicit part describing relevant features and/or requirements. To what extent the explicit part spells out details, depends on the purpose of the specification. While less details leave room for creativity, at some point in the design process the specification has to become *determinate* in order for the designer to be able to recover the design object from the verbal specification. In fact, given sufficient information, the specification process might lead to *implementable specifications* or even to concrete solutions in which case we talk about "executable specification" (c.f. [Davis, 1982], [Partridge and Wilks, 1987], [Rattray *et al.*, 1991], [Lenart *et al.*, 1994]). Usually, however, there is a huge *gap* between informal specification and design implementation. The author's **goal** is to *minimize* this gap and support the automation of this process.

For this, they are developing a domain independent, generic knowledge-based (KB) requirement elicitation (RE) tool for engineering design, called DDSC (Dynamic Design Specification with Chunking). Capturing design requirements is often done by post hoc knowledge acquisition, which in general is inadequate and difficult [Kellogg *et al.*, 1989]. In the here described approach, however, design specification is a *dynamic* process evolving through a *direct* interaction between all participants of the design process. Similarly to [Lancaster *et al.*, 1989], the elicitation process is based on the structure of human cognition through the use of recognized cognitive categories as defined in [Cameron-Bandler and Lebeau, 1985]. DDSC is a participatory design tool that can be use in any application domain. This sets it apart from most of the existing participatory design tools aimed at system design [Comm. of the ACM, 1995].

DDSC is based on an extended blackboard concept similar to the one in [Harris and Harvey, 1994], employing several (not necessarily computer-based) intelligent agents. DDSC takes as input an initial, informal requirement specification for a design object, as well as an extended generic ontology of the design process (see next section for definition) and generates a *complete, unambiguous, consistent*, and *implementable* requirement specification of the desired design object. It

*This work has been sopported by the German Reseach Association(DFG), grant no.: LE 442/10-1

†This work has been sopported by the National Science Foundation, grant no.: IRI-9630699

does so by "walking" the Designer and the User Agents[1] trough the design process. During this "walk-through," specifications are stepwise modified by both the User and the Designer Agents, respectively, and requirements are *chunked* down with the help of a Dialogue Agent and Chunking Agents into implementable bits. Implementable specifications correspond to the *atomic objects* of [Alexander *et al.*, 1988] or *surrogates* in data base terminology [Codd, 1979]. A session with DDSC implementation results in the generation of a tree structure representing various specification refinement levels, the specification chunking process, as well as the various design decisions made by the User Agent during the "walk-through".

2 Solving Problems inherent in Informal Specifications

Informal specifications are *incomplete, ambiguous* and *inconsistent.* The following is an individual description of each of these problems and how the DDSC RE system addresses them.

2.1 Incompleteness

When we "future pace," i.e. we pack our luggage to travel and we want to make sure that nothing will be missing, then we mentally "walk" through a day or situations on the trip and see, hear or feel what we will wear, need or use. Sometimes we daydream about the trip and suddenly we remember to pack or buy something for the trip. Similarly, "walking" the user with the designer through the design process helps the user to *find requirements* which s/he did not think of initially. In DDSC, the designer "walks" the user through the design process soliciting missing requirements in this fashion.

For example, let us assume that the user wants the designer to design a chair with the requirement that it be easy to clean. "Walking" through the design process with the user, the designer explores the possibility of an upholstered chair with nylon cover in order to satisfy the requirement "easy to clean". Only at this point does the user remember to tell the designer that s/he is allergic to nylon.

The "walk-through" is carried out iteratively: First the designer mentally "walks" the user through the design process until they get to a point at which there are too many uncertainities to continue. Here the designer proceeds with the actual design to satisfy existing requirements. Once the uncertainities are resolved, the mental "walk-through" resumes, etc.

In a case study, we refer to as "The Heart Project" and whose aim was to develop a fluid mechanical model of the heart for testing artificial valves, a customer's requirement was to be able to change the stroke volume on the fly. The designer team satisfied this requirement by designing an adjustable pivot point on the crank arm connecting two parallel shafts — one attached to the motor and the other to the piston. "Walking" with the user and the designers through the process, the authors, facilitating RE-functions (as the Dialog Agent), noticed that after changing the position of the position of the pivot point, all information about the previous position is lost. At this point, the user realized that he needs this information for keeping track with and adjusting the stroke volume. This new requirement forced the designer to add a monitoring device for registering the position of the pivot point.

2.2 Ambiguity

Verbal specifications are often *vague.* The user might have a clear mental image of the design object, but be unable to *describe* it verbally with the same clarity. In addition, verbal descriptions often have *different meaning* for different people due to their individual experiences. This is particularly true if the *chunk size* of a requirement is *too big.* During the "walk-through", the *user* and the *designer* interact to *chunk down* these requirements into pieces which are unambiguous and therefore *implementable.* (A specification is called implementable if it completely and unambiguously describes a design object such that it can be generated according to this description to the full satisfaction of the user.) The idea is to offer the users ways of refining their original large chunk requirements in terms of various elements/variables/criteria involved in the design process.

The DDSC system helps to elicit more *determinate* representations of the design object with the goal of narrowing down the choice of images of this object in the designer's mind. Johnson-Laird [Johnson-Laird, 1983] (p. 160) describes experiments proving that people tend to form a mental model from a determinate description (specification), but fail to do so from an indeterminate one. For indeterminate descriptions, people often come up with models consistent with the description, but which the specifier did not have in mind.

For example, going back to the chair design, let us assume that the specification calls for a "comfortable" chair. Clearly "comfortable" has different semantics for different people. The designer can not use this specification as it is now, unless s/he does some "mind reading", which actually is practiced a lot in design, as well as in ordinary, every day communication. Using the *extended design ontology*, in DDSC the "walk-through" process prompts the user to refine its requirement "comfortable" in terms of variables like size, shape, material etc. (According to [Gruber, 1993]: "A specification of a representation vocabulary for a shared domain of discourse — definition of classes, relations, and other objects — is called an ontology." The concept of extended design ontology used herein refers to a model of the terms and their *dynamic* relationships in the respective design domain, i.e. to a highly structured model of the design *process.* This model has a domain independent structure.)

[1] Here we use the term "user" to include all participants like customers, clients, financing agents, etc. who are involved in the definition of the design object. We use the term "designer" to include all participants who are involved in the design and production of the design object.

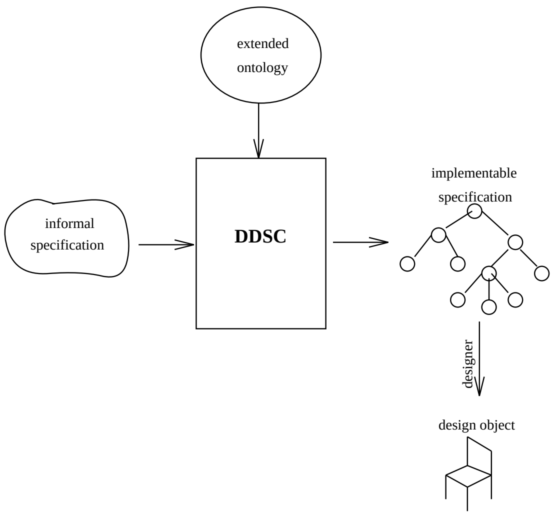

Figure 1: The DDSC concept

In the case of the Heart Project, almost all initial requirements, such as "durable," "visible," "easy to maintain," "portable," etc., had to be chunked down into unambiguous bits which were then made it almost trivial to test their satisfaction. An example for discrepancy was mathematical calculations performed for the same purpose by the user and the designers. The "walk-through," specifically the chunking, revealed that the discrepancy was due to different semantics of the term "static forces".

At the end of the "walk-through," the user establishes whether all aspects of say "easy maintainance" were exhausted in the process. If s/he thinks that this is not the case, then the Dialogue Agent *elicits the missing aspects* from the user and possibly enriches thereby the extended design ontology as well. In the above example such a missing aspect could be the maximum number of parts need to be removed for maintainance.

2.3 Inconsistency

Inconsistency may occur *within* the user's specification, but also *between* the user's specification and the *extended design ontology/designer's knowledge base*. An example for the first kind of inconsistency is a specification containing the requirement of modern furniture and at the same time an antique desk. Another example of the first kind, which occured in the Heart Project and is very common, is the inconsistency between the extent of the requirements and the time constraints set by the user. An example for the second kind of inconsistency is to require a wood floor in the kitchen while the code requires fire proof material.

Inconsistencies also occur between user requirements and designer's critera. As a consequence of such inconsistencies, designers ignore or override (not necessarily consciously) user requirements. For example, in the Heart Project, the designers chose to ignore the requirement of the ability to change heart rate on the fly because they didn't want to change the existing electronics. The "walk-through" process of DDSC is designed to discover and remove these inconsistencies (see next section).

3 The DDSC Concept and Architecture

DDSC is a domain independent, generic KB requirement elicitation tool which assists in the creation of a specification tree structure representing a complete, unambiguous and consistent specification of a desired design object. DDSC uses intelligent agents to provide expert knowledge about

1. the design process,
2. the user requirements,
3. knowledge common to the User and Designer Agents,
4. quantifiable requirements, and
5. calibrating requirements.

DDSC is *extensible* in both the number and the type of intelligent agents employed and in terms of the tools it interfaces. DDSC is an *interactive* system. DDSC takes as input an initial, informal requirement specification for a design object, as well as an extended generic ontology of the process needed to design the desired design object, and generates a complete, unambiguous, consistent, and implementable requirement specification of the desired design object. This specification is represented as a tree-like structure. This is a top-down structure, where the requirements at the terminal nodes are those which are either common knowledge, quantifiable, or calibrated (corresponding to the measurands in [Harris and Sydenham, 1994]).

In the present phase of concept development, the domain-specific intelligent agents of DDSC are the Designer Agent and the three Chunking Agents: Common

Knowledge Agent, Quantifying Agent, and Calibrating Agent. In addition, there are the User Agent and the Dialogue Agent, the latter being responsible for the transfer of information, eventually via a blackboard.

An initial implementation of a user interface module of DDSC is presently in process. The underlying architecture of DDSC will be an extended blackboard system (c.f. [Harris and Harvey, 1994]). As mentioned before, presently both the User and the Designer Agents are human: DDSC is designed as a system which *aids the user, not replaces him/her.*

3.1 The "Walk-Through"/Elicitation Process

This process is described recursively (see **LOOP** below).

An initial informal user specification, say SPEC, initiates the construction of a specification tree, which becomes the primary specification and is placed on the screen.

The designer enters the operations s/he has to perform to satisfy SPEC. These are stored in $OP_1, \ldots, OP_n$.

(begin LOOP) In general, for each operation OP to be performed by the designer, the corresponding window contains this operation in the main field, and below three kinds of subfields for rules, variables, and suboperations, respectively.

Rules

Here the designer lists all rules related to OP in his/her knowledge base. Each rule is of the form $R \rightarrow R'$ and is compared with the current user specification SPEC. If R is a requirement contained in SPEC, then the user is asked whether R' is consistent with the requirements in SPEC. If the answer is "no", the user changes SPEC to remove the inconsistency, and the "walk-through" backtracks as many steps as needed to accommodate the changed specification.

Suppose the design task is to design a zero scape landscaping of a yard and the specification calls for gardenias. Since there is a designer rule *Gardenia* $\rightarrow$ *Ten gallons of water per day*, the user has to change at least one of the requirements.

Variables

Here the designer lists all variables (slots) which need to be given values in order to perform OP.

In order to chunk the non-implementable requirements of the current specification SPEC into implementable requirements, the user is asked by the Dialogue Agent whether any of the requirements relate to the variables. If the answer is "yes," the user instantiates the relevant variables. For example, the user may find that the requirement "comfortable" relates to size, so the user will specify the size s/he thinks accommodates "comfortable". In addition, the user might not have thought of size to start with. S/he might want to instantiate variables even if no initial requirement relates to them, but at this point s/he realizes they are relevant. Variables which are not instantiated by the User Agent are given default values.

The instantiation of the variables by the user is controlled by the Chunking Agents. Values provided by the user for variables like size, color, fabric etc. are called *user requirements.* These requirements are often not implementable. Requirements like "big," "hot," "comfortable," "durable," "portable," or "easy to maintain" are subjective. They have a different meaning even for people with the same cultural and educational background. The exact meaning of such requirements can not be determined. It can only be determined relatively by calibrating individual perceptions. There are, however, *generally accepted* concepts within a domain. These fall into two categories: *common knowledge* and *quantifiable entities.*

The Chunking Agents take a requirement and check whether it is common knowledge, quantifiable, or can be calibrated. The Chunking Agents recognize common knowledge by looking it up in a domain specific database. A requirement is quantified if it has a *standard* numeric measure assigned to it. Examples for variables whose values are quantifiable are height, weight and volume.

Calibration means to match or adjust two value systems to a given scale. The concept of calibration is related to rating grids [Boose and Bradshaw, 1988] and scaling/clustering [Cooke and McDonald, 1988] used for knowledge acquisition.

If a requirement x does not fall into any of these categories, then the Chunkng Agents request through the Dialogue Agent a *more detailed or specific* definition of x, called *requirement equivalent of* x. To elicit from the user the requirement equivalent for x, where x can be a value like "durable", "comfortable" or "simple", the Dialogue Agent may use several knowledge elicitation techniques like e.g. structured interviews ([Lerner, 1956], [Sjoberg and Nett, 1968], [Zeisel, 1981]), and [McGraw and Seale, 1988], protocol analysis [Ericcson and Simon, 1984], and hypermedia tools ([Wells, 1989], [Kopec *et al.*, 1989])[2]. In addition, it uses pacing, leading, observation of linguistic cues, body language etc, or may even start an other "walk-through" process to facilitate the chunking of the value x into implementable values.

Suboperations

Here the designer lists all suboperations needed to be carried out in order to perform OP. **(end LOOP)**

Now **(begin LOOP)** through **(end LOOP)** is repeated for another operation, until all operations are exhausted and all relevant variables have implementable values. As mentioned before, this process can be interrupted by phases of actual design, in which the present specification is being satisfied. At the end of the process, the user is presented with the generated specification tree. If s/he thinks that any of its non-implementable requirements has not been fully redefined using just the variables presented during the "walk-through" process, the Dialogue Agent with the Chunking Agents once again help find the missing variables and give them val-

[2] For an excellent survey see [Byrd *et al.*, 1993].

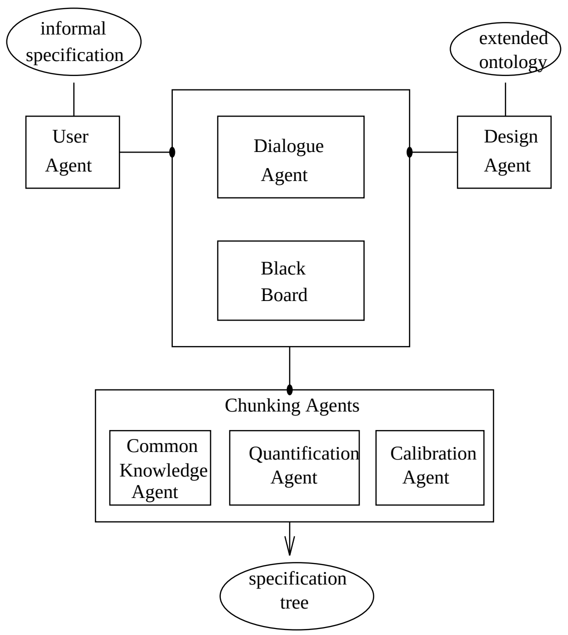

Figure 2: The architecture of DDSC

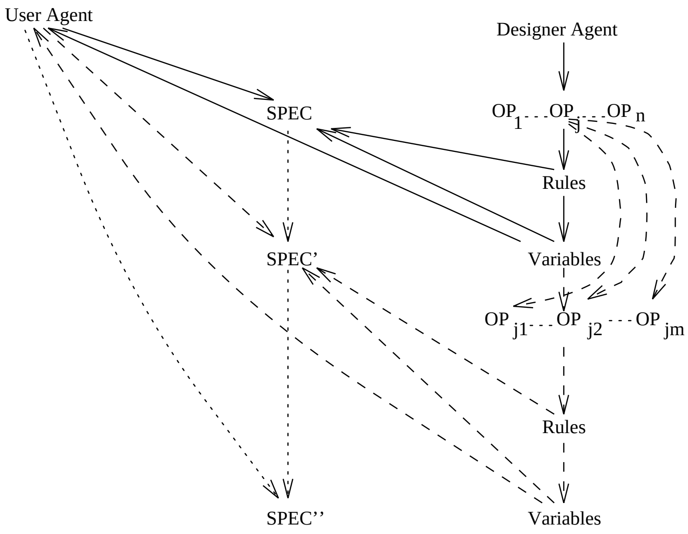

Figure 3: Schema of the "walk-through" process

ues as before. The specification tree is expanded accordingly.

4 Relation of DDSC to Relevant Research

The her described research closely relates to two major areas: Requirement Gathering, in particular Participatory Design for software systems, and Knowledge Acquisition for KB/Expert Systems.

1. The authors are not aware of any other general, domain independent RE tool for participatory design. Participatory design for software systems has recently become the focus of attention due to an increasing gap between users and software systems as these become more and more complex [Comm. of the ACM, 1995]. In particular, "The interest in participatory design, ethnographic techniques, and field research techniques grows out of the recognition that traditional interviews and surveying techniques are not adequate for today's applications." [Beyer and Holtzblatt, 1995]. The ideas behind the "walk-through" as a tool to aid requirement elicitation are closely related to the ideas behind tools like Contextual Inquiry ([Holtzblatt and Jones, 1993]), Future Workshops, or Organizational Games (see [Boedker *et al.*, 1993]).

   The concept of refining specifications interactively in a step-wise manner has been long pursued in software engineering and automating this process is an important research topic in AI (c.f. [Harandi and Lubars, 1991]).

2. Knowledge acquisition is a fundamental part of numerous problem solving processes. With the advancement of KB/Expert Systems it became one of the major research topics in AI. KA is not only fundamental, but also probably the most difficult and time consuming part of the problem solving process ([Gruber, 1989]). It is also one of the major bottlenecks in the development of KB/Expert Systems. This bottleneck problem has been tackled first by domain specific tools such as SALT [Markus, 1988], [Marcus and McDermott, 1989], KNACK [Klinker, 1988], MORE [Kahn *et al.*, 1985], ROGET [Bennett, 1985], MOLE [Eshelman, 1988], and SIZZLE [Offutt, 1988]. These are based on modeling of problem solving methods and domain models.

   Other KA-tool designs took this approach a step further and tried to integrate a variety of problem solving methodologies or utilities into a single workbench ([Abrett and Burstein, 1987], [Boose and Bradshaw, 1988], [Diedrich *et al.*, 1987]). However, these tools are limited to particular application domains and are difficult to modify for new applications.

   In order to overcome such limitations, new methods and a new generation of *generic* meta-level tools, such as DIDS [Runkel and Birmingham, 1993], Spark [Eriksson and Musen, 1993], KADS [Wielinga *et al.*, 1992], DASH [Eriksson *et al.*, 1994], GAS [Chien *et al.*, 1995], SBF [Yost *et al.*, 1994] and PRO-MINDS [Harris and Sydenham, 1994] have been developed. These tools can either help to generate new domain-specific KA-tools or acquire knowledge for a larger domain by applying multiple and alternative problem solving methods.

   Many features of these KA-tools are incorporated into the DDSC concept. In particular, ontologies are used as a basis for the RE-process similar to the approach proposed in DASH. Currently, the authors are also working on the development of a user-interface that is modeled after the principles of the *dialog-designer module* of DASH. An important model for the development of DDSC is the PRO-MINDS project whose aim is to support the design process by a multi-sensor system that can elicit knowledge about observed objects or events.

   Besides the above mentioned similarities to other KA-tools and methods, DDSC has the following unique features:

   (a) DDSC is a generic RE-tool Its application domain is not restricted to a particular design domain. This allows a *structuring* of the extended ontology which is *independent* of the particular application. DDSC, besides being a domain independent software engineering tool like e.g., DASH or GAS, also provides RE techniques.

   (b) Sessions with DDSC *change dynamically* the extended design ontology as well as the KB of the agents involved.

   (c) DDSC is the only RE-tool which systematically refines/chunks all elicited requirements into implementable bits. It is able to do this because it employs effective elicitation methods based on the structure of human cognition through the use of recognized cognitive categories.

Although currently DDSC presupposes the use of single agents communicating sequentially, future implementations will accomodate the concept of concurrent design with geographically distributed teams.

References

[Abrett and Burstein, 1987] G. Abrett and M. Burstein. The kreme knowledge editing environment. *International Journal of Man-Machine Studies*, 27(2):103–126, 1987.

[Alexander *et al.*, 1988] J.H. Alexander, M.J. Freiling, S.J. Shulman, S. Rehfuss, and S.L. Messick. *Knowledge Acquisition Tools for Expert Systems*, volume 2 of *Knowledge Based-Systems*, chapter Ontological Analysis: An Ongoing Experiment, pages 25–37. Academic Press, 1988.

[Bennett, 1985] J.S. Bennett. Roget: A knowledge-based system for acquiring the conceptual structure of a diagnostic expert system. *Journal of Automated Reasoning*, 1:49–74, 1985.

[Beyer and Holtzblatt, 1995] Hugh R. Beyer and Karen Holtzblatt. Apprenticing with the customer. *Communication of the ACM*, 38(5):45–52, May 1995.

[Boedker *et al.*, 1993] S. Boedker, K. Groenbaek, and M. Kyng. *Participatory Design*, chapter Cooperative Design: Techniques and Experiences from the Scandinavian Scene. Lawrence Erlbaum Assoc., 1993.

[Boose and Bradshaw, 1988] J.H. Boose and J.M. Bradshaw. *Knowledge Acquisition Tools for Expert Systems*, volume 2 of *Knowledge-Based Systems*, chapter Expertise Transfer and Complex Problems: Using AQUINAS as a Knowledge-Acquisition Workbench for Knowledge-Based Systems, pages 39–64. Academic Press, 1988.

[Byrd *et al.*, 1993] Terry Anthony Byrd, Kathy L. Cossick, and Robert W. Zmund. A synthesis of research on requirrement analysis and knowledge acquisition techniques. *Management Information Systems Quarterly*, pages 117–138, March 1993.

[Cameron-Bandler and Lebeau, 1985] L. Cameron-Bandler and D. Gordon Lebeau. *The Emprint Method. A Guide to Reproducing Competence.* Real People Press, Moab, Utah, 1985.

[Chien *et al.*, 1995] ChiH-Cheng Chien, Cheng-Seen Ho, and Hahn-Ming Lee. mwkat: A meta-tool-generated knowledge acquisition tool for multimedia workstations development. *Applied Intelligence*, 5:51–72, 1995.

[Codd, 1979] E.F. Codd. Extending the database relational model to capture more meaning. *ACM TODS*, 4(4):397–434, December 1979.

[Comm. of the ACM, 1995] Comm. of the ACM. Requirement gathering: The human factor, May 1995.

[Cooke and McDonald, 1988] N.M. Cooke and J.E. McDonald. *Knowledge Acquisition Tools for Expert Systems*, volume 2 of *Knowledge Based-Systems*, chapter The Application of Psychological Scaling Techniques to Knowledge Elicitation for Knowledge-Based Systems, pages 65–82. Academic Press, 1988.

[Davis, 1982] R. Davis. *Knowledge-Based Systems in Artificial Intelligence*, chapter Teiresias: Applications of meta-level knowledge, pages 227–490. McGraw-Hill, New York, 1982.

[Diedrich *et al.*, 1987] J. Diedrich, I. Ruhmann, and M. May. Kriton: A knowledge acquisition tool for expert systems. *International Journal of Man-Machine Studies*, 26(1):29–40, 1987.

[Ericcson and Simon, 1984] K.A. Ericcson and H.A. Simon. *Protocol Analysis: Verbal Reports as Data.* MIT Press, Cambridge, 1984.

[Eriksson and Musen, 1993] H. Eriksson and M. Musen. Metatools for knowledge aquisition. *IEEE Software*, pages 23–29, May 1993.

[Eriksson *et al.*, 1994] H. Eriksson, A.R. Puerta, and M.A. Musen. Generation of knowledge-acquisition tools from domain ontologies. *International Journal Human Computer Studies*, 41:425–453, 1994.

[Eshelman, 1988] L. Eshelman. *Automating Knowledge Aquisition for Expert Systems*, chapter MOLE: A knowledge acquisition tool for cover-and-differentiate systems, pages 37–80. Kluwer Academic Publishers, MA, 1988.

[Gruber, 1989] T.R. Gruber. *The Acquisition of Strategic Knowledge.* Academic Press, San Diego, CA, 1989.

[Gruber, 1993] T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.

[Harandi and Lubars, 1991] M.T. Harandi and M.D. Lubars. *Artificial Intelligence & Software Engineering*, chapter Automating Software Specification and Design, pages 43–59. Ablex Publishing, Norwood, New Jersey, 1991.

[Harris and Harvey, 1994] D.D. Harris and J.G. Harvey. Tailoring the blackboard concept to produce a flexible, extensible knowledge-based design tool. Technical report, University of South Australia, Levels Campus, The Levels, 5095, South Australia, 1994.

[Harris and Sydenham, 1994] D.D. Harris and P.H. Sydenham. Pro-minds: Development of a software tool to support the measurement system designer. *Measurements*, (15):1–14, 1994.

[Holtzblatt and Jones, 1993] K. Holtzblatt and S. Jones. *Participatory Design*, chapter Contextual Inquiry: A Participatory Technique for System. Lawrence Erlbaum Assoc., 1993.

[Johnson-Laird, 1983] P.N. Johnson-Laird. *Mental Models.* Harvard University Press, 1983.

[Kahn *et al.*, 1985] G. Kahn, S. Nowlan, and J. McDermott. Strategies for knowledge acquisition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 7(5):511–522, September 1985.

[Kellogg *et al.*, 1989] C. Kellogg, R.A. Gargan Jr., W. Mark, J.G. McGuire, M. Pontecorvo, J.L. Schlossberg, J.W. Sullivan, and N. Singh M.R. Genesareth. The acquisition, verification, and explanation of design knowledge. *SIGART Newsletter, Knowledge Acquisition Special Issue*, (108):163–165, April 1989.

[Klinker, 1988] G. Klinker. *Automating Knowledge Acquisition for Expert Systems*, chapter KNACK: Sample-driven knowledge acquisition for reporting systems, pages 125–174. Kluwer Academic Publishers, MA, 1988.

[Kopec *et al.*, 1989] D. Kopec, L. Latour, and M. Brody. Towards an expert/novice learning system with application to infectious disease. *SIGART Newsletter, Knowledge Acquisition Special Issue*, (108):140–143, April 1989.

[Lancaster *et al.*, 1989] Juliana S. Lancaster, Christopher R. Westpha, and Karen L. McGraw. A cognitively valid knowledge acquisition tool. *SIGART Newsletter, Knowledge Acquisition Special Issue*, (108):152–154., April 1989.

[Lenart *et al.*, 1994] M. Lenart, P. Padawitz, and A. Pasztor. *Formal Design Methods for CAD*, chapter Formal Specification for Design Automation, pages 201–220. IFIP Transactions B18. Elsevier, North Holland, 1994.

[Lerner, 1956] D. Lerner. Interviewing frenchmen. *American Journal of Sociology*, (61):187–194, 1956.

[Marcus and McDermott, 1989] S. Marcus and J. McDermott. Salt: A knowledge acquisition language for propose and revise system. *Artificial Intelligence*, 39(1):1–37, May 1989.

[Markus, 1988] S. Markus. *Knowledge Acquisition Tools for Expert Systems*, volume 2 of *Knowledge-Based Systems*, chapter Taking Backtracking with a Grain of SALT, pages 211–226. Academic Press, 1988.

[McGraw and Seale, 1988] K. McGraw and M.R. Seale. Knowledge elicitation with multiple experts: Considerations and techniques. *Artificial Intelligence Review*, 2:31–44, 1988.

[Offutt, 1988] D. Offutt. *Automating Knowledge Acquisition for Expert Systems*, chapter SIZZLE: A knowledge-aquisition tool specialized for the sizing task, pages 37–80. Kluwer Academic Publishers, MA, 1988.

[Partridge and Wilks, 1987] D. Partridge and Y. Wilks. Does ai have a methodology that is different from software engineering? *AI Review*, 1:111–120, 1987.

[Rattray *et al.*, 1991] C. Rattray, J. McInnes, and A. Reeves. *Artificial Intelligence & Software Engineering*, chapter A knowledge-based model for software reusability, pages 61–82. Ablex Publishing, Norwood, New Jersey, 1991.

[Runkel and Birmingham, 1993] J.T. Runkel and W.P. Birmingham. Knowledge acquisition in the small: building knowledge-acquisition tools from pieces. *Knowledge Acquisition*, 5:221–243, 1993.

[Sjoberg and Nett, 1968] G. Sjoberg and R. Nett. *A Methodology for Social Research.* Harper and Row, New York, 1968.

[Wells, 1989] T.L. Wells. Hypertext as a means for knowledge acquisition. *SIGART Newsletter, Knowledge Acquisition Special Issue*, (108):136–138, April 1989.

[Wielinga *et al.*, 1992] B.J. Wielinga, A.T. Schreiber, and J.A. Breuker. Kads: a modeling approach to knowledge engineering. *Knowledge Acquisition*, 4(1):5–53, March 1992.

[Yost *et al.*, 1994] G.R Yost, G. Klinker, M. Linster, D. Marques, and J. McDermott. The sbf framework, 1989-1994:from applications to workplaces. In G. Schreiber and W. Van de Velde, editors, *A Future for Knowledge Acquisition, 8th European Knowledge Acquisition Workshop, EKAW '94*, pages 318–339, Hoegaarden, Belgium, September 1994. Springer. Lecture Notes in AI 867.

[Zeisel, 1981] J. Zeisel. *Inquiry by Design: Tools for Environment-Behavior Research.* Brooks Cole, Monterey, CA, 1981.