Zeitschrift: Technische Mitteilungen / Schweizerische Post-, Telefon- und Telegrafenbetriebe = Bulletin technique / Entreprise des postes, téléphones et télégraphes suisses = Bollettino tecnico / Azienda delle poste, dei telefoni e dei telegrafi svizzeri

Herausgeber: Schweizerische Post-, Telefon- und Telegrafenbetriebe

Band: 73 (1995)

Heft: 10

Artikel: How can we communicate with computers?

Autor: Allemang, Dean / Liver, Beat

DOI: https://doi.org/10.5169/seals-876008

ABSTRACTIONS: THEIR PURPOSE AND APPLICATION IN TELECOMMUNICATIONS

HOW CAN WE COMMUNICATE WITH COMPUTERS?

The successful deployment of any software system depends on its ability to communicate with its environment. In the case of Decision Support Systems, this means that the computer must be able to communicate with human decision makers. Since humans do not normally communicate in the same cumbersome language as a computer implementation, this means that the issue of finding appropriate abstractions to facilitate this communication is particularly important. In this paper we study the nature of such abstractions and show how they can be used to guide the design of a decision support system.

DEAN ALLEMANG AND BEAT LIVER, BERN

Decision Support Systems (DSS) come in a variety of forms, from very simple systems that provide access to data or perform simple calculations to complex systems that take over a considerable amount of decision making. In order to be useful, a DSS must provide some service that would be tiresome, difficult, or even impossible for a human reasoner to provide on his own. For example, a DSS can calculate the price of a proposed network, helping a designer to choose the most favorably priced solution, or it could index a data base of past failure cases, which stores far more cases of network failure than a human can possibly be expected to remember.

Abstractions and computation

The leverage that a DSS offers to a decision process can come from one of two sources: the DSS can have an algorithm that reliably solves some general class of problems, or the DSS can rely on problem-specific information that allows it to support solving a particular type of problem. The network cost calculator is an example of the former, while the well-indexed data base is an example of the latter. Any complex DSS is likely to use both sources of leverage. In [7] we describe how for many practical problems, it can be proven that no fast, general algorithmic solution is likely to exist. For this reason, we concentrate on solutions that rely on specifics of certain problems for their decision support leverage. We will refer to the collection of information about how to proceed in a specific situation as *knowledge*. Knowledge about a particular problem-solving domain usually results from a communication with an expert in that field. The accessibility of a large amount of knowledge about a field is what distinguishes an expert from a novice. The knowledge-based part of a DSS now looks like figure 1; the computer system must communicate with an expert in order to represent the knowledge of the domain. The computer system then supports the decision maker (the end user) by making use of this knowledge to exchange useful information with the user. This results in two bottlenecks for information flow in a DSS: between the expert and the computer and between the computer and the end user. The particular needs of decision support place special requirements on these two communication channels *(Fig. 1)*:

Cooperative problem solving

For interesting decision support applications, neither a human alone nor a machine is capable of solving problems with the required speed and reliability. In the simplest case (e.g. the network price calculator), the human is in complete control of the problem-solving activity, and the machine only provides passive consultation. In more complicated situations, the machine can also take over some of the direction of the problem solving (e.g. deciding what information to collect next). This requires the user and the computer to be able to carry on a dialog at several different points in the problem-solving process.

User responsibility

In most knowledge-intensive problem-solving situations, the person who makes a decision is responsible

for its correctness. This responsibility can be legal (as in the case of medical decision making, where incorrect decisions can be punished by lawsuit) or fiscal (as in the case of an equipment repair engineer, who will have to make another visit to the customer site if the first solution fails). A DSS that takes over a decision process must have a facility for allowing its human user to take responsibility for their combined actions. This means that the system designer (or the system itself) must be aware of the activities in the problem-solving process that require responsibility.

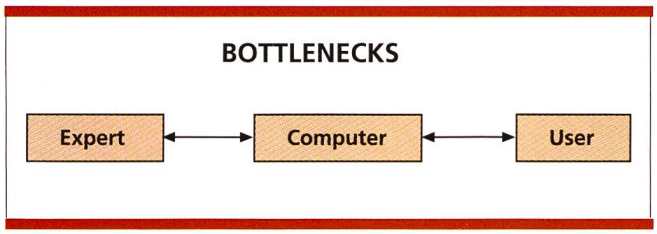


Fig. 1.
The bottlenecks of decision support applications; communication between expert and computer and between computer and user.

System maintenance

One of the main motivations for installing a DSS at all is its use as a repository of knowledge. As such, the knowledge covered by a particular system will change, either as new knowledge about the domain is obtained (e.g. new research about system performance under certain conditions) or the cope of the system is changed (a new telecommunications technology is installed). This means that there must be a flexible way to connect the system with its source of knowledge (an expert or an expert community).

In order to facilitate such a dialog, we need more than just a good graphic interface between the users and the machine – we need a vocabulary with which the humans and the computer can exchange information. The problem with finding such a vocabulary is that humans and computers are accustomed to dealing at very different abstraction levels. Humans have to deal with the real world with all its complexity; they have a set of abstract concepts they use to simplify the world for certain purposes *(Fig. 2)*. A computer normally deals at a much lower level, having to do with the mathematical formalisms on which its processing is based. Human-computer communication is facilitated when we can program the computer to understand the human's abstractions.
In Figure 2 we see an example of this. At the left is the real world with a global communication network. An *abstraction* is a concept that reduces the information in the real world to a manageable amount for some particular purpose. The two abstractions shown are a sphere (for purposes requiring information about the shape of the earth) and a graph (for purposes requiring information about the connectivity of the network). Abstractions of this sort are routinely used by human decision makers to control the complexity of the problems they solve. A human and a computer can also share these abstractions, as long as the abstractions are both understandable to the human and can be represented in the computer. An abstraction is represented in the computer by some programming construct or data structure; in this case, the spherical shape can be represented in the computer by the familiar formula, and the graph can be represented by a connection list. In order to use a computer program, it is up to the users to interpret the computer's abstractions in the real world; the better they are able to do this, the more effectively they can use the program.
Below we present three successful applications that illustrate this pattern – some real-world structures will be simplified with abstractions that are simultaneously comprehensible to a human and usable by the computer. We will then examine how various kinds of abstractions have made communication between experts or users and computer systems possible and why the resulting systems were successful. From this experience, we will examine how it is possible to construct and verify good abstractions, so that these successes can be repeated in a systematic way. In short, the key to successful decision support is to find appropriate abstractions to mediate the communication between human and machine, allowing them to work together to solve problems efficiently and effectively.

Successful applications to telecommunications

The following three systems were presented at the sixth annual Innovative Application of Artificial Intelligence conference. All three of them succeed sufficiently in overcoming the problems of cooperation, responsibility and maintenance to become profitable systems. We briefly describe each application and show what abstractions facilitated the communication between experts, users, and machines.

Unbilled calls at Pacific Bell

In November 1993 Pacific Bell put a system called EMCS (Expert Message Correction System) into operation [6]. Pacific Bell has millions of phone calls that cannot be charged routinely. This is usually a result of feature interaction. For example: How should a conference call be billed when one of the parties is a free-phone line? EMCS assists charge investigators to determine the liability for such nonroutine calls. EMCS does not try to bill all nonroutine calls; it simply acts as a filter for hard vs. easy cases. For cases that have straightforward answers, EMCS provides the answer automatically. If there is something especially irregular about the case, then EMCS forwards it to a human expert. This is a simple way to organize the interaction between user and machine – the machine works on the large amount of initial data and cuts it down to a size manageable by a human analyst. It also deals with the responsibility problem by trusting the program with simple cases and trusting it to distinguish these cases from difficult ones.

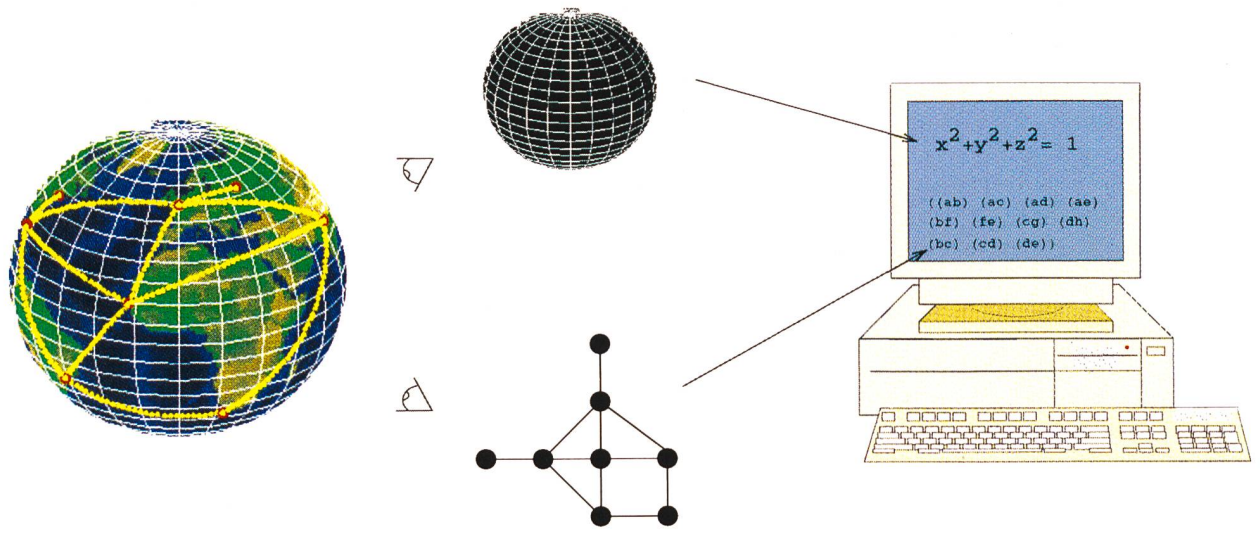


Fig. 2. The real world, two possible abstractions of it, and their representations in the computer.

The abstractions used in EMCS are rules. A rule is a condition/action pair: when the condition matches the current situation, then the corresponding action is taken. A rule for the conference call example might say 'if a conference call has exactly one free-phone party, then bill it as a normal conference call (with no free-phone party)'. The computer represents this abstraction in the form of an 'IF-THEN' rule, with conditions specified in terms of features and actions as billing activities. Abstractions in general simplify the real world; the rules in EMCS simplify the billing procedure by abstracting away the order in which the conditions have to be considered. Thus the expert who provides the rules need not consider this order when writing the rules. In this domain, the expert works with rules directly, so that maintenance of EMCS is quite simple – if the expert thinks of a new pattern of call features that can be classified easily, then it is a simple matter to write it down as a new rule. An abstraction is useful whenever the details it hides are not important for the problem. For rules in the billing domain, the order in which they fire is not important, since every pattern is independent; therefore, rules are a useful abstraction for the communication between expert and DSS.

Sales service support at Bell Atlantic

In October 1992, Bell Atlantic began putting SSNS (Sales Service Negotiation System) into operation [3]. SSNS supports sales personnel in advising customers about telephone services. Abstractions are represented in SSNS by both rules and *frames*. Frames are the representational unit in the computer for object-oriented modeling; the corresponding abstractions are categories of real-world entities. The frames are connected by *links* chosen from a fixed set of link types. These correspond to relationships between the corresponding entities. For example, in the SSNS case, the real-world entities are telephone service features and products (such as Answer Call, Call Waiting, or Call Forwarding), while the fixed set of relationships is the set {*restricts, interacts, depends*}, indicating that the services may not be supplied together, that they interact, or that one requires the other, respectively. For each known feature a frame is constructed, and for each relationship between features a corresponding link is made. Thus, the fact that Answer Call requires Call Forwarding is represented by constructing a 'depends' link from the frame for Answer Call to the frame for Call Forwarding.

SSNS is used in the following way: a customer who requires some services contacts a salesperson. The salesperson interviews the customer and enters his requirements in SSNS. SSNS informs the salesperson if there are any problems with these requirements, if contradictory services have been ordered, or if some required services have not been ordered. The salesperson then works with his customer to find a set of services that satisfy the customer's requirements. Since the SSNS system reflects the requirements of the product technology, feature combinations that are accepted by SSNS are feasible in practice.

SSNS leaves the responsibility for the final configuration of services to the salesperson – all it takes responsibility for is the interaction among the chosen services. The messages that SSNS can give to the salesperson are limited to the three types of service interaction. In this sense, SSNS communicates to the salesperson through a limited but abstract language.

In contrast to EMCS, the control of the problem solving in SSNS is flexible and is left to the salesperson, who can choose to respond to the messages in any order or manner desired. This means that the vocabulary with which the salesperson communicates with SSNS must be able to support this dia-

log. As long as the salesperson needs no more information than can be given by these three interactions (e.g., there is no need to know why a particular feature depends on another), this vocabulary will be sufficient.
The maintenance of the knowledge in SSNS uses the same abstractions as the user cooperation, namely the features and their interactions. The system maintainer is an expert on feature interaction. System maintenance does not require any new rules; the expert simply adds new frames for new features and new links for new relationships between the features.

Help desk at AT&T

In August 1993, AT&T deployed the system ESP (Expert Solutions Platform) [5]. ESP assists help-desk personnel in two of their major activities: determining the cause of customer problems and providing information about product features. These two functions are combined, because they both require extensive indexing based on the customer's current configuration and use. ESP uses *case-based reasoning* (CBR) to solve this problem. The principle behind CBR is simple: knowledge is represented as previously solved problem cases, and reasoning is performed by finding the stored case that is most appropriately similar to the current case and by reusing its solution. CBR has the advantage that one need not have a comprehensive understanding of all possible situations (in contrast to a rule-based or frame-based representation); instead, only the cases that have been observed need to be recorded and understood. This has the disadvantage, of course, that only cases that bear similarity to some earlier case can be solved. When a new case is encountered, the system can be easily extended, the new case (along with its solution) must simply be entered at an appropriate point in the case base.
During a consultation, the help-desk agent collects a description of the case from the customer (usually by telephone). The initial data collection includes data about the features of the customer's service and symptoms of the problem. The ESP system finds the stored case that best matches the current case and offers that solution to the help-desk agent. The help-desk agent takes the responsibility of providing the customer with an answer, based upon the information in the retrieved case. The labor is divided between the user and the system by leaving the final judgment of a case's appropriateness to the help-desk operator and by having the case base find a relevant case from a data base too large for a human user to manage. In this way, the case base is valuable as a repository of corporate knowledge; this means that when the experience gained by one help-desk operator is added to the case base, the performance of all help-desk operators (including operators who are new on the job) improves.
The maintenance of the ESP case base should, in principle, be automatic; each time a new case is solved, it is placed back into the case base (along with its solution) to be found during a future run (as appropriate). In practice, it is not so simple. In order to be able to find the case at an appropriate time, it is necessary to index the case base. To solve this problem, the team at AT&T organized the case base with *templates.* A template corresponds to a set of cases that share some commonalities. Cases are not entered automatically; *a case base engineer* figures out how to express each case in terms of the known templates and places it in the correct part of the case base.
These templates are abstractions; they represent commonalities among groups of cases. The templates are organized hierarchically with general templates high in the hierarchy and more specific templates farther down. The success of ESP depends on how well these templates index the case base.

Solutions and abstractions

Referring back to Figure 2, we can see the role of abstractions in all three of these solutions. An abstraction (like the sphere and the connectivity graph in Fig. 2) suppresses some detailed information about the real-world entity it describes. The power of a particular abstraction depends on the decision activity it supports; in the figure, the sphere abstraction of the world is useful for determining, say, the gravitational force at the Earth's surface, while the graph representation is useful for determining how many relay stations will be needed to send a message from one city to another. An appropriate abstraction, relative to some planned use, is one that suppresses unwanted and unnecessary details while retaining useful details. In all three of the above applications, the chosen abstractions have proven to be appropriate to their decision support problem.
The three examples also show how abstractions can range from the general to the specific:

- EMCS uses condition/action patterns to represent unusual billing patterns; this abstraction represents independent patterns, suppressing any ordering relationships among the patterns. These condition/action patterns are represented in the computer as rules and processed by a rule interpreter. As long as the expert understands rules and is willing to write them, this solution will support communication between expert and machine.
- SSNS uses frames to represent more specific abstractions that are particular to its domain of product feature interaction (namely, the three interaction types 'restricts', 'interacts', and 'depends'). These abstractions catalogue the entire range of feature interactions in these three categories, which are then represented in the computer as links between frames. Any distinctions not captured by these three categories are suppressed. These abstractions have wide applicability in this domain and constitute a language for communication not only between the expert and the machine but also between the machine and the salesperson.
- ESP is the only one of the three solutions that concentrates on developing abstractions that are appropriate for the specific application. The templates correspond to commonalities among sets of cases and ignore their individual differences. The abstractions are represented in the program by a tree of indices corresponding to the templates. In contrast to the interaction types in SSNS, these templates are dynamic – new templates are developed in response to the set of cases encountered during the use of the system. This means that the vocabulary with which the case base engineer communicates with the system develops as the case base grows.

The selection of good abstractions is essential to the success of these three systems; EMCS and SSNS depend on a good choice of *a priori* abstractions for their success, while ESP provides the possibility to add new abstractions to the system. In expert systems terminology, the activity of determining an appropriate vocabulary of abstractions is known as *knowledge engineering*. The knowledge in ESP is engineered in a flexible way, so that the knowledge engineering activity can continue after the system has been deployed. But even the ESP case base engineer has no guidance in how to construct these abstractions. There are no criteria that can determine when an abstraction is useful, nor are there even any general examples of templates that have been useful in the past; therefore, the case base engineer has no systematic way to generate useful abstractions. This is a dangerous situation, since, as we have seen, the success of a DSS depends on engineering a good set of abstractions.

The importance of good abstractions can be seen by examining a scenario that is commonly found during the development and deployment of a DSS. Often during DSS development, a set of abstractions is determined (much as the ESP team has determined a set of appropriate templates). These abstractions, which facilitate communication between expert and computer and between computer and end user, also serve to facilitate communication directly between the expert and the end user. The original motivation for a knowledge-based DSS, namely that the expertise is too difficult to transfer to more than a small group of 'experts', is no longer valid; it becomes possible to communicate the expertise more easily to the user, making 'experts' plentiful. The software, in such situations, remains only as a training tool for the new experts. In short, the determination of good abstractions is more important than the production of software; hence, knowledge engineering provides more than just methods for constructing knowledge-based systems.

Abstraction methods

Because of the importance of finding or constructing appropriate abstractions, we have been researching methods for systematically finding abstractions and evaluating their appropriateness. This work falls broadly into two categories, which correspond roughly to the two ends of the human/machine communication channel. From the user side it is necessary to have abstractions that are comprehensible to their human users. How can we acquire abstractions that correspond to a human's world view? From the machine side it is necessary to have abstractions that simplify the computational properties of the information to be processed. How can we evaluate the leverage that an abstraction offers? Here we will show one example from each of these categories.

Eliciting abstractions

How can we make sure that an abstraction corresponds to the concepts used by an expert or user in the course of decision making? The simplest way to find out is simply to ask. But such a direct approach has psychological problems: users and experts often are not explicitly aware of the abstractions they use to solve problems, and even if they are aware of them, they often report them inconsistently, incompletely, and incorrectly.

One method for helping humans to express abstractions is to recognize that many of the decision problems faced by humans have considerable commonalities, including consistent patterns of abstractions that support the decision process. If the expert expresses some abstractions that form part of such a pattern, then it is reasonable to insist that the expert give abstractions that complete the pattern. For example, the well-known method for solving problems known as 'divide and conquer' requires not only that one divide a problem and solve the resulting sub problems but also that one be able to combine the solutions to the sub problems together to form a solution to the overall problem. If an expert tells how to divide and conquer, it is reasonable to ask how to combine the results.

A number of problem types and corresponding patterns of abstractions have been identified; they are called *generic tasks*. For instance, diagnosis and fault classification problems are modeled using the generic task called '*hierarchical classification*'. A more detailed description of the application of task models to decision systems, with examples from telecommunications, can be found in [8, 9]. By modeling a problem-solving process with some generic task, a knowledge engineer suppresses some details of the process, concentrating only on certain aspects[1]. Modeling a problem-solving process as 'hierarchical classification' simplifies it by categorizing all abstractions as *hypotheses, refinements,* or *profiles*. Hypotheses correspond to fault categories and can be described at any of several levels of detail. For each hypothesis, a set of more specific faults can be defined that correspond to subsets of the fault category. Finally, for each hypothesis, some information about the profile (in terms of measurable values) of typical examples for the fault category must be given. Thus the expert's abstractions are organized as a tree of hypotheses, with the most general at the top and more specific hypotheses at the bottom *(Fig. 3)*. The refinement of a hypothesis is simply the set of its immediate children in the tree. The profile of a typical example of the category is included at each node in the tree.

Using the 'hierarchical classification' generic task, the knowledge engineer begins eliciting abstractions from the expert by asking for some fault categories (hypotheses); the consistency of the set of categories is enforced by arranging all the categories in the tree (refinements). Then, for each category, typical patterns of measurable values (profiles) are elicited. The result is a complete set of abstractions, along with their relations to one another. This process of elicitation of abstractions is the same whenever a problem-solving process is modeled by the generic task 'hierarchical classification'; hence, one can support this modeling process with a computer tool [1]. Such a tool makes it easy for the expert to maintain a set of hypotheses; if new information is acquired that changes the way the fault categories should be organized, then the tree of abstractions can be modified

[1] In this sense, a generic task is an abstraction; but since it simplifies the knowledge engineering process, it is an abstraction used by the knowledge engineer, not the expert or end user.

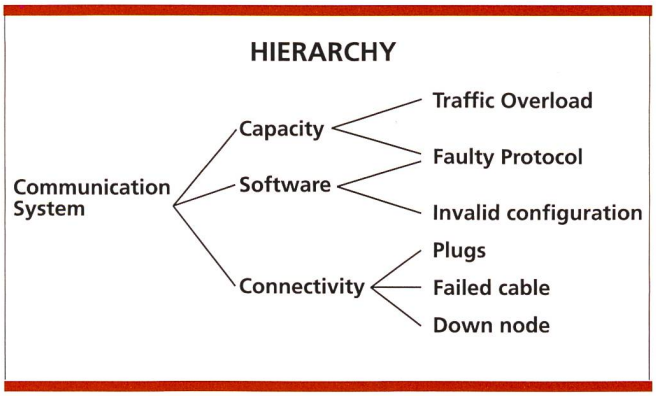


Fig. 3.
A partial classification hierarchy for communication network failures (e.g., Token Ring, Ethernet). In general, the hierarchy is not strict; in this case, certain software problems (Faulty Protocol, e.g. a noncooperative token-ring peer that never passes the token along) can appear as capacity problems. Only hypotheses and their refinements are shown here.

directly. But how can the domain abstractions collected by such a tool be used to support decision making?
Diagnostic Master II (DMII) [2] is a system that supports this acquisition process for 'hierarchical classification' problems. It provides graphical support for eliciting domain abstractions from the expert. It then uses these abstractions along with the generic task 'hierarchical classification' to automatically produce indices for a case base. Put into the context of the examples in section 'successful applications to telecommunications' DMII can be used to construct a system like ESP, but with the case base engineer replaced by the expert. This gives the expert freedom that is not offered by any of the three sample systems above, namely, the freedom to be able to define the abstractions that one wants to use to communicate with the machine.

Automatic construction of abstractions

In the previous section, 'Eliciting abstractions', we saw how it is possible to empower the expert (or end user) to define abstractions in such a way that the computer can use them. This solution only works for situations in which the generic task 'hierarchical classification' is appropriate. How can we, more generally, determine whether a particular abstraction is useful from a computational point of view? Presumably, an abstraction is useful if it makes a problem easier to solve.
In order to study how an abstraction might simplify a problem, it is first necessary to formalize the problem. In [4], Choueiry models resource allocation problems as *Constraint Satisfaction Problems* (CSPs). A CSP is made up of *variables, values,* and *constraints*. Each variable has a set of allowed values, and constraints place limits on the allowed values. For example, a common constraint says that two variables may not have the same value.
Once a problem has been formalized as a CSP, what are the possibilities for simplifying it? Choueiry offers a variety of such methods, but here we will describe only two: decomposition and interchangeability.
Decomposition refers to the identification of small sets of variables whose value assignments are independent of the rest of the problem. When such sets can be found, then the entire CSP can be decomposed into two (or more) smaller pieces which can be solved independently. Since the component set is independent of the other variables, the solutions to the subproblems can be simply combined to obtain a solution to the original problem. Problem decomposition can be used to apply a divide-and-conquer strategy to a resource allocation problem.
Interchangeability means that within one problem certain values can be replaced by others, without losing any possibilities for a solution. One can gather all the interchangeable values into a single abstraction and treat them as if they were a single value. This reduces the number of values that have to be examined in any subsequent search for a solution.
Of course, these two techniques (and others) can be used together – one can find interchangeable values for decomposed sets of variables. Abstractions that correspond to such sets are guaranteed to simplify the problem.

Example

An example of a simple resource allocation problem is shown in *Figure 4.* Suppose we have a team of seven service engineers, two of whom are senior engineers. Each engineer is capable of servicing the types of user equipment shown in the table. The chart below shows a plan for a seven-hour day, during which appointments have been made with nine customers whose complaints concern systems as shown. How do we assign the seven service engineers to the nine appointments, so that each appointment has an engineer who is trained to respond to the complaint?
This problem is NP-complete, which means, roughly speaking, that in general the best possible algorithm will require an exponential amount of time to find if a satisfactory solution exists [7]. In special cases, however, a human can sometimes solve these problems easily – if the problem can be broken down into pieces that the human can understand. Component sets and interchangeable values provide such pieces.
In the example we can decompose the problem into isolated components as shown in *Figure 5.* Recall that problem decomposition means that decisions made in one part of the problem do not affect decisions to be made in another. A short consideration of this example will show that the decisions to be made in subproblem A can be made independently of the rest of the

problem; these tasks all require senior engineers. Also, there are two tasks to be performed at once, hence they require two senior engineers. There are only two senior engineers available, so no assignment of a senior engineer outside this set can possibly result in a solution.

Subproblem A can be further decomposed, based on the fact that the tasks in the morning do not overlap with the tasks in the afternoon, and we can reassign our senior engineers. Any assignment made in the morning cannot affect assignments made in the afternoon. In this case, we say that subproblem A is decomposed into *problem components* C and D. We can now try to solve these (smaller) problems, without worrying that our decision might affect the assignment of junior engineers. The difficult problem at the top of Figure 5 has been decomposed into the three simple, isolated problems highlighted at the bottom.

When we try to solve subproblems C and D, we need to decide which senior engineer should be assigned to which customer. For problem D the required capabilities are ISDN and PABX. Since both senior engineers have both of these capabilities, the two senior engineers are interchangeable for subproblem D, and the problem is solved without any search at all. For problem C, since only one of the senior engineers has ATM competence, the two are not interchangeable; therefore, the two must be considered separately when finding a solution.

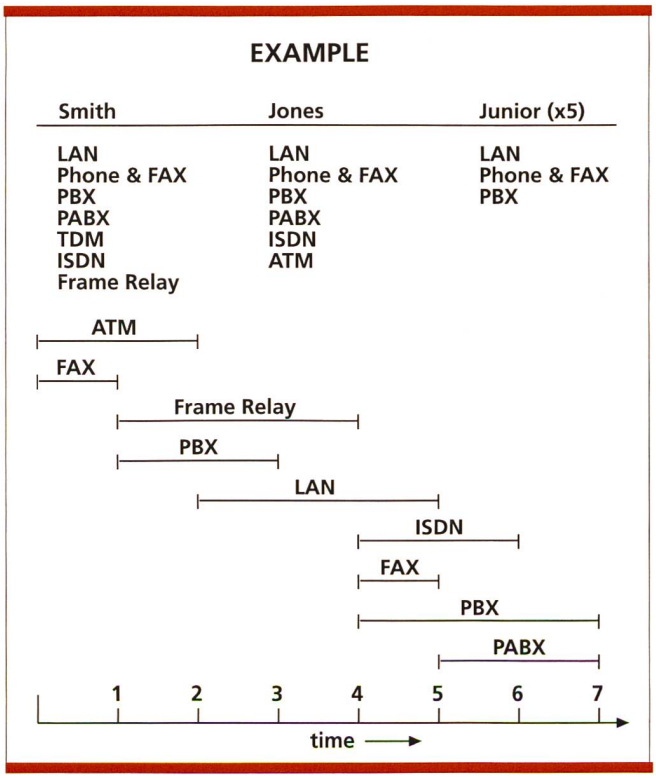


Fig. 4.
A small resource allocation problem. Two senior engineers (Smith and Jones) have a large set of service capabilities; one specializes in Frame Relay, while the other specializes in ATM. There are also five junior engineers (Junior) with more limited capabilities. Customer complaints have been planned for a seven-hour day as shown below. How can we assign engineers to these tasks?

Advantages

The resource allocation problem is NP-complete, which means that, in general, it cannot be solved efficiently by any known algorithm (see [7] for more details of the ramifications of NP-complete problems). Any particular problem, such as the one shown in the example above, might have a solution that is easy to find. In the example we have shown how decomposition and interchangeability can help a human problem solver find easy solutions. It makes sense, therefore, for a machine to help a human user by finding such abstractions (when they exist), rather than by trying to find a solution. This leaves the human user free to bring in any further constraints that might not have been modeled; in the example the human dispatcher might recall that Jones had installed the ISDN system in question. The interchangeability of Smith and Jones in subproblem D in Figure 5 allows the dispatcher to assign Jones to the ISDN problem, with perfect confidence that this will not cause problems elsewhere in the plan.

If no solution exists, algorithmic solutions (as are common in Operations Research) are even more problematic. After completing an exhaustive search of all the possible solutions, such an algorithm can determine that no solution exists. No further analysis is available – the reason for failure is not known. An interactive solution that decomposes the problem into isolated and interchangeable sets can help pinpoint the problem. For example, if we were to add another two-hour ISDN task in the afternoon in the example above, no solution would exist, because three senior engineers are required at the same time, while only two are available. The details of the possible assignments of junior engineers to other tasks are not important for understanding this limitation. Problem components and interchangeable values constitute abstractions in that they hide certain details of the problem that are not of interest to the solution. A problem component hides all other variables and values, while a set of interchangeable values hides currently unimportant distinctions between values.

Choueiry has developed a system that will automatically construct a hierarchy of abstractions (as shown in Fig-

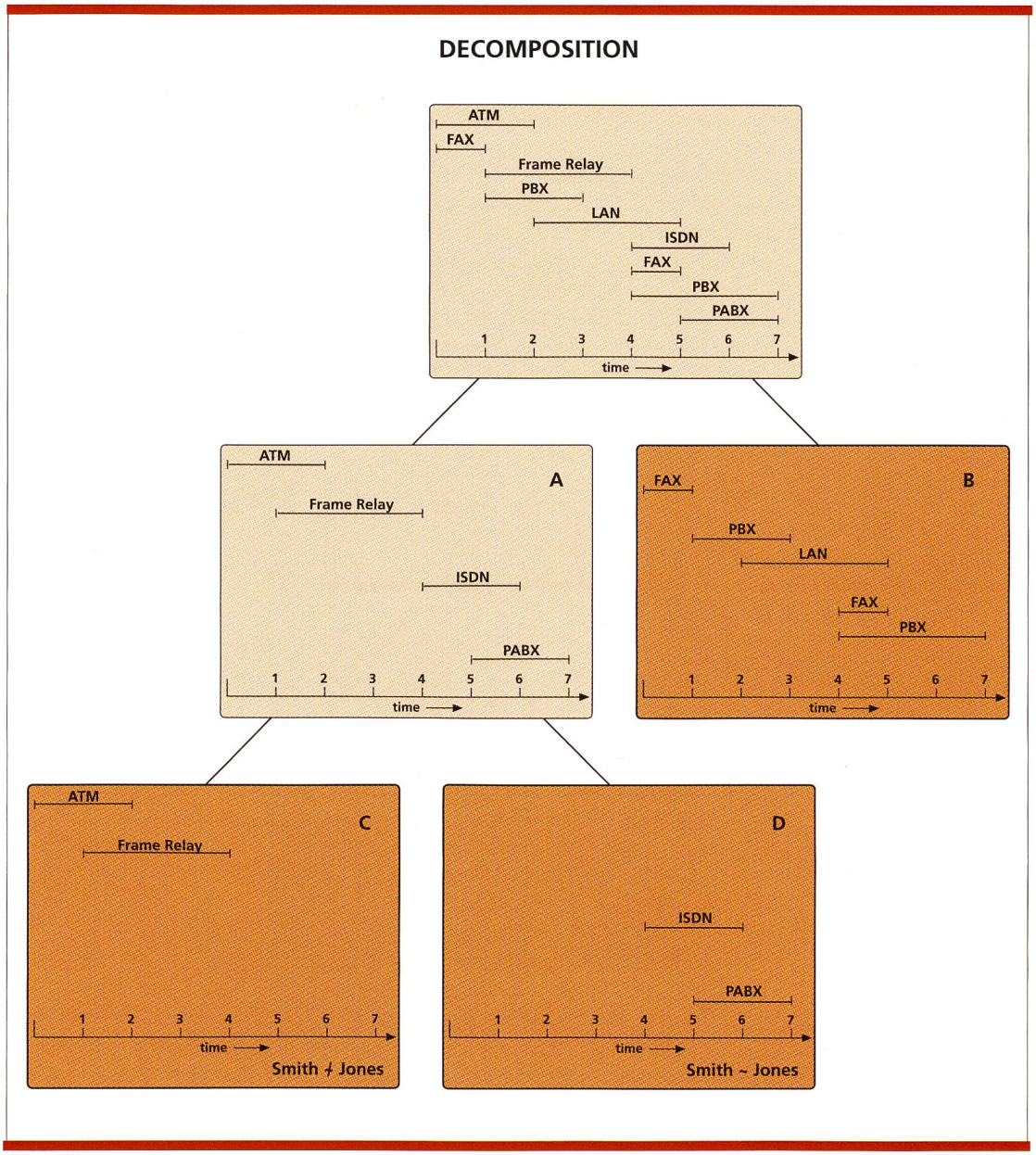


Fig. 5.
A decomposition of the problem from Figure 4 into isolated subproblems.

ure 5) using decomposition and interchangeability (among other techniques) to simplify a resource allocation problem. When such abstractions can be found, they make descriptions with them easier for a computer to process (since the number of possible solutions is smaller). Furthermore, the abstractions found by computing interchangeable values and problem components are typically comprehensible to users, since, for example, interchangeable variables have some features in common that make them interchangeable. Thus, Choueiry offers us a method for automatically finding abstractions that will facilitate communication from expert to machine to user.

Conclusions and horizons

With the current explosion of telecommunication services and technologies, telecom systems are becoming more and more difficult to design, manage and maintain. As the complexity of these systems grows beyond a level manageable by humans alone, decision support systems become essential. From the above examples it is clear that a major success factor in a DSS is the determination of abstrations that are appropriate to the problem-solving needs of its users. In order to make more powerful and flexible decision support systems, we need a sound, methodological ap-

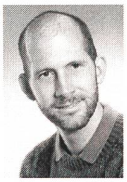
ZUSAMMENFASSUNG

Der Schlüssel zum erfolgreichen Einsatz von Softwaresystemen ist die Verständigung zwischen dem Programm und dessen Benützern. Bei Entscheidungssystemen muss das Programm fähig sein, mit dem Entscheidungsträger in einer angemessenen Form kommunizieren zu können. Normalerweise kommunizieren Menschen nicht in den schwerfälligen formalen Sprachen der Computer miteinander. Daher ist es wichtig, dass angemessene Abstraktionen gefunden werden, welche die Mensch-Maschine-Kommunikation wirkungsvoll unterstützen. Die wesentlichen Eigenschaften solcher Abstraktionen und die Entwicklung von Entscheidungssystemen unter Berücksichtigung dieser Eigenschaften werden in diesem Beitrag dargestellt. Der Einfluss der Abstraktion auf den Erfolg von Entscheidungssystemen wird anhand von drei Fallstudien im Bereich der Telekommunikation untersucht. Die verwendeten Abstraktionen reichen einerseits von allgemeinen zu spezifischen und anderseits von statischen zu dynamischen Abstraktionen. Abschliessend werden zwei Methoden zum Auffinden geeigneter Abstraktionen für die Entwicklung erfolgreicher Entscheidungssysteme dargestellt.

proach for representing, constructing and verifying abstractions.

In our work we are pursuing research on abstractions along two directions. First, since abstractions are already used by humans, we need a way to collect and organize these abstractions, so that they can be used in the construction of a DSS. For this purpose, we use the Generic Task methodology. Second, since the DSS is in fact a computer, the advantages of the constructed abstractions must be realizable computationally.

We bring these aspects of abstractions together by constructing cooperative systems, where part of the problem is solved by the computer and part by the human user. In this way, we can bring computationally intensive search methods (such as those studied under the name of Operations Research) together with knowledge-intensive methods under the direction of a human user, who is then in the position to take responsibility for the decisions made. The synergy of a combined human/machine decision support system improves the speed and cost effectiveness, and so the customer's expectations of quality service can be satisfied. This gives Swiss Telecom PTT a competitive edge in realizing the full commercial potential of telecommunication.



Dean Allemang completed his Ph.D. in Artificial Intelligence at the Ohio State University in 1990 on the application of device models to automatic debugging of computer programs. He joined the research and development directorate of the Swiss Telecom in 1994 as a research engineer, where he develops AI applications for the design, management and analysis of telecommunications networks. As a senior research assistant at the EPF in Lausanne, Dr. Allemang worked on applying AI methods to industrial problems. His current projects include technology transfer for telecommunication applications.

Beat Liver received the Diploma in Informatics from the Swiss Federal Institute of Technology (ETH) in Zurich in 1989 with a specialization in communication networks, distributed systems and computer-aided tools for VLSI. He joined the research and development directorate of the Swiss Telecom PTT in 1989. He is researching computer-aided tools for telecommunication network management problems in the broader sense of the term, in particular using results from operations research and artificial intelligence for developing innovative tools to diagnose, configure, and design networks.

References

[1] ISE Software AG. Diagnostic Master, Benutzerhandbuch. ISE Software AG, Tägerwilen, Switzerland, 1992.

[2] Dean Allemang. Combining case-based reasoning and task-specific architectures. IEEE Expert, pages 24–34, 1994.

[3] Mike Carr, Chris Costello, Kare McDonald, Debbie Cherubino. Embedded AI for sales-service negotiation. In Proceedings of the Sixth Innovative Applications of Artificial Intelligence Conference, pages 25–34, 1994.

[4] Berthe Yazid Choueiry. Abstration Methods for Resource Allocation. Ph.D. thesis, Ecole polytechnique fédérale, Lausanne, 1994.

[5] Carol Hislop and David Pracht. Integrated problem resolution for business communications. In Proceedings of the Sixth Innovative Applications of Artificial Intelligence Conference, pages 63–74, 1994.

[6] Hieu Le, Gary Vrooman, Philip Klahr, David Coles, Michael Stoler. Expert investigation and recovery of telecommunication charges. In Proceedings of the Sixth Innovative Applications of Artificial Intelligence Conference, pages 75–82, 1994.

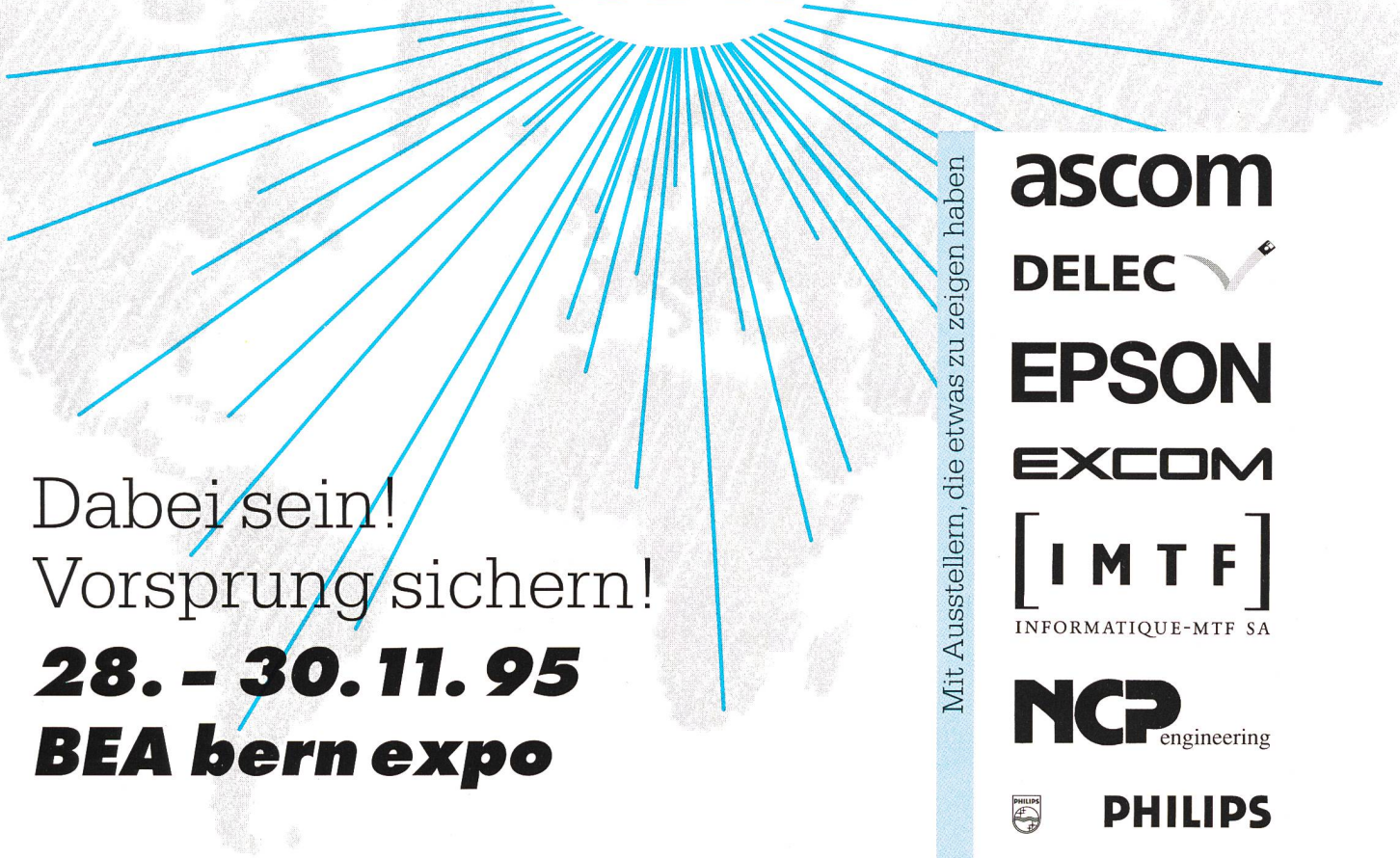
[7] Beat Liver. Rechnergestützte Konstruktion: Nutzen und Problemlösungsmethoden. Technische Mitteilungen PTT (11): 523–528, 1993.

[8] Beat Liver, André Prim. Wissensbasierte Systeme im Netzwerkmanagement. Teil 2: Erfahrungsbericht. Technische Mitteilungen PTT (2): 38–44, 1992.

[9] André Prim, Beat Liver. Systèmes experts pour la gestion de réseau. 2e partie: compte rendu des resultats. Technische Mitteilungen PTT (1): 12–17, 1992.

GLOBAL 95

WORLDWIDE ISDN SOLUTIONS



Dabei sein!
Vorsprung sichern!

28. – 30.11.95
BEA bern expo

Multimediale Datenkommunikation wird Wirklichkeit.

Mehr noch: Mit ISDN wird Information überall und jederzeit verfügbar. Information bedeutet Vorsprung.

Und Vorsprung sichert Ihre Existenz.

An der GLOBAL '95 treffen Sie sich mit allen, die in ISDN die Technologie der Zukunft erkennen.

Die GLOBAL'95 verbindet für drei Tage mehr als 30 Länder zu einem weltumspannenden Netz.

Fachleute diskutieren aktuelle Themen. Anbieter zeigen Projekte und Lösungen.

Mit Ausstellern, die etwas zu zeigen haben

ascom
DELEC
EPSON
EXCOM
IMTF
INFORMATIQUE-MTF SA
NCP engineering
PHILIPS
TELECOM PTT
TELELINK
DIE SCHWEIZER MODEMMACHER
TERCOM
TECHNICAL ENGINEERS FOR RELIABLE COMMUNICATION

Veranstalter:
TELECOM PTT

Organisation:
Icon AG
Kaspar Pfeiffer-Strasse 21
Postfach 508
4142 Münchenstein 1
Telefon 061-413 03 81
Fax 061-413 03 83

HOLSTEIN DESIGN BASEL