Knowledge Elicitation
for the Design of Software Agents

Guy A. Boy
European Institute of Cognitive Sciences and Engineering (EURISCO)
4 avenue Edouard Belin, 31400 Toulouse, France
Tel. +33-62 17 38 38 - FAX +33-62 17 38 39 - Email: boy@onecert.fr

Abstract
For many years, the theory and practice of knowledge acquisition for knowledge-based systems tended to focus on how to elicit and represent knowledge in a context-free way. More recently, the evolution of the design of software agents has forced the focus to shift so that the context of the interaction between human and software agents has become the unit of analysis. In terms of empirical research, the initial goal was to establish a unified model that would guide knowledge elicitation for the design of software agents. The SRAR (Situation Recognition and Analytical Reasoning) model has been developed over the last ten years to serve this goal. Subsequently the knowledge block representation has been elaborated as a mediating representation to help formalize elicited knowledge from experts or, more generally, end-users. Hence, human-centered automation studies have more recently started to focus on the requirements for the design and evaluation of "human-like" systems that are currently called software agents. In this chapter, we introduce the cognitive function analysis method that helps elicit task/activity models from experts.

1. Introduction
For many years, the theory and practice of knowledge acquisition for knowledge-based systems (expert systems) tended to focus on how to elicit and represent knowledge in a context-free way. This reflected a position which was dominant both in cognitive psychology and artificial intelligence (AI) in the 1970s and 1980s, where cognition was seen as a product of isolated information processors (Newell and Simon, 1972), and where the notion of context was not a central issue. Even worse, AI research mainly focused on decontextualization of knowledge. In the late 1980s, several research efforts began to highlight the importance of context from a social perspective.

"A major weakness of the conventional approach to knowledge elicitation (and the epistemology that underpins it) is that it ignores the social character of knowledge — that is, the possibly unpalatable (to AI), but none-the-less inescapable fact that knowledge is part-and-parcel of the social world (Collins, 1987)... Sociologists of science argue
that much of knowledge is held in a tacit basis forming part of our intersubjectively shared commonsense understanding of the world... the rules contained in an expert system are meaningless; their meaning arises from the social or cultural contexts within which expertise is usually embedded and rests on an immense foundation of tacit knowledge." (Bloomfield, 1988, page 20)

Dreyfus's (1981) critique of AI microworlds, and AI in general, was based on the fact that "a world is an organized body of objects, purposes, skills, and practices in terms of which human activities have meaning or make sense." In this chapter, we will assume that software agents are computer programs that are designed and developed within/for a social context to respond to specific needs such as enhancing human productivity, comfort and safety. From this perspective, the design of software agents such as intelligent assistant systems (Boy, 1991) requires the use of knowledge elicitation techniques in order to gather requirements from various viewpoints that can be either technical, economical, psychological, or sociological. Typically, such software agents are designed to be used for decision support, i.e., systems that support the decision of people who already know quite a lot about a given domain (Bloomfield, 1988).

"As a further consequence of the over-emphasis on knowledge which can be articulated, knowledge engineering largely ignores the visual/ostensive component of expertise. For instance, experts — like most other humans — tend to employ pictures or images as part of their thinking processes and indeed such elements have been an integral part of scientific and technological development. It turns out that the visual component of expertise also has a tacit dimension rooted in our learning experiences within the social world around us." (Bloomfield, 1988, page 23)

An expert system emulates what a human expert can do. In the context of human-computer interaction, such an expert system is a specialized application at the disposal of the user. In this chapter, a software agent is defined as a specialized application with an appearance that is the visual dimension of its purpose.

"There has been some debate concerning whether intelligent interfaces should be structured as agents or whether it is better to think of them as tools with intelligently organized direct-manipulation options available to the user. In the interface-as-agent view, the interface is seen as a separate entity that mediates between the user and the machine." (Chin, 1991).

Obviously, the user needs to know what to do with the agent, i.e., possible actions on it, messages that can be sent to and from the agent, the agent's behavior, its possible reactions, etc. Conversely, the agent view requires that the interface has a well-defined model of the dialogue between the user and the interface [4]. Our agent approach differs from conventional natural language dialogues in the sense that it is more constrained. Our goal is, however, to keep a natural interaction mode and its relative flexibility. Sometimes, the interface has greater knowledge than the user. Thus it may automatically correct user's mistakes, suggest an alternative course of action, and provide appropriate information to help the user to interact with it. In other words, interface agents assist the user to perform a particular task.
"Human interface designers are struggling to generate more effective illusions for purposes of communicating to their users the design model of their applications. At the same time, they are confronting serious issues of ethics: when does an attempt to create an empowering illusion become trickery, when does an attempt at anthropomorphism become cheap fraud?" (Tognazzini, 1993).

Thus, the products of knowledge elicitation for the design of software agents are not only rules and facts that would be used by an information processor, but also pictures, graphics and sound. That is, a software agent is not only a "crunching" mechanism, but also a visible and audible entity.

1.1. Knowledge elicitation

Knowledge elicitation is an inherently co-operative activity (Addis, Gooding and Townsend, 1993). For this reason, the design of software agents is an incremental process that should enable the design and refinement of artifacts approximating expert views. The computer screen is then a mediating tool that enables collaborative design of such artifacts. A software agent can be constructed using a diagrammatic representation (knowledge blocks) associated with an appearance design tool. Experts and cognitive engineers cooperate through the current block base and its corresponding appearances. The more appearances are familiar to the expert, the more visual design is improved.

Leplat gave six general traits characterizing an expert (Leplat, 1986):

- the expert knows how to solve problems in his/her field, even those occurring for the first time;
- the expert knows how to restate badly formulated problems clearly and to represent the problem in such a way as to allow it to be solved;
- the expert knows how to evaluate the consequences of a deviation from optimal conditions in the solving of a problem;
- the expert builds up his/her knowledge through practice and may periodically reorganize this knowledge;
- the expert can make decisions with incomplete or temporarily missing information (Lenat, 1983);
- in certain cases, the expert is also the person who knows how to combine different types of knowledge that are available concerning a particular phenomenon in order to predict its properties or behavior (Hawkins, 1983).

Expert knowledge is inherently compiled by analogy with a compiled computer program. Once it is compiled, it is extremely difficult and often impossible to de-compile it. The only way to approximate such knowledge is to observe the result of its processing, and rely on a relevant model. Rasmussen (1983) has already proposed a categorization of human behavior (Figure 1).
The acquisition of skill-based behavior, both sensory-motor and cognitive, is the result of long and intensive training. Skill-based behavior permits rapid stimulus–response type operations. The level of rule-based behavior is also an operative level, with manipulation of specific plans, procedures (e.g. a checklist or a cooking recipe) or rules (know-how). Knowledge-based behavior is the "true" level of intelligence. Current expert systems are mostly still at the level of procedures, because it is difficult (and often impossible) to elicit the compiled expert knowledge which exists at the skill-based behavior level (Dreyfus, 1982). Expert explanations usually come from the intermediate level. Like a professor teaching, the expert must "decompile" his knowledge to explain the "why" and "how" of his behavior. The result of such decompilation is easily transferable to a declarative form on a computer using for example the IF...THEN... format (generally called a rule). Unfortunately, the result of the decompilation through the observation of behavior is usually not sufficient to elicit the internal cognitive processes that generated this behavior. Furthermore, the analytical process of decompilation usually performed by a cognitive engineer or psychologist does not necessarily capture the knowledge and the expert behavior used at the skill-based level without a good cognitive model.

"It is simplistic and misleading to assume that the process that leads to the emulation of human expertise in a computer program is one of transferring knowledge — 'expertise transfer' is an attractive metaphor but it leaves open questions of what is expertise and how it may be transferred. A better metaphor might be one of modeling, that the emulation involves building a model of expertise, where a 'model' is according to Webster's dictionary: a representation, generally in miniature, to show the construction or serve as a copy of something." (Gaines, 1993).
Gaines's remark of the shift from expertise transfer to expertise modeling is essential, and deserves further analysis. We claim that expertise transfer (from human to machine) leads to clumsy automation (Wiener, 1989) when it is not controlled using a socially-recognized model and incrementally assessed through experimental protocols. Knowledge cannot be dissociated from its related inference mechanisms. Furthermore, knowledge elicitation is guided by specific intentions that influence the knowledge acquired, e.g., human-centered automation where one of the major objectives is to master the balance of control and monitoring between the human and the machine being automated (Billings, 1991). Finally, we need to consider that expertise evolves. We cannot trust a snapshot of it. This is why context is important to capture in order to properly index expertise.

1.2. User-centered design: a problem statement

Over the years, we have developed the paradigm of integrated human-machine intelligence (IHMI) (Boy & Nuss, 1988; Shalin & Boy, 1989; Boy & Gruber, 1991) to provide a framework for acquiring knowledge useful for the implementation of a software agent. This paradigm is presented in Figure 2. Arrows represent information flows. In this model, a software agent has three modules: a proposer that displays appropriate information to the human operator; a supervisory observer that captures relevant actions of the human operator; and an analyzer that processes and interprets these actions in order to produce appropriate information for the proposer module. Human-factors research and practice already propose generic rules for the definition of what relevant actions and appropriate information means in specific context and domains. However, there is no universal model available for the specification of such entities in general. The proposer, the supervisory observer and the analyser use two knowledge bases: the technical knowledge base and the operational knowledge base. The technical knowledge base is the theory (syntax and semantics) of the domain that the agent should know. It represents how its environment works. The operational knowledge base is the pragmatism that the agent should possess to act on the environment effectively. This model includes two loops:

- a short-term supervisory control loop that represents interactions between the human operator and the software agent; it involves mutual control between the human operator and the software agent; and
- a long-term evaluation loop that represents the knowledge acquisition process; it involves a designer and a knowledge acquisition method.
In the past, software design that leads to advanced automation has been essentially constructed and applied by engineers. Human factors people had very little impact on the design of complex systems. Marc Pélégrin, a well-recognized scientist in control theory, advocated in his introductory talk at the French Academy of Sciences that "the future of automation is semi-automation," apologizing for the fact that automation specialists have gone too far without taking into account end users. We would like to add that end users are only one concern. Software design needs to be considered in a broader sense for the whole of humanity including social, ecological, economical as well as psychological repercussions. Thus, software design needs to be human-centered because it should be done for the benefit of the largest range of end users. This benefit is related to a massive use and self-customization of the information technology, e.g., the Internet.

The aeronautical industry used to be a good example of human-centered design. In the early days, airplanes (we might say flying machines) were built by the people who flew them. Thus, they adjusted the controls and other important devices according to their needs and knowledge. Today there is a chain of jobs between design and routine use. The main problem is to bridge the gap between design and routine use. This issue is about viewpoint management. The more we broaden the spectrum of viewpoints during design, the more complex the design process (it is difficult to manage too many creative people!). Efficiency is at stake here. A designer alone in his/her office or laboratory can be very efficient but not necessarily open to other viewpoints (not because his/her mind is not open, but because he/she does not have access to other people's experience). Introducing end users in the design loop is not trivial for many reasons such as cooperation, complexity and the difficulty of gathering the right people at the right time. This can be
extremely inefficient, and there is a price to pay in terms of adaptation to a new way of doing things, but the price may decrease as we better understand how various participants in the design phase can communicate, share context, negotiate, etc. Cooperative work in design is therefore an important issue that is developed in other chapters of this handbook.

An important issue is to make the constraints explicit enough to guide the decisions during the design process. Some constraints are economical, others might be safety-oriented or ecological. People in charge of a project are responsible for the decisions they make and need to know explicitly what they are doing! If deciders know about potential risks or potential problems that a new design induces, they will reconsider this design. Design should be human-centered for the designers themselves. Human-centered design tools are extremely important in the perspective of an efficient use of a corporate memory. The concept of corporate memory has become important for several reasons (Boy, 1995). One of them is the visualization (conceptualization aid) of the concepts/artifacts being designed. In particular, explicit visualization of designed artifacts facilitates usability tests.

1.3. Context categories

The concept of context is not easy to define. We will take the reflexive-deliberative viewpoint to define the context, in contrast with an environment-driven viewpoint (Suchman, 1987). Context can be related to the persistence of a situation (see section 2.1 for the definition of a situation), i.e., the more a situation persists, the more it is contextual. For instance, if you stay in the same room everyday in the morning for three hours, this pattern becomes contextual. We contextualize by incrementally constructing increasingly complex but meaningful contextual patterns. For instance, words are defined by incrementally using them in various specific situations. Even if you do not know what the word wishbone means, the following sentences will contribute to its definition:

He used the best wishbone available in the race... This very short wishbone allowed him to turn shorter than the others at the marker buoy... Another reason is that such a wishbone is attached very high on the mast... In fact, all windsurf boards having such a wishbone should be used by real professionals...

After the first sentence, we learn that a wishbone can be used in a race. Even if we do not know what kind of race, this eliminates the possibility of an animal bone. In the second sentence, we learn that the wishbone is used on the water because of the marker buoy. After hearing the third sentence, we know that the wishbone is on a boat or some floating device. Finally, in the last sentence, we learn that a wishbone is part of a windsurf board (we assume that we know what a windsurf board is, otherwise we will continue to identify this new word in following conversations).

Thus, by adding situations (context) in which a word is used, this word can be incrementally defined. Such a context is called verbal context. This means that a word defined in a verbal context is defined by the other words around it. In information retrieval, the keyword in (verbal) context (KWIC) has been used as visual correlation during the selection (Luhn, 1959). This technique has been automated by Zimmerman (1988) and is used in CID (Boy, 1991c).
There is another form of context, that is physical and social. If you are presented with a windsurf board for instance, its owner may explain to you how to use it. In the conversation, he may say, "If you take the wishbone that way, the wind will push the sail in that direction". You suddenly discover that this piece of light metal supporting the sail laterally is called the wishbone. You learn this in a physical context. It may even happen that you will remember this word in this context forever, i.e., each time you think about your friend demonstrating his windsurf board, you will remember the anecdote of the wishbone, and conversely when you think about the wishbone. This is to say that context is very important for remembering things. In particular, in information retrieval, we claim that if context is used appropriately, it will enhance the search in memory. Hayakawa (1990) says that: The "definitions" given by little children in school show clearly how they associate words with situations. They almost always define in terms of physical and social contexts: "Punishment is when you have been bad and you have to sit on the stairs for a time out." "Newspapers are what the paperboy brings.

In more formal terms, context can be defined by relationships between objects. An object can be a word or phrase, a picture, a movie sequence, a physical object (such as a table), a person, etc. These attribute/value pairs may be descriptions of verbal context (what is around the object and defines it in some situations), physical and social context (what one can point out in the environment of the object to concretize its definition), or historical context (what happened before and is causally related to the object being described). For instance the context of an information retrieval search could be described by a user profile (user model), e.g., the type of user, the type of task he/she is currently doing, the time, etc. The main problem with formal definition of context is that it can lead to a very large set of attribute/value pairs. From a computational point of view, consequent pattern matching may become practically impossible. This leads to the idea that context is usually defined by default for the most frequently used attribute/value pairs structure. This structure may have some exceptions. For instance, when we use the word "chair", it denotes a physical device that is used to sit on. However, in the context of a conference, the "chair" is usually the person who is in charge of the conference.

As Hayakawa (1990) pointed out, words have extensional and intensional meaning. If a word can be described by a real object that you can see in the environment, its meaning is extensional. For instance, if you point at a cow in a field, you denote the cow. If a word is described by other words to suggest its presence, then its meaning is intensional. For example, you may describe a cow by saying that it is an animal, it has horns, it lives on a farm, etc. Each object can be associated to a particular intensional description according to who is giving the description, when the description is given, etc. Again, context is a key issue when objects have to be described. It is a fallacy to claim that each object could be associated with a single word. Anyone who has tried to retrieve documents in a library using keywords equations knows this. It is almost impossible to retrieve the desired information because the description one gives for the query seldom matches the descriptions that librarians have developed. However, if you ask the
librarian, he/she may help you better by acquiring more context from you. In the best case, if the librarian is a good friend and knows your work and needs, then he/she will be very helpful. If the librarian does not know you, he/she can capture physical context by simply looking at you. He/she may consider facts such as: you are young; you wear a lab coat; etc. He/she will also capture context from what you say (verbal context). Hayakawa says that "an examination of the verbal context of an utterance, as well as an examination of the utterance itself, directs us to its intensional meanings; an examination of the physical context directs us to the extensional meanings."

Context includes a temporal aspect, i.e., context summarizes a time period. Furthermore, persistence of some events reinforces context. For instance, a user told the librarian that he/she is looking for some information on geometry, he/she is currently involved in a computer class, he/she has a problem of drawing curves using a set of points, he/she needs to obtain a continuous drawing on the screen, and later he/she finally requested a reference on splines. The librarian usually integrates historical context, and, in this specific example, will not be confused between mathematical spline functions (what the requester needs) and the physical splines that draughtsmen used in "ancient" times. In this example, the words "computer" and "screen" help to decide that the requested "splines" will be used in the last quarter of the 20th century.

1.4. The agent perspective

The agent-orientation of human-machine interaction is not new. Autopilots have been commonly used since the 1930's. Such agents perform tasks that human pilots usually perform, such as following a flight track or maintaining an altitude. Transferring such tasks to the machine modifies the original task of the human operator. The job of the human operator evolves from a manipulation task (usually involving sensory-motor skills) to a supervisory task (involving more cognitive processing and new situation-awareness skills) (Sheridan, 1992). Creating software agents involves new cooperation and coordination processes that were not explicit before. Software agents involve active behavior of the users. They enable users to center their interactions at the content level (semantics) partially removing syntactical difficulties. They also enable the indexing (contextualization) of content to specific situations that they understand better (pragmatics).

Software agent technology is consistent with the design and development of component software architectures such as OpenDoc, OLE 2.0 or CORBA. These architectures support collaboration, integration and customization of applications. They provide software interoperability by enabling the design and refinement of distributed, cross-platform component software. They enable cooperative design. The nature of component programming leads to the notion of component assembly to fit the targeted task. Software agents are components that must be kept simple to understand and use. We claim that knowledge elicitation is easier and more efficient when it is possible to create and "immediately" assess software images of knowledge chunks. A chunk is defined as a parcel of knowledge that can be treated as an independent variable. In addition, this constructive process improves with experience. For instance, the six letters "A", "T", "L", "E", "B", and "S" can be treated...
independently, as the word "TABLES" can also be treated independently and results from assembly of the previous six letters.

Knowledge elicitation becomes a design process enhanced by the interactive and component-based software technology. However, this technology-rich approach should not overshadow the awareness of the representational model that is used in such a knowledge design process. Ferber (1989) differentiated three different representational models: analytical, structural and interactionist.

- The *analytical* approach considers knowledge as a set of assertions in a particular domain. These assertions may be first principles (e.g. physical laws), transcriptions of system behavior in the form of statements (e.g. "the thermostat regulates the temperature of the room") or facts based on experience (e.g. "generally when it gets too cold, it is because the thermostat is broken"). Logic has been used as a basic representation in this kind of approach. The analytical approach is linguistic (no knowledge without language), logical and reductionist (inference is made from primitive elements). This approach fails when temporal entities have to be considered, when statements describe dynamic processes, when different statements are contradictory (resulting from different points of view), and when ambiguities are present.

- The *structuralist* approach considers knowledge as a network of interrelated concepts. Semantic networks and frames have been used to understand deep structures in natural languages. In particular, generic (classes) and individual (instances) concepts and inheritance mechanisms have been studied in this approach. Knowledge processing is considered to be a form of general pattern recognition. The main limitation of this approach is its lack of definition and formalization. The structuralist approach is well developed in Europe, especially with Piaget's work (1967) on the development of intelligence.

- The *interactionist* approach is very recent. It is based on the principle that knowledge is acquired and structured incrementally through interactions with other actors or autonomous agents. Interaction and auto-organization are the key factors for the construction of general concepts. There are two main currents: automata networks (von Neumann, 1966; McCulloch & Pitts, 1943; Rosenblatt, 1958) and connectionism (Rumelhart *et al*., 1986; McClelland *et al*., 1986); and distributed AI (Minsky, 1985; Hewitt, 1977; Lesser & Corkill, 1983).

The knowledge elicitation approach developed in this chapter is essentially interactionist. It is based on the design/refinement of a model of the human-machine system being developed. Following Gaines, expertise should be modeled to better understand what should be automated (transferred to the machine) and what should be kept by the human to process. Knowledge elicitation is then enhanced through the mediating role of computer media. On this perspective, the term "Sociomedia" has been introduced to emphasize the social construction of knowledge (Barrett, 1992). The main issue is not so much to validate assumptions as to find appropriate assumptions to test (abduction process). Thus the usual hypothetical-deductive approach mainly based on a (one shot) batch process is replaced by a multi-
abduction approach relying on interaction.

1.5. Designing agents by incremental knowledge elicitation

We have advocated the position that knowledge elicitation should be situated. Software agents both help knowledge elicitation in context, and are designed and refined from this knowledge elicitation. An artifact is a physical or conceptual human-designed entity useful for a given class of users to perform specific tasks. Carroll and Rosson (1991) described transactions between tasks and artifacts. It is sometimes difficult to know if the task defines the artifact or if the artifact defines the task. In reality, users' profiles, tasks and artifacts are incrementally defined. The classical engineering tradition is centered on the construction of the artifact. The task and the user are usually taken into account implicitly. Tasks can be modeled from a task analysis or a model of the process that the artifact will help to perform. A specified task leads to a set of information requirements for the artifact. Conversely, the artifact sends back its own technological limitations according to the current availability of technology. Users can be incrementally taken into account in the design loop either through the development of syntaxo-semantic user models or through the adaptation of analogous user models. User modeling can be implicit or explicit, and leads to the definition of appropriate user profiles. When a version of the artifact and the task are available, the user can use the artifact to perform the task. An analysis of the user activity is then possible, and contributes to modifying both the task and the artifact. The use of the artifact provides data to adapt both the artifact to the user (ergonomics), and the user to the artifact (procedures and training). The artifact-task-user triangle is described in Figure 3. It implicitly defines an incremental approach to design that is elsewhere described as a spiral model for software development (Boehm, 1988). It is based on the design of representative components that are incrementally validated with respect to functional requirements, hardware and software constraints, and strategic and economic issues of the organization. Design rationale is attached to each component.

A similar analysis was provided by Woods and Roth (1988) with their cognitive systems triad. They outlined the factors that contribute to the complexity and difficulty of problem solving by describing: the world in terms of dynamism, many interacting parts, uncertainty and risk; the representation that can be fixed, adaptive, computational or analogical; and the agent in terms of multiple agents and joint cognitive systems. In our approach, these dimensions are taken as independent variables useful for incremental definition of the task, the artifact and the user.
In the next section, an agent model is proposed. The process of incremental knowledge elicitation is supported by the SRAR model that is described at length in section 3 of this chapter. This model leads to the knowledge block representation presented in section 4. Knowledge blocks support the representation of cognitive functions that are useful for modeling user-centered designs. This new method, called cognitive function analysis (CFA) is presented, and a global approach to design is provided in section 5. Section 6 is devoted to conclusions and perspectives.

2. An agent model

Various definitions have already been proposed for an agent. Commenting the on-line Random House Dictionary definition of an agent, Minsky said:

"When you use the word 'agent'... there is no implication that what the agent does is simple... the agent is seen as having some specialized purpose. If you need help with making investments, you call a financial agent. If you're looking for a job you call an employment agent. In the present-day jargon of computer interface agents, the word is used for a system that can serve as a go-between, because of possessing some specialized skill." (Minsky and Riecken, 1994)

The function of an agent can be defined in several ways. In an aircraft for example, a human copilot shares the work with the pilot—but not the ultimate responsibility. The pilot can consult the copilot on any point concerning the flight but will make ultimate decisions. If the pilot delegates a part of his responsibilities to the copilot, the copilot will take this delegation as a task to execute. The same principle applies to the pilot who is mandated by his company to transport safely and comfortably the passengers from one airport to another. The pilot can stop the execution of a copilot task at any time, if it is necessary. However, a copilot may have personal initiatives, e.g., testing parameters, constructing his/her own awareness of the actual situation, predicting deducible failures, etc. A copilot may process the information included in an operation manual at the request of the pilot. He should be able to explain, at an appropriate level of detail, the result of his processing.

A substantial increase in the presence of software agents in work situations causes a job shift from a control activity to a supervisory control activity (Sheridan, 1984). For instance, software
agents such as flight modes in modern aircraft cockpits tend to change the nature of pilots' jobs. The choice of a model for the description of an agent is rather difficult since we need to represent information gathering, processing and control, as well as to provide conceptual tools for taking into account factors such as cooperation or delegation. We have adopted a mixed approach to modeling agents that is both structuralist and interactionist. Our agent model is based on Newell and Simon's classical model (1972). It was first developed during the MESSAGE project of analysis and evaluation of aircraft cockpits (Boy, 1983; Boy and Tessier, 1985).

Following Minsky's definition of an agent, an agent is itself a society of agents (Minsky, 1985). Thus, the agent model includes a supervisor sub-agent managing three other types of sub-agents called channels, that are receptors, effectors and cognition (Figure 4). Each sub-agent may act autonomously accomplishing tasks that were required by the supervisor agent or wait for new requirements from the supervisor. The internal function of an agent is based on a cognitive architecture usually called a blackboard (Nii, 1986). Each channel exchanges information with the supervisor. The supervisor may sequence tasks for the channels, or leave them to work independently as active demons. It may use a performance model such as Rasmussen's (1983). The notions of automatic and controlled processes, introduced and experimentally observed by Schneider and Shiffrin (1977), have been implemented in the supervisor of the generic agent of the MESSAGE system (Tessier, 1984). This allows the generation and execution of tasks either in parallel (automatisms), or in sequence (controlled acts). In our agent model, automatic processes involve specific knowledge modeled by a situational representation. In contrast, controlled processes involve knowledge represented by an analytical representation (see section 2.2 of this chapter).
2.1. Situational representation

The term situation is used here to characterize the state of affairs of the agent's local environment. A situation is described by a set of components called facts. Facts are causal entities that can be events, circumstances, stories or conditions. Formal notations are introduced here to show the reader both the complexity of the concept of situation and the extreme difficulty to elicit situation rationally. A generic fact will be noted \( f_i \) (meaning fact number \( i \)). At a given time, three basic types of situation will be distinguished (Boy, 1983):

(1) The real situation characterizes the "true" world. It is only partially accessible to the operator and must be interpreted as: "The real situation at a given time \( t \) is the fact set, which may be restricted to the local environment". The set is a priori infinite. It is noted:

\[ S_R(t) = \{ f_i / i = 1, \infty \} \]

(2) The perceived situation is a particular image of the local environment. It is the part of the environment accessible to the operator. It is characterized by incomplete, uncertain and imprecise components. It will be noted:

\[ S_P(t) = \{ \pi_i / i = 1, n \} \]

which must be interpreted as: "The perceived situation at a given time \( t \) is the set of facts perceived by the operator." This set is assumed to be finite. Each perceived fact \( \pi_i \) is the result of the mapping of operator \( P_i \) on a subset of the fact set:

\[ \pi_i = P_i ( \{ f_j / j = 1, p \} ) \]

In general, operator \( P_i \) is "fuzzy" in Zadeh's sense (1965). All the subsets of the fact set characterizing each perceived fact are also assumed to be finite.

(3) The desired situation characterizes the set of operator's goals. It will be noted:

\[ S_D(t) = \{ \partial_i / i = 1, m \} \]

which must be interpreted as: "The desired situation at a given time \( t \) is the set of the goals which the operator intends to reach." This set is assumed to be finite. Each goal \( \partial_i \) is the result of the mapping of the operator \( D_i \) on a subset of the fact set:

\[ \partial_i = D_i ( \{ f_j / j = 1, q \} ) \]

In general, operator \( D_i \) is "fuzzy". All the sub-sets of the fact set characterizing each goal are also assumed to be finite.

The real situation characterizes the local environment. Perceived and desired situations characterize the operator's short term memory. In the expert system terminology, the perceived situation is called the fact base (i.e., "perceived facts") and the desired situation is called the goal (and sub-goal) base.

**Situation patterns**

The concept of a situation pattern is fundamental. A situation pattern characterizes the operator's expected situation at a given time. We will say that a situation pattern is activated if it belongs to the short term memory. It will be noted:

\[ S_E(t) = \{ \mu_i / i = 1, l \} \]
which must be interpreted as: "The expected situation at a given time $t$ is the set of the situation patterns which are activated in the short term memory". This set is assumed to be finite. Each situation pattern $\mu_i$ is the result of the mapping of operator $\prod_i$ on a sub-set of the fact set:

$$\mu_i = \prod_i \left( \{ f_j / j = 1, r \} \right).$$

In general, operator $\prod_i$ is "fuzzy". All the sub-sets of the fact set characterizing each situation pattern are also assumed to be finite. In practice each fact $f_j$ is represented by a (numerical or logical) variable $v_j$, and its tolerance functions $\{ TF_{i,j} \}$ in the situation pattern $\mu_i$. A tolerance function (TF) is an application of a set $V$ into the segment $[0,1]$. The elements of $V$ are called "values", e.g. the values indicated by a numerical variable. $V$ is divided into three sub-sets: preferred values $\{ TF(v)=1 \}$, allowed values $\{ 0<TF(v)<1 \}$, and unacceptable values $\{ TF(v)=0 \}$. This model of a tolerance function is able to take into account logical variables, e.g. position of a valve or truth of a statement, and numerical variables, e.g. temperature or pressure. An ordinal scale has been defined in accordance with the above definition of the three sub-sets. This definition is appropriate where natural language concepts must be modeled and manipulated. In fact, a tolerance function is a membership function related to the concept tolerance. The major attraction of this representation is its ability to handle the vagueness inherent in categories of language, e.g. "the pressure $P_1$ is close to the maximum."

A situation pattern is an element of the operator's long-term memory. It is the result of a long period of learning. At each time interval, the operator's vigilance will be characterized by the number and relevance of the situation patterns "activated" in the short-term memory. For instance, in a monitoring process, the operator matches perceived facts with activated situation patterns. Reason (1987) calls this process: "similarity matching". At a given instant, if several patterns are candidate for a perceived situation, Reason's results show that the most frequent pattern is selected first. He calls this process: "frequency gambling." We also call this process "conflict resolution." We have defined the total process as situation recognition (Boy, 1987).

It seems reasonable to envisage that situation patterns (i.e. situational knowledge) are compiled because they are the result of training, which is a type of learning. The situational knowledge of an expert results mainly from the compilation, over time, of the analytical knowledge he relied on as a beginner (see section 3.3). This situational knowledge is the essence of expertise. It corresponds to skill-based behavior in Rasmussen's terminology. "Decompilation", i.e. explanation of the intrinsic basic knowledge in each situation pattern, is a very difficult task and is sometimes impossible. Such knowledge can be elicited only by an incremental observation process.

### 2.2. Analytical representation

When a situation is recognized, problem solving generally follows. A problem is initiated by an event (i.e., perceived or desired situation) which necessitates that the operator allocate intellectual resources. These intellectual resources can be represented by structured objects and inference rules, such as: IF <Conditions> THEN <Hypothesis> AND <Actions>. Note that if several situation patterns have been matched and kept as "interesting to be
considered", then several hypotheses are suggested. Thus, problem solving can be multi-hypotheses. Generally, each hypothesis defines an "extension" ("world" or "viewpoint"). Consistency maintenance in each extension and between generated extensions is a central mechanism. Such a mechanism has been formally described and developed by De Kleer (1986). De Kleer called this mechanism an "Assumption-Based Truth Maintenance System". Thus, generally, analytical reasoning involves a problem solver and a consistency-maintenance mechanism.

The problem solver provides inference. Inference can be performed in forward chaining (from data to conclusions) or in backward chaining (from goals to elementary actions). Human problem solving has been defined as "opportunistic" by Hayes-Roth and Hayes-Roth (1978), i.e., either forward or backward chaining is possible at any given time. Moreover, the human being tends to reason using knowledge already structured in knowledge chunks. Each chunk is connected to one (or several) situation pattern(s) and defines a context. In multi-hypotheses problem solving, each extension is said to be working in its context. The corresponding representation will be called analytical representation. This notion will be further developed in section 3 of this chapter. Analytical knowledge can be decomposed into two types: procedures or know-how, and theoretical knowledge.

Procedural knowledge can be represented in the form of instantiated plans. It can generally be translated into production rules and propositional logic. Thus, it is also the result of instantiation of more general knowledge. We here use interchangeably the terms instantiated knowledge, surface knowledge and procedural knowledge. The knowledge in operation manuals is almost exclusively of this type. Rasmussen (1983) relates rule-based behavior to this level of knowledge. Procedural knowledge is the easiest to acquire. It may be more or less easy to formalize.

Another type of processing, called knowledge exploration, is based on so-called deep knowledge (Van der Velde, 1986). From a formalization viewpoint, this kind of knowledge is not always available in all disciplines and corresponds to the theoretical basis of the domain in which one is working. Several models (and sometimes theories) may be used to describe or reason about an engineered system. In fault diagnosis, for example, one might develop a structural model of the system to be diagnosed and then decompose it into its basic components and the relationships among those components. Sometimes, a graphical model is necessary. A functional model will permit simulation of the behavior of the various components. A causal diagram may then be drawn to establish the various connections between the properties of the components. The associated type of reasoning will call on a mechanism for instantiation of general knowledge. This knowledge is organized according to the structure of the field of the problem to be solved: its topology, its functionality, its causality, and other aspects such as the hierarchy of objects and the autonomy and heteronomy of various parts of the system. Deep knowledge is the basis of Rasmussen's knowledge-based behavior.

3. The SRAR Model

Using software agents is a situated activity. We proposed a model called the situation recognition and analytic
reasoning model (SRAR). This model has been applied to various dynamic situations and problems.

3.1. The HORSES experiment

In this section, we describe a knowledge elicitation experiment that was carried out at NASA Ames Research Center (Boy, 1987). The Orbital Refueling System (ORS) used in the space shuttle to refuel satellites was chosen as an example of a system to be controlled. Three software agents were used to implement knowledge elicitation experiments: an ORS simulator, a malfunction generator, and a fault diagnosis expert system, called HORSES (Human—ORS—Expert System). These software agents were concurrent processes, communicating through a shared memory. The malfunction generator generated simulated malfunctions for the ORS, and introduced them at the appropriate times into the ORS simulation. HORSES was composed of a set of software agents having a graphical appearance on IRIS 1200 graphic system. HORSES has been incrementally improved through an incremental knowledge elicitation process.

Two groups of four pilots participated in the experiment. The first group was naive in the ORS operations whereas the second was knowledgeable in that domain. Sessions were videotaped for subsequent analysis. Pilots were asked to oversee the transfer of fuel from one container to another and were given simulated display and controls as they were currently implemented in the ORS. Each pilot was involved in five sessions of 3 hours, performed on 5 days. The first session was training only. Fifteen runs were proposed to pilots during the three following sessions. For each run there was a probability of encountering a failure. Faults were leaks, sensor biases, and avionics failures. During these experiments, two types of operator assistance were tested. The first type corresponds to the use of a classical paper operation manual. The second type corresponds to the use of HORSES connected to the system being controlled. It helped by determining possible situations (contexts) that the operator then selected. HORSES was able to start a process of analytical reasoning, interacting with the operator. It asked the same questions, requested the application of the same procedures, and gave the same diagnoses as the operation manual. The operator answered the questions, applied the procedures or not, and could stop the reasoning at any time. The last session was a debriefing during which individual interviews were performed. Experiments involved subjects being asked to think aloud while interacting with the ORS. Protocols (the chronological history of all the pilot’s actions and system responses) were transcribed for later analysis. Verbal reports were very useful to understand what variables and processes were used when pilots performed diagnosis tasks, i.e., the analytic reasoning. But, as the situation recognition process is very interconnected with the diagnosis inference, verbal information from the pilots during the process control task was also very useful for identifying situation patterns. Verbalizing information is shown to affect cognitive processes only if the instructions require verbalization of information that would not otherwise be attended to (Ericsson and Simon, 1980). Thus, the experimenter did not ask any question during operations. The results are based primarily on protocol analysis. They rely on the use of three criteria: the importance of an event during the run, its frequency, and the number of pilots dealing with it (Robert, 1985). In
analysis, the following three topics were considered: operation manual versus software agent; beginners versus experienced operators; and the derivation of SRAR model (see below) for fault identification.

### 3.2. Derivation of the SRAR model

#### Operation manual versus software agent

Pilots using the operation manual could be separated into two groups. The first group used the procedure-following mode: "Pressure P1 is not normal according to what it should be on the graph, therefore I take the procedure to diagnose the fault." They recognized critical situations by matching observations to situation patterns included in the operation manual. They used the operation manual very carefully and diagnosed faults according to the knowledge available in the operation manual. Detection and subsequent reasoning were very straightforward. The second group used an intuition-based mode: "Losing pressure P1, I have a leak in tank 1, close immediately valve 4." They recognized situations not necessarily included in the operation manual. The first group was slower than the second in diagnosing faults when the fault was easy to detect. The second group failed more often. The first group always applied procedures, whereas the second was generally problem-solving-driven.

Pilots using the software agent as an intelligent assistant could be divided into two groups: situation recognizers and diagnosis processors. In the former group, pilots used the software agent as an assistant in situation recognition. They tended to rely on the advice of the software agent about the situation. However, they also used the explanation facility very often. They generally used the software agent in a forward-chaining mode. In the latter group, pilots used the situation recognition agent after having detected a problem by themselves. They acknowledge the results given by the situation recognition agent, and started the corresponding analytical processing. These pilots generally guessed a diagnosis and asked the software agent to verify it in a backward-chaining mode. It was observed typically that the situation recognizers are inclined to use a procedure-following mode, whereas the diagnosis processors correspond to an intuition-based mode.

#### Beginners versus experienced operators

Experienced people monitored the system being controlled with more sophisticated situation patterns than the beginners. Once they recognized a situation, experts implemented short sequences of tests. The analytical reasoning they employed at this stage was minimal, by comparison with beginners. Thus, experts and beginners recognize differently situations and subsequently implement associated analytical reasoning differently.

The situation patterns of beginners are simple, precise and static, e.g. "The pressure \( P_1 \) is less than 50 psia". Subsequent analytical reasoning is generally major and time-consuming. When a beginner uses an operation manual to make a diagnosis, his behavior is based on the precompiled engineering logic he has previously learned. In contrast, when he tries to solve the problem directly, the approach is declarative and uses the first principles of the domain. Beginner subjects were observed to develop, with practice, a personal procedural logic (operator logic), either from the precompiled engineering logic or from a direct problem-solving approach.
This process is called *knowledge compilation*. Conversely, the situation patterns of experts are sophisticated, fuzzy and dynamic, e.g. “During fuel transfer, one of the fuel pressures is close to the isothermal limit and this pressure is decreasing”. This situation pattern includes many implicit variables defined in another context, e.g. “during fuel transfer” means “in launch configuration, valves $V_1$ and $V_2$ closed, and $V_3$, $V_4$, $V_7$ open.” Also, “a fuel pressure” is a more general statement than "the pressure $P_1$." The statement “isothermal limit” includes a *dynamic* mathematical model, i.e. at each instant, actual values of fuel pressure are compared *fuzzily* (“close to”) to a time–varying limit $[P_{isoth} = f(Quantity, Time)]$. Moreover, experts take this situation pattern into account only if “the pressure is decreasing," which is another dynamic and fuzzy pattern. It is obvious that experts have transferred part of analytical reasoning into situation patterns. This part seems to be concerned with dynamic aspects.

Thus, with learning, dynamic models are introduced into situation patterns. It is also clear that experts detect broader sets of situations. First, experts seem to fuzzify and generalize their patterns. Second, they have been observed to build patterns more related to the task than to the functional logic of the system. Third, during the analytical phase, they disturb the system being controlled to get more familiar situation patterns which are usually static: for example, in the ORS experiment, pilots were observed to stop fuel transfer after recognizing a critical situation.

By definition, a pair $\{S_i, A_i\}$ is called a SRAR chunk (Figure 5). Beginners have small, static and crisp situation patterns associated with large analytical knowledge chunks. Experts have larger, dynamic and fuzzy situation patterns associated with small analytical knowledge chunks. The number of beginner chunks is much smaller than the number of expert chunks.

### 3.3. SRAR chunks as interface components

The following generalizations can be drawn from the HORSES experiments.
First, by analyzing the human–machine interactions in the simulated system, it was possible to design a display that presented more polysemic information to the expert (e.g. a monitor showing the relevant isothermal bands). Polysemic displays include several types of related information presented simultaneously and are readily understandable to experts because the presentation is derived from their situation patterns. This improved user and system performance. Second, the HORSES agent achieved a balance in the sharing of autonomy. The original system designer did not anticipate the way that the operators would use the system, but letting them have indirect control over the assistant allowed them to utilize what they had learned to do well.

SRAR can be used to design interface agents that would help human operators to anticipate the evolution of dynamic systems. It is intended to provide help to find the best compromise in the design of interface and procedures. Taking the example developed in section 3.1 above, experts were observed using a diagram displaying two curves to follow the evolution of the pressures with respect to the quantity of fuel transferred. These curves were graphical representations of two ideal models, namely isotherm and adiabatic (Figure 6). Experts followed the evolution of the point (Q, P) represented by the quantity transferred Q and the corresponding pressure P within the domain defined by these two curves.

Paper-based technology makes it possible to use easily complex patterns such as the one displayed by the indicator (Q, P). Operations are rather difficult—and most of the time impossible—due to the fact that the human operator needs to use several sheets and manuals at the same time. In books, situation patterns are very simple and crisp conditions that can be written as titles to index pages of analytical reasoning. Expert situation patterns are constructed by experience. They cannot be stored in books because they are far too complex and, even if this storage was possible, human operators would have immense difficulty in retrieving them. When integrated in the user interface, they are easy to use because they are used as monitoring displays. The next section proposes a methodology to approximate expert situation patterns and subsequent user-centered interface components.

Figure 6. Construction of the indicator (Q, P) as an expert situation pattern.
3.4. Constructing Situation Patterns: the LSS model

The construction of situation patterns is a learning mechanism. Learning can be viewed as a bi-directional transformation: specialization and generalization. Most research efforts in automatic symbolic learning (Kodratoff and Tecuci, 1987; Michalski et al., 1986) have focused on generalization. In expertise, however, specialization plays a major role. An expert is not necessarily "intelligent", in the sense of possessing highly sophisticated and general knowledge, but possesses specialized situation patterns for his domain of expertise. These patterns have been built with experience. Generalization consists in building disjunctive chunks, while specialization consists in building conjunctive chunks.

The learning by specialization/structuring (LSS) model is one of a class of symbolic learning processes which are based on explanation (Boy and Delail, 1988). The explanations come from experts in the domain under consideration, or from external observers. At the beginning of the learning process, the input is procedural knowledge. The learning process can be decomposed into three phases:

- acquisition of procedural knowledge (explanations) and construction of a suboptimal knowledge base (analytical and situational);
- transfer (specialization) of certain elements of the analytical knowledge into situation patterns, as the result of experimentation;
- restructuring of the analytical knowledge as a function of the new patterns, i.e. new "chunks" are created.

In some ways, this method is similar to the chunking method developed in the SOAR system (Laird et al., 1987). The difference is that here, the process of chunking works explicitly on two types of knowledge, i.e. analytical and situational. In addition, chunks based on the concept of situation patterns open the possibility of parallel processing of knowledge. This parallelism is concerned not only with asynchronous processing of situational knowledge, but also with the processing of several analytical reasoning processes simultaneously. As in SOAR, the specialization process reduces the search.

**Initial suboptimal knowledge base**

The first phase of the process consists in building a suboptimal knowledge base (KB) which will be refined by experimentation. The situation patterns in the suboptimal KB are small and crude. These are the Demons which suggest subsequent analytical reasoning. The initial analytical KB describes the functioning of the system to be controlled. It is divided into several subsets of rules, each of which contains a large number of rules (SRAR beginner in figure 4). We have taken the following example from an operator assistance (Boy and Delail, 1988; Mathé, 1990) project in space telemanipulation. The situation patterns and the analytical rules are written here in the form:

\[
\text{IF } \langle \text{premise(s)} \rangle \quad \text{THEN } \langle \text{hypothesis} \rangle \quad \text{AND } \langle \text{action(s)} \rangle
\]

Under these conditions, an example from the initial suboptimal knowledge base is:

**Situation pattern:**
IF \{ TETA1 > 60 \}
THEN suggest subset-of-rules(A)
AND start-the-analytical-reasoning

Analytical knowledge:

IF user-acknowledgment
THEN subset-of-rules(A)
AND reset user-cross-check

The first analytical rule functions in backward-chaining mode if its consequence (hypothesis) is suggested by a situation pattern, as is the case in the above example. For simplicity, we will restrict discussion to this case. If, however, a situation pattern leads to a new set of values for a set of facts, then a forward-chaining mechanism would be used. As the analytical reasoning progresses, the corresponding knowledge is instantiated and thus may be represented in tree form. The instantiated knowledge from the above example is summarized in the following graph (Figure 7).

**Figure 7. Analytical search tree.**

### Specialization

We have already shown (Boy, 1987b) that as operators gain experience in using the system with which they are charged, their situation patterns become more complex, more dynamic and more numerous (SRAR expert in Figure 4). In the above example, we found that all the experts considered the node Preceding-movement very early in the process of situation detection. This element of knowledge can thus be transferred into the situational part of the knowledge base. Thus, specialization of the knowledge base proceeds as follows. (1) Detection of the values or instances of the node to be transferred. Here, the fact Preceding-movement can take the values Forward or Backward. (2) Identification of the initial situation pattern from which the node was deduced, and construction of \( n \) new patterns corresponding respectively to \( n \) values or instances of the transferred node. Each new situation pattern is a conjunction of the contents of the initial pattern and a particular instance of the transferred node. In the above example, the new situation pattern is:

IF \{ TETA1 > 60 \\
AND Preceding-movement (Forward ) \}
THEN suggest subset-of-rules(A1)
AND start-the-analytical-reasoning

IF \{ TETA1 > 60 \\
AND Preceding-movement (Backward ) \}
THEN suggest subset-of-rules(A2)
AND start-the-analytical-reasoning.
(3) Restructuring of the analytical knowledge base. As \( n \) situation patterns have just been created in step 2 of the learning process, the analytical KB can be redivided into \( n \) corresponding subsets. This transformation results in the duplication (\( n \) times) of the part of the analytical KB which was initially upstream to the transferred node. In a more schematic form, let \( T \) be the node transferred, and \( I_1 \) and \( I_2 \) the values or particular (relevant) instances of \( T \). \( S \) is the initial situation pattern, and \( A, B, C, D, E \) are other nodes (Figure 8).

**Figure 8. The specialization process.**

**Structuring**

Step 3 of the specialization process requires restructuring of the part of the graph upstream from the transferred node. The concept of structuring of the analytical KB introduces the concept of hierarchy among the subsets of rules. In the above example, the subsets \( A_1 \) and \( A_2 \) were constructed from the initial subset \( A \). \( A_1 \) and \( A_2 \) contain the same subset of nodes (\( B \) and \( C \)). Thus, these nodes are common to the disjunctive subset (\( A_1 \lor A_2 \)). The process of restructuring thus leads to the extraction of the intersection of the disjunctive subsets of rules elaborated by the specialization process, and to the contextual connecting of this intersection to "specialized" disjunctive subsets (Figure 9).
Figure 9. The structuring process.

Figure 10. Commonality among the subsets \{KB_{ai}\}. For example, KB_{a2} is common to KB_{a4} and KB_{a5}.

Implementation of this technique in the form of rules leads to the first rule of the analytical reasoning (for A1, for example):

\[
\text{IF user-cross-check} \\
\text{AND Preceding-movement (Forward )} \} \\
\text{THEN suggest subset-of-rules (A1)} \\
\text{AND reset user-cross-check} \\
\text{AND suggest subset-of-rules (A1 v A2)}
\]

and the subset (A1 v A2) starts with the same type of rule in backward chaining. If this first rule does not contain an action of the type suggest subset-of-rules(...), then the corresponding subset of rules is called a terminal subset. In all other cases, there will be a common subset. Each subset of rules has a priority corresponding to its degree of commonality. Thus, various degrees of commonality exist. This concept (degree of commonality) introduces the notion of recursiveness on several common subsets of rules (Figure 10). It should be noted that a common subset of rules may be activated directly by its own situation pattern, if such a pattern exists.
3.5. Problem Statement and Problem Solving

The situational/analytical distinction induces another distinction: the statement of a problem and its solution. We have seen that problem solving may be of two kinds: using know-how and exploring theoretical knowledge. Until now, most research in AI has addressed problem solving. We consider here the well-known maxim "a problem well-stated is a problem already half-solved." In effect, an expert problem-solving method has a refined and sophisticated problem statement, and an easy-to-implement problem solution. However, the problems that we propose to study in this work are poorly defined and cannot be expressed in algorithms. It should be noted that a problem may be badly stated a priori but its statement may be refined by experience.

We have seen that it is very difficult, and sometimes impossible, to elicit expert knowledge. In contrast, it is easy to ask an expert to give a tutorial. If this tutorial is taken as analytical knowledge, this knowledge can be tested incrementally and used to generate situational knowledge progressively by experimentation. In the same way, the statement of a problem may evolve with experience, and asymptotically approach a genuine skill. Certain experts, when faced with a problem, function by stimulus response with virtually no reasoning remaining. Because it corresponds to the mechanisms used by human operators, the situational/analytical distinction can be used to implement intelligent assistant systems which improve the overall performance of man–machine systems. In particular, an expert system built using this model enables the developer to design an initial knowledge base which is essentially analytical and, by using a complementary simulation tool, to construct the situational knowledge base incrementally by experimentation.

We have already defined a situation pattern as a set of constraints bearing on a finite set of facts. We know that a person's behavior becomes more and more rapid as he accumulates such "patterned" knowledge. For example, consider the task of driving a car. The more a driver encounters cases of a particular type of situation, the more he will develop situation patterns which thereafter will be activated by reflex when a real situation of that type is perceived. In other words, he will not have to think for a long time to make a "right" decision. The corresponding learning process could be described as the compilation of patterns, i.e. this process creates automatisms.

If we try to explain the observed behavior of the driver by a computerized model separating out analytical and situational reasoning and the process of learning by specialization and structuring, it quickly becomes apparent that as the number and complexity of patterns increases, the calculation time also increases. This is exactly the opposite of what is observed in human behavior. This phenomenon has been explained in (Tambe & Newell, 1988). This phenomenon can only be explained by the parallel processing in human mechanisms of cognition and perception, as opposed to the essentially sequential functioning of computers, up to the present.

4. The block Representation

This section presents the knowledge block representation that was developed to support the SRAR model and consequently the LSS model. Such a
representation was designed to support the incremental construction of situation patterns from analytical knowledge and experience. Consequently, the knowledge block representation is commonly used as a mediating representation to help design teams eliciting operative models from users. In addition, knowledge blocks are declarative entities that can be linked whenever necessary to simulate (operational) procedural knowledge (Boy, 1989).

When beginners start to operate a particular system, they start learning formal procedures which they will improve incrementally over time. The procedural knowledge improves by augmentation of the knowledge base, and by transformation of the various entities of available procedures into more contextual procedures that are better suited for routine use (see section 3 of this chapter). Results of this transfer lead incrementally to more operational knowledge. Such a knowledge compilation mechanism is similar to those of Anderson (1983) and Rosenbloom (1983). The main claim of our approach is that contextual knowledge can be gained through experimentation. Thus, computers can provide experimental support for an incremental contextualization of knowledge.

In this section, we present a new approach to knowledge elicitation that utilizes existing procedures and modifies them by introducing the concept of abnormal conditions. It is based on a block representation. Its advantage in eliciting knowledge is its naturalness to humans. The resulting knowledge bases are similar in form to technical and operation manuals. Furthermore, the block representation is very convenient for incrementally building operational knowledge. The resulting system uses on-line user requirements and suggestions either to reinforce current procedures in cases of success or to generate new recovery procedures in cases of failure. This allows the system to provide helpful responses, even when no robust user model is available.

4.1. Knowledge block definition

A block can be described as a set of goals, a set of conditions, and a set of actions to achieve the goals. Conditions are decomposed into triggering preconditions, abnormal conditions and contextual conditions (Figure 11).

![Figure 11. Block representation.](image-url)
4.2. Context versus triggering preconditions

As we have already seen in the previous section, situation patterns are constructed incrementally by experimentation. In a situation pattern, we distinguish between contextual conditions and triggering conditions. In section 1.3, context has been already described as a set of persistent conditions that remain valid for a certain period of time. Thus, when such conditions are persistent, it is not necessary to trigger them at any time. We will say that contextual conditions are contributing factors to the activation of a situation pattern. In contrast, there are other preconditions that need to be triggered to actually start analytical reasoning.

For instance, if somebody is solving a problem in a given environment, there are a lot of tacit preconditions which are "obvious" (e.g. constant) in that environment. These preconditions are included in the contextual conditions. Given this definition, if a system is in a context, reasoning can be done on the set of blocks belonging to that context. Thus, triggering preconditions can be made much simpler, leading to faster pattern matching.

Contexts are organized in hierarchies. This facilitates the organization of the block knowledge base. For example, in aviation, a flight is generally organized into phases, which are decomposed into subphases and finally each subphase is described in the form of knowledge blocks or procedures (Figure 12).

![Figure 12. Simple hierarchy of contexts.](image-url)
In the context of a flight, there are several subcontexts (such as before take-off and take-off). In each terminal context (e.g., take-off), a set of blocks has to be performed. The corresponding block representation of the take-off context of blocks is given in figure 13. If everything is normal, the pilot accelerates the plane up to a decision speed (called $V_1$) after which he will not be able to stop the plane in case of a major incident. He executes the first block Before $V_1$ (block A in figure 13). The block Before $V_1$ is valid in the context take-off. It is triggered when the precondition Brake release and Maximum power are satisfied. The goal of this block is to reach the speed $V_1$. A variety of actions can be offered by the block. Some actions can be designed as resources to reach the goal. Some other actions are only loosely connected to the goal. If no abnormal condition occurs, he executes the blocks Before rotation (block B) and Before $V_2 + 10$ (block C). If an abnormal condition is satisfied during the execution of the block Before $V_1$, then he can decide to execute the block Stop the plane (block D). The notion of context of blocks is mutually inclusive, i.e., a block is included in a context that itself is included in a more general context, and so on. For instance, Figure 13 presents a context of blocks that is a block with the following characteristics:

- triggering condition: the triggering condition of block A;
- goal: goal of block C;
- abnormal condition: abnormal condition of block D.

### 4.3. Abnormal conditions and non-monotonic reasoning

The abnormal conditions of a block can be satisfied or not satisfied. When a set of actions is no longer applicable to the current situation, the situation is said to be abnormal for this block and an abnormal condition corresponding to this situation must be attached to the block. Abnormal conditions can be associated with the entire block or with a specific set of actions. For instance, during the execution of a navigation

---

**Figure 13. Context of knowledge blocks.**

![Diagram of knowledge blocks context](image-url)
procedure in an aircraft, a cabin depressurization in flight is an abnormal condition which leads to the application of a recovery procedure.

The execution of a block consists in performing its actions and controlling the satisfaction of the corresponding abnormal conditions. Abnormal conditions can be of two types: weak abnormal conditions that, if they occur, will cause an exit from the current block towards another block in the same context; and strong abnormal conditions that, if they occur, will cause an exit from the current block towards another context of blocks.

For instance, a blown bulb in an aircraft cockpit may be a weak abnormal condition that does not necessitate changing the context of the flight. Conversely, an engine shut-down will cause a radical change of context, i.e. the pilot will adopt a different strategy (fly to the closest airport, for example) and apply the appropriate recovery procedures.

An example of a context of blocks is represented in Figure 13 in the form of a flow diagram. Let \{A, B, C, D\} be a set of blocks having the same contextual conditions. Arrows represent links. Links between blocks are identified by numbers, from 1 to 7 in Figure 13. For example, A and D are connected by link 5 when a specific abnormal condition of block A is satisfied. Block A is called the "root" of the context of blocks. Notice that a context of blocks is a block itself. In our example, triggering preconditions of the context block are the triggering preconditions of block A, its abnormal conditions are those of block D, and its goal is the goal of block C.

In the previous aviation example, blocks A, B, and C stand for procedures Before V1, Before Rotation and Before V2+10. In normal situations, blocks are organized and processed in a tree sequence, i.e. A->B->C. We will call the resulting process *linear browsing* of a set of blocks. Abnormal situations interrupt this linear sequence to branch onto other blocks (generally called recovery procedures). In our aviation example, let us assume that, when applying the procedure Before V1, the speed indicator shows an unacceptable value (weak abnormal condition 5). The pilot has to use procedure D, that is, to monitor the thrust, except if the speed is lower than a given threshold (strong abnormal condition 7). In this case, the pilot has to change the context and Stop the plane. If the speed is within an appropriate range, then procedure D succeeds and the pilot can apply procedure B (link 6). We will call the resulting process *nonlinear browsing*, i.e. A->D->B->C.

4.4. Incremental construction of blocks

This representation is very convenient for representing knowledge that is needed to perform tasks which have to be performed to accomplish given operations on an engineered system. Usually, task analysis is performed by operations engineers who design and develop operation manuals that have to be revised often. This representation allows for declarative programming by incremental decomposition.

Knowledge design starts from raw blocks described from a high-level (macroscopic) point of view. A block can be decomposed into a network of blocks which will inherit the context of the initial block (Figure 14). This will create layers of contexts of blocks which can be browsed easily. Furthermore, as a context of blocks is a block itself, the lowest granularity of knowledge is being represented in the...
same way as the highest. This property is very interesting from a knowledge-acquisition point of view because the level of description of some parts of a domain, a task or a function may be different according to current available expertise and knowledge.

A situation pattern is a problem statement that leads to analytical reasoning on a context of blocks. The situation pattern of a context of blocks is composed of the minimal set of preconditions of the blocks characterizing the context. "Minimal" expresses the fact that, if a precondition is expressed in two different blocks of the same context, it will be taken only once in the situation pattern. Blocks are then explored and executed to solve the problem stated by the situation pattern.

A knowledge block represents a cognitive function that is used by a human operator to perform a task. It enables the representation of how a human operator performs the task. In the first place, the block is instantiated with the knowledge corresponding to the prescribed task (what the operator should do). Once a first prototype is built, it can be used and assessed. The corresponding block representation can be modified according to the information elicited from the observation of the operator activity. In particular, contextual and abnormal conditions are generated by experimentation.

![Figure 14. Decomposing a block.](image)

5. Cognitive Function Analysis (CFA)

5.1. Cognitive function: Definition

According to the French tradition in ergonomics (Laville, 1976), a distinction is made between task (prescribed task) and activity (effective task). When a job needs to be performed, it can be decomposed into tasks. Thus, tasks (at any level) are the requirements of the job. When someone
performs a task, he/she produces an activity. This activity may differ from the original requirements, i.e., the effective task differs from the prescribed task. This is due to the fact that all situations are very difficult to anticipate when tasks are designed. A human operator facing a situation produces a specific activity according to the requirements, but may need to adapt the task to this situation. Cognitive functions are the representation of the mechanisms that produce activities (Figure 15).

![Cognitive Function Diagram](image)

Figure 15. A cognitive function (CF) transform a task (what is prescribed) into an activity (what is effectively done).

In the sense described above, cognitive functions are agents. From a purely pragmatic viewpoint, the term cognitive function is better suited to describe human cognitive agents (processes). The term agent denotes humans or machines. Cognitive functions denote the processes that they use to perform the tasks that are assigned to them.

At design time, these cognitive functions can only be speculated upon according to past experience, accumulated knowledge and know-how. When a new device is being designed, it is extremely difficult to anticipate how it will be used. The designer has a use in mind, but cannot imagine all the possible uses of this device. At evaluation time, these users can be inferred from the observation of real activities. In both cases, they enable the simulation of cognitive processes. We call cognitive simulation the simulation of the appropriate cognitive functions. Examples of cognitive functions in aeronautics include:

- clearance transmission;
- frequency change;
- rolling authorization;
- parking space allocation; and
- flight level allocation.

A cognitive function (CF) can be decomposed into smaller grain cognitive functions. In particular, a CF invoked for performing a specific task can be decomposed into two main CFs (Figure 16):

- the CF devoted to the interaction: it is related to the syntax used to perform the task;
- the CF devoted to the content of task itself; which is related to the semantics of the task.

An easy-to-use user interface usually results in simple CF interaction components. For instance, if the task is to program a flight plan using the current interface called the multifunction command and display unit (MCDU). Most pilots do not like to use the MCDU, e.g., they do not like its lateral location, its alphanumeric keyboard, its overloaded screen, its complex menu system, etc. We carried out a study to improve the current concept of MCDU (Boy, 1995). The pilot needs to push keys (e.g., a first interaction sub-CF is devoted to locating the right key and checking what the computer
response is), browse menus that can be more or less complicated due to the depth or recursion of these menus, etc. The interaction component (I-CF) of the overall CF can be complicated. CFs devoted to the content of the task are concerned with processes such as minimizing the distance between two points, satisfying a constraint imposed by Air Traffic Control (ATC), etc.

A complex cognitive function such as programming a flight plan using a MCDU can usually be decomposed into a graph of knowledge blocks. The first layer of blocks mainly represents TC-CFs such as presented in Figure 19. For instance, the Preflight TC-CF is decomposed into three TC-CFs Setting up, Flight plan preparation, and Performance (Figure 18). The Setting up TC-CF is then decomposed into two TC-CFs System status check, and Nav aids deselection. The System status check TC-CF is conditionally (If A/C Status page is not displayed) decomposed into four I-CFs Depress ‘DATA’ key, Select ‘A/C STATUS’, Check Database period of validity, and Check Clock/Date.

<table>
<thead>
<tr>
<th>TC-CF</th>
<th>I-CF</th>
</tr>
</thead>
</table>

Figure 16. Interaction and task content components of a cognitive function.
A cognitive function analysis of the flight programming task enabled the elicitation of eight primitive cognitive functions: browse, check, depress, enter, insert, modify, select and set. Each more complex cognitive function is a composition of these primitives.

### 5.2. Automation as cognitive function transfer

The development of new tools can facilitate the execution of such cognitive functions by taking over part of the job currently performed by humans. The cognitive functions that are purely devoted to interaction itself (I-CFs) could be usually directed towards the machine (Figure 20), and the cognitive functions that are purely devoted to the content of the task are directed towards the human. Transferring cognitive functions from humans to machines is also called automation. For instance, technology-mediated human-human communication can be greatly enhanced by directing tedious and time-consuming I-CFs towards the machine.

Note that TC-CF also can be transferred to the machine. I-CF transfer defines an automation that transforms a physical interaction into a more cognitive interaction. TC-CF transfer defines an automation that distributes the responsibility of the task between the human and the machine. The way TC-CF are understood by designers is crucial because a transferred TC-CF necessarily has limited autonomy. The way this autonomy is perceived by end-users is an important issue. Sometimes users may not anticipate this limited autonomy because the system (using the transferred TC-CF) functions well most of the time.
1. PREFLIGHT

**SETTING UP**
- **PRECONDITIONS**: None (initial agent)
- **ABNORMAL COND**: System status check
- **SUBGOALS**: None

**PRECONDITIONS**: None (initial agent)
- **SUBGOALS**: Display A/C Status on ND : A320-211
- **SECOND DATABASE STORED** Rtes, Rwys, Wpts, Navs
- **PERF FACTOR** CONTINUE

**SUBGOALS**: None

---

**CO RTE entry**
- **PRECONDITIONS**: Setting up completed
- **ABNORMAL COND**: None
- **SUBGOALS**: CO RTE entry

**Data checking**
- **PRECONDITIONS**: RTE entered
- **SUBGOALS**: None

**Wind entry**
- **PRECONDITIONS**: RTE entered
- **SUBGOALS**: None

---

**Wrong Present Position**
- **PRECONDITIONS**: The Rte is not stored
- **ABNORMAL COND**: Wrong Present Position
- **SUBGOALS**: None

---

**Abnormal Cond**
- **PRECONDITIONS**: FROM is not present AIRPORT
- **SUBGOALS**: None

---

**ABNORMAL COND**
- **PRECONDITIONS**: Desired rte not stored
- **SUBGOALS**: None

---

**ABNORMAL COND**
- **PRECONDITIONS**: Enter present LL coordinates
- **SUBGOALS**: None

---

**ABNORMAL COND**
- **PRECONDITIONS**: Enter (keyboard ?) navaid(s) to be deselected
- **SUBGOALS**: None

---

**Navaid Deselection**
- **PRECONDITIONS**: NOTAM indicating unreliable Navaid
- **SUBGOALS**: None

---

**Database cycle**
- **PRECONDITIONS**: Information between Date and Database validity
- **ABNORMAL COND**: Cycle database
- **SUBGOALS**: None

---

**F-PLN INITIALIZATION**
- **PRECONDITIONS**: Setting up completed
- **ABNORMAL COND**: None
- **SUBGOALS**: CO RTE entry

**Wrong PP**
- **PRECONDITIONS**: NO CO RTE EXISTS
- **ABNORMAL COND**: None
- **SUBGOALS**: None

---

**ABNORMAL COND**
- **PRECONDITIONS**: RTE entered
- **SUBGOALS**: None

---

**1. PREFLIGHT**

---

**Preferable to build the route on a screen with usual tools or to insert it with a keyboard as in the MCDU ?**

---

**Figure 19. Example of a network of blocks for the Preflight cognitive function in the MCDU example.**

---

**Human cognitive function**
- **I-CF**
- **TC-CF**

**Figure 20. Transfer of part of an interaction cognitive function (I-CF) to the machine.**
There are several goals that motivate automation. These goals can be classified according to the following list of goals adapted from the list proposed by Billings for the air transportation system (Billings, 1991, page 68):

- **safety**: to conduct all operations, from start to end, without harm to persons or property;
- **reliability**: to provide reliable operations without interference from environmental variables;
- **economy**: to conduct all operations as economically as possible; and
- **comfort**: to conduct all operations in a manner that maximizes users' and related persons' health and comfort.

Formalizing the transfer of cognitive functions is a means to better understanding and controlling automation according to a list of automation goals such as safety, reliability, economy or comfort.

In the MCDU experiment, it was decided to remove the alphanumeric keyboard and to introduce a larger navigation display (ND) screen and a direct manipulation facility such as a trackball to manage information on this screen. This is an alternative to the current one. We consequently developed a set of cognitive functions that are used by the pilots to program flight plans. The efficiency of these functions was assessed using a keystroke level model analysis (Card et al., 1983), performed with the help of operational people. It showed that the alternative to the current setup is much better, i.e., it takes much less time to access any planning function when one uses this alternative design.

### 5.3. Human factors criteria

Our objective is to make this automation human-centered. What is *human-centered automation* (HCA) from a cognitive function viewpoint? As stated earlier, CFA involves eliciting, constructing and chaining cognitive functions that are involved in a specific task. HCA is a step further as it involves a set of principles and guidelines that guide CF transfer from an agent to another, and help understand the repercussions of this transfer. These repercussions can be expressed in terms of new CFs created and new relations between agents. HCA principles and guidelines are supported by design rationale criteria expressed in terms of attributes of automation (Billings, 1991):

- **accountable**: it must inform the human operator of its actions and be able to explain them on request;
- **subordinate**: except in pre-defined situations, it should never assume command. In those situations, it must be able to be countermanded easily;
- **predictable**: it should be able to be foretold on the basis of observation or experience;
- **adaptive**: it should be configurable within a wide range of human operator preferences and needs;
- **comprehensible**: it should be intelligible;
- **flexible**: it should be tractable, characterized by a ready ability to adapt to new, different, or changing requirements;
Billings' attributes of automation can be reinterpreted as ranges of dimensions that enable the designer to direct CF transfer towards I-CF and TC-CF as shown in Figure 21.

Billings provided a framework for analyzing human-centered automation that is based on the analysis of monitoring functions and control functions. The more a machine is automated, the more users can be subject to boredom, complacency and competence erosion. The more users need to perform the work manually, the more they can be subject to fatigue and high workload. Billings's analysis showed that CF transfer influences psychological factors such as boredom, complacency, competence erosion, fatigue or workload. These factors can be used to assess the relevance of CF transfer. Billings provides a list of principles for HCA (Billings, 1991):

- the human operator must be in command;
- to command effectively, the human operator must be involved;
- to be involved, the human operator must be informed;
- the human operator must be able to monitor the automated systems;
- automated systems must be predictable;
- automated systems must also be able to monitor the human operator;
- each element of the system must have knowledge of the others' intent;
functions should be automated only if there is a good reason for doing so; and
automation should be designed to be simple to train, to learn, and to operate.

5.4. Elicitation method using a cognitive simulation

Cognitive functions are difficult to capture. The main idea of the method is to incrementally refine elicited cognitive functions in order to simulate cognitive interactions, in the use of the tool or system being designed. Cognitive simulations make it possible to test design alternatives. Cognitive function analysis (CFA) involves a cognitive simulation based on the block representation (cognitive designer part) and explicit interface artifacts (user part) (Figure 22).

The cognitive simulation is started using the block representation by specifying a starting context and satisfying (one or several) triggering condition(s) (Figure 19). There is a direct correspondence between the formal knowledge block and its interface artifact, i.e., an appropriate metaphor of its meaning. For instance, if a block represents the various actions to take in the before V1 context, its interface artifact will be all the relevant instruments and procedures to follow in this context. This interface artifact can be static or dynamic according to the level of detail that we would like to investigate. Standard hypermedia tools can be used to develop such external faces.

![Overview of the cognitive function database](image)

Figure 22. A view of a cognitive simulation using the CFA approach.

This simulation enables domain experts to assess the relevance and correctness of the cognitive function analysis. In particular, the level of granularity of the representation is essential in order to capture relevant and necessary
cognitive functions useful in air-ground communications. The CFA method consists of the following steps:

- define a first sub optimal set of contexts using classical elicitation methods such as interviews, brainstorming, terrain observation;
- develop the knowledge blocks that describe processes responsible for both pilots' and controllers' activities within these contexts;
- run a cognitive simulation and assess its relevance and correctness;
- refine current contexts and knowledge blocks; and
- run a new cognitive simulation and assess its relevance and correctness.

We claim that the cognitive simulation method enables the study of human-machine interactions. Other authors agree with this view (Woods & Roth, 1995). This is due to the fact that it has been intrinsically based on the concept of agent. The main difficulty is to determine the level of granularity that we would like to study. This involves tradeoffs and compromises. In the MCDU example, each cognitive function interface artifact was designed on a Hypercard card that gave to the user the illusion of a "real" navigation display screen. Using such a hypermedia tool enables the generation of active documents that can both simulate a real-world behavior and provide on-demand explanations.

This example has demonstrated that the use of such a cognitive simulation to rationalize design decisions can be extremely useful to explain why a part of a device has been specifically designed (Boy, 1995). This approach can be understood by designers and end users. Cognitive functions represented as knowledge blocks provide a powerful tool for mediating interaction between these practitioners. Furthermore, blocks enable incremental storage of design rationale in context, and easy retrieval and better interpretation of design rationale in context.

5.5. CFA improves the task analysis concept

CFA can be used to design procedures that are usually designed to improve performance, workload, safety, recovery from errors and failures in complex systems control tasks. A task analysis breaks down a task into sequences of “elementary” actions and control decisions. Since the role of the operator has evolved from direct control of a physical process (doing) to supervision and decision making of an automated system controlled by computers (thinking), task analysis has also evolved from physical to cognitive tasks. GOMS (Goals, Operators, Methods and Selection Rules) is a good model for analyzing the way goals and actions are organized for the accomplishment of a task (Card et al., 1983). A GOMS analysis is basically goal-driven, and one of the main difficulties of GOMS is to describe the goal structure. In addition, GOMS was developed to analyze and predict the total time for user task performance in human-computer interaction. The types of tasks that are usually analyzed using GOMS do not involve process control. In flying tasks, context and abnormal situations are important issues. GOMS does not enable the analyst to take into account context and abnormal conditions easily. In contrast, the block representation was originally designed to take into account these two types of attributes. Context and abnormal conditions enable the development of event-driven analyses. In addition, pilots have difficulties to explain their activity by decomposing their goals-subgoals. They usually prefer to tell stories in an event-driven way.
Another important problem with traditional task analyses is that they provide a theoretical (analytical) model that can be very far from the real world interface/procedures to be designed. CFA associates the block descriptions with the current interface artifacts incrementally designed. Thus, users can be directly associated within the design cycle and contribute to design decisions through usability tests of incrementally developed interface artifacts. CFA offers a mediating representation accessible both to designers and users. Such a mediating representation enables the users to improve their ability to generate hypotheses and immediately test them. In this sense, CFA is a good tool that facilitates abduction, i.e., heuristic generation of relevant hypotheses.

Sheridan’s recommendations for computer functions to aid supervisory control of physical processes are based on generic tasks (Sheridan, 1988). Abilities and limits of human operators and also the flow of information guide the development of such functions. CFA is very consistent with Sheridan’s approach to supervisory control functions. In addition, CFA proposes a function allocation framework based on usability assessments of discrete event simulations of an incrementally upgraded prototype. CFA enables designers to better anticipate the distribution of cognitive functions between humans and machines. They can test how various configurations affect cognitive attributes such those defined in section 5.3 of this chapter. NASA’s project MIDAS is another approach that develops and validates simulations of cockpit design tests the same kind of configurations by assessing workload and goal completion (Corker & Pisanich, 1995).

6. Conclusion and Perspectives

The research that went into developing the block representation for the design of software agents produced several contributions to the study of knowledge elicitation.

MESSAGE, first developed as a cognitive simulation to study the usability and safety of aircraft cockpits (Boy, 1983; Boy & Tessier, 1985), motivated an early attempt at the design of human agents and associated software agents. Subsequently, HORSES was developed to carry out a study of (human-machine) cooperative fault diagnosis, and the SRAR model was derived (Boy, 1987). The LSS model was derived to represent the incremental construction of situation patterns (Boy & Delail, 1988). It highlighted the need for a knowledge representation that makes it possible to model situated learning. The block representation was developed and first tested on a space telerobotic application. Considerably more information about the block representation can be found in Nathalie Mathé’s doctoral dissertation (Mathé, 1990). More details are available in the proceedings of various knowledge acquisition workshops (Boy and Caminel, 1989; Boy, 1990; Boy, 1991b), and a NASA Technical Memorandum (Boy, 1991d).

Computer Integrated Documentation (CID) has been designed and developed according to the principles and methods that were presented in this chapter (Boy, 1991c, 1992). CID is now a fully developed system that enables its users to index and retrieve documents in context. Several software agents have been developed that help users...
customize documents. In the beginning, agents are "inexperienced" and must rely on broad analytical knowledge—that is, nominal models of tasks and procedures that may be incomplete and incorrect. These models may be constructed through analytical knowledge elicitation methods (LaFrance, 1986; Wielinga et al., 1992). Learning mechanisms incorporated in CID software agents rely on the reinforcement of successful actions, the discovery of abnormal conditions, and the generation of recovery knowledge blocks to improve performance.

This chapter introduced a knowledge elicitation methodology for the design of software agents that is based on the incremental construction and refinement of cognitive functions. From a cognitive viewpoint, software agents can be defined as the result of the transfer of a cognitive function to the computer, and as a knowledge block with an interface artifact. Cognitive function analyses based on appropriate human factors criteria can be developed and simulated to help analyze, design and evaluate emerging software agents.

A decade of experience in the domain of knowledge elicitation for the design of automated systems has led us to believe that the software agent technology enhances human-centered design. In addition, people use computers for more and more complex tasks, often involving multiple programs and a variety of media. We need simple solutions to handle this increasing complexity. Object-oriented programming has become a real practice in industry. From a software development viewpoint, software agents can be programmed as an object augmented with an attitude (Bradshaw, 1994). Component programming is becoming a reality with new interoperable software architectures. In this perspective, software agents are components that can be exchanged between people to enhance knowledge elicitation.

Acknowledgments

Many ideas described here benefited from discussions with Nathalie Mathé, Rachel Israel, Erik Hollnagel, Jeff Bradshaw, Laurent Karsenty, Mark Hicks, David Novick, Alain Rappaport and David Woods. I would like to specially thank Helen Wilson for her comments on several drafts of this paper.

References

Boehm, B.W. (1988). A spiral model of software development and


