An environment to support problem solving

Bots, P.W.G.; Sol, H.G.

Published: 01/01/1987

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 18. Nov. 2018
AN ENVIRONMENT TO SUPPORT PROBLEM SOLVING

by

P.W.G. Bots
and
H.G. Sol

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN, The Netherlands

September 1987
AN ENVIRONMENT TO SUPPORT PROBLEM SOLVING

P.W.G. Bots and H.G. Sol
Delft University of Technology
The Netherlands

An environment to support problem solving provides means for the recognition, specification and analysis of problem situations, and integrates these with automated tools for the realization of systems that will aid in finding and implementing solutions. As such, it addresses both the methodological and the technological aspect of problem solving. An important feature of a problem solving support environment is that the results of both problem analysis (formal descriptions) and solution finding (working system components) can be stored and documented effectively, as they represent knowledge of a specific problem area that can later be applied in related situations.

Problem solving and organizations

The process of solving a complex problem involves the execution of a large number of specific tasks. These tasks will be performed by people, either individuals or groups. Each task can be viewed as a decision process which affects and is affected by some part of the organization. Organization, task and decision are key concepts that will need clarification, as they are often used in subtly different meanings.

If we define a system as a part of the world we want to take into consideration, organizations are dynamic systems. Any dynamic system can be abstracted to an information system (IS) and a real system (RS), where the former controls the latter. Control is realized through communication: the RS sends messages concerning its state to the IS, which interprets these messages. The IS then acts according to the obtained information by sending messages that will affect the RS, i.e., cause its state to change.
In order to constitute a usable paradigm for describing organizations, two things must be added to this perspective: a recursion principle and the notion of an environment*. The recursion principle guides the demarcation of a system, and consists of the idea that IS and RS themselves are dynamic systems, and therefore IS-RS combinations. Similarly, the original dynamic system can be viewed as a component of some larger IS-RS combination. The recursion principle can be applied repeatedly until an appropriate level of detail (or abstraction) has been reached. At any level of abstraction an IS-RS combination has an environment which comprises everything its IS component does not control. The environment may send messages to the IS, thus possibly affecting the way in which the IS controls the RS. It may, however, not send messages directly to the RS, for this would imply that the IS does not really control the RS.

In general, each task that has to be performed constitutes an IS-RS combination. Thus, if an organization becomes larger (that is, its function becomes more comprehensive, involving more tasks), more IS-RS combinations can be identified. Application of the recursion principle to either component corresponds to dividing a task into subtasks, a process that can be repeated until an RS component can no longer be considered to be a dynamic system:

The purchase department's problem

As an illustration, we shall apply this paradigm to an imaginary company that manufactures certain chemical products (an example we shall use throughout the rest of this article). One department of the company's office is responsible for the acquisition of raw materials. Each quarterly period these raw materials can be purchased directly from suppliers at the ongoing price. Prices may change every period. This fluctuation can be circumvented by contracting a supplier for a large quantity at the ongoing price. For four consecutive periods, one fourth of this quantity will then be delivered as if it had been ordered directly for that period, except that the price has been fixed to that of the first delivery. At any given moment no more than one contract may exist for each type of raw material. Any surplus of raw materials can be

* Note that this environment is not the problem solving support environment mentioned in the first paragraph.
stored, but warehouses must be rented. The problem the purchase department faces each period is to determine for each type of raw material the most economical quantities to order directly and to purchase on contract.

The first step in the analysis of this problem is the description of the organization in terms of IS-RS combinations. If we view the purchase department as an IS, the RS it controls consists of existing contracts and quantities of raw materials (both ordered and in store). All other parts of the company, the suppliers and the warehouses are considered to belong to the environment of this IS-RS combination.

The messages the RS can send include the specifications of existing contracts, the price of each ordered quantity and the total amount of raw materials in store. Relevant messages from the environment include the estimated need for each type of raw material (messages sent by the company's production department), names and addresses of suppliers, and the minimum amount of raw material that they will deliver on contract (messages sent by the suppliers), and the rental costs of the warehouses (messages sent by the owners of these warehouses). The messages sent by the IS are either direct purchase orders or forms to initiate new contracts.

To achieve one higher level of detail, the recursion principle can be applied once by viewing the acquisition of each type of raw material as a separate task constituting a smaller IS-RS combination. The RS component of each such combination consists of the order and contract initiation forms for one specific type of raw material. The corresponding IS determines whether and for which quantities these forms will be sent out during a given period.

Distributed systems and coordination

The basic assumption we make is that task improvement can be achieved through the use of information technology (cf. Sprague, 1986). A plausible assumption, as in most organizations nowadays different information technologies, such as Electronic Data Processing systems, Management Information Systems, Decision Support Systems, Expert Systems, Word Processing
systems and Electronic Mailing Systems have become indispensable for the adequate performance of tasks. Ideally, these systems are perfectly integrated. In reality, this is hardly the case. The developments in computing technology have made computer systems smaller and cheaper, but at the same time more oriented towards individuals (the term Personal Computer illustrates this). Developments in communication technology counterpoint this individualization, and would seem to provide the necessary means for integration. However, the actual realization of task improvement is a design problem, rather than a technological one.

The tendency to use more distributed systems (with the extreme of each staff member having a personal computer system) strongly affects the organizational structure in terms of IS-RS combinations. Tasks will eventually be performed mainly by manipulating electronically stored information. Even at the highest level of abstraction, the RS component of a task will be some subset of all accessible pieces of information. For many tasks these subsets will not be disjoint. As most tasks are performed in parallel - a situation which both stimulates and is stimulated by the distributed nature of the computer system - there will be an overlap between their RS components. This means that some RS receives control messages from more than one IS. Unless proper coordination between these IS components exists, conflict is inevitable.

We share the conviction that task improvement can be achieved through the use of information technology, though only when a new methodology is used. This methodology should address the following problems:

- The coordination problem that arises when the RS components of different tasks overlap (cf. Bosman and Sol, 1985, Bosman, 1986). Coordination of tasks in the first place requires knowledge on the nature of each task. A first step in obtaining this knowledge is proper demarcation of the IS and RS components. Analysis of these components will indicate where coordination is needed, not so much how it can be achieved. In most cases, the choice of an appropriate coordination mechanism will remain.
The problem of adaptiveness, caused by the phenomenon that tasks will change, due to a changing environment. This implies that respecification and a new analysis is required whenever a (major) change occurs. In addition to changes in information needs, this analysis may reveal different needs for coordination.

Dealing with both the coordination problem and the problem of adaptiveness is the essence of designing computer based information systems for an organization.

In the remainder of this paper we will describe an approach and supporting tools for task analysis that explicitly address the coordination problem. The aim is to obtain detailed, recognizable descriptions of tasks, and of the way these tasks - true to the recursion principle - determine the organizational structure by their interrelationships. We shall indicate how the results of task analysis can also provide considerable support in solving the problem of adaptiveness.

Problem solving and task analysis

Solving a complex problem involves the performance of a large number of tasks. The focal concept in our method for analysis is the decision. We view performing a task as the identification, making and implementation of a set of decisions (cf. Huber and McDaniel, 1986). As it incorporates the identification aspect of decisions, this perspective leaves room for the often stated belief that problem solving is a creative activity in the sense that different individuals will solve a specific problem in different ways. This translates in our terminology as "each individual identifies his or her own set of decisions".

Identification of a decision is an analytic activity that yields a description of the circumstances under which the decision will have to be made, the part of the RS and the environment that may influence the making of the decision, and the part of the RS that may be affected by the implementation of the decision. Ideally, this description will be sufficiently detailed to provide adequate support for the decisionmaker.
Making a decision is an activity that involves taking stock of the actual situation, formulating a number of alternative courses of action, pondering their possible effects on the organization, and choosing one alternative. If this activity is performed according to some algorithm (that possibly could be executed by a machine), we speak of an autonomous decision.

Implementation of a decision is the activity of translating the chosen alternative into information that will cause the RS to change, and communicating this information (sending the appropriate messages to the RS).

Tasks that involve a confusingly large number of decisions should be divided into subtasks until each subtask can be managed. This method of reducing complexity is a common design practice. However, if the total number of tasks is large, the coordination problem arises. We believe that this problem can be contained by adding some specific coordination decisions to each task that is divided into subtasks. To explain this idea, we need a more formal definition of the notions decision and task.

Some formal definitions

We define a decision $D$ as a 3-tuple $(I,A,S)$. Here, $I$ stands for the part of the RS and the environment that influences $D$, that is, affects the making of the decision. $A$ stands for the part of the RS that is affected when $D$ is implemented. $S$ denotes the support of $D$, which is the collection of means that facilitate the making and implementation of $D$.

We define a task $T$ as an ordered pair $(D,S)$ where $D$ is a set of decisions and $S$ is a set of tasks called the subtasks of $T$. If $S \neq \emptyset$, every $d \in D$ is called an internal decision. If $D = \emptyset$, the subtasks $s \in S$ are called independent tasks. Each task constitutes an IS-RS combination, where implementation of the decisions of subtasks affects the RS. Coordination of these subtasks is realized through internal decisions like "How should subtasks be scheduled?" and "What kind of support and information should actually be used when making decision $d$ in subtask $s$?". In other words, implementation of internal decisions affects the IS.
The collection of all decisions that can be identified in the process of solving a problem is called the *decision structure* of that problem. One should think of a decision structure as a directed graph, in which each node $d_i$ is a decision. There is an edge leading from $d_i$ to $d_j$ if and only if the influencing part $I$ of $d_i$ overlaps with the affected part $A$ of $d_j$. The advantage of this representation is that difficult situations such as *strongly connected components* (parts of the decision structure in which the outcome of each decision depends on the outcome of all other decisions) and *cycles* (cyclic dependencies between decisions) can be detected algorithmically. The existence of such situations has a great impact on how decisions can be allocated to tasks.

The collection of all tasks involved in the solving of some problem is called the *task structure* of that problem. We stress that a task structure is not something predefined which can be "discovered". Analysis of a decision structure may reveal the need for coordination of certain decisions, but the choice of the appropriate coordination mechanism is a design issue. Each choice will show up in the task structure as a number of internal decisions. A task structure should be viewed as an essential *variable* in the design of information systems, with the purpose of conveying an impression of both what the problem is and how it is going to be solved.

**The purchase department’s problem revisited**

Consider again the problem faced by the purchase department of our imaginary company. Determining how the IS controls the RS by identifying decisions is the second step in the problem analysis. The messages sent by the IS are suitable starting points, as they correspond to the implementation of decisions that are made within the IS. For each type of raw material (let us assume there are only three types, called A, B and C), two messages can be sent: a direct purchase order and a form to initiate a new contract. The first message corresponds to the decision "How much should we purchase on contract?", the second to the decision "How much should we order directly?". Further identification of these two decisions involves determining their influencing and affected parts, and the required support (see figure 1).
The decision structure of the purchase department's problem (see figure 2) can be determined by searching for overlapping influencing and affected parts. Such overlap can be found between $c_X$ and $d_X$ - hence the straight edges - and within $c_X$ itself, causing the cycles with the $c_X$ decisions as sole node. These cycles indicate the need for some coordination mechanism. In this
case the mechanism is simple: if a time series is used as data object for storing contract specifications, the decisions can be made on information from the previous time period, so no internal decisions need to be added.

![Decision structure of the purchase department's problem.](image)

As there are no cross dependencies (edges between two decisions concerning different types of raw material), the acquisition of each raw material \( X \) is an independent task \( T_X \). A possible task structure would therefore be:

\[
(\{\}, \{T_A, T_B, T_C\}) \quad \text{with} \quad T_A = \{(c_A, d_A), (\{\})\}, \\
T_B = \{(c_B, d_B), (\{\})\} \\
\text{and} \quad T_C = \{(c_C, d_C), (\{\})\}
\]

To illustrate the use of internal decisions when solving more complex coordination problems, we can modify the purchase department's problem. Suppose that raw material \( A \) can substitute raw material \( B \), and vice versa. This would add a number of cross dependencies, as shown in figure 3.
Figure 3. Decision structure of the modified purchase department's problem.

Task $T_A$ and $T_B$ now are no longer independent, and therefore constitute a coordination problem. To overcome this problem, two internal decisions should be added: $g_{AB}$ ("Which type of raw material is cheapest over four periods?") and $h_{AB}$ ("Which type of raw material is cheapest in the next period?"). The outcome of $g_{AB}$ will determine whether decision $c_A$ or $c_B$ should be made. Likewise, the outcome of internal decision $h_{AB}$ will make either $d_A$ or $d_B$ obsolete. The resulting decision structure is shown in figure 4. An appropriate task structure would now be:

$$((\emptyset),(T_{AB},T_C)) \quad \text{with} \quad T_{AB} = ((g_{AB},h_{AB}),(T_1,T_2))$$

$$T_C = ((c_A,d_C),(\emptyset))$$

$$T_1 = ((c_A,c_B),(\emptyset))$$

and

$$T_2 = ((d_A,d_B),(\emptyset))$$
Problem analysis is an iterative process

As demonstrated in the previous section, problem analysis is performed in accordance with the recursion principle. First comes the demarcation of the IS and RS components of the problem, and the identification of the messages these components can send. The messages sent by the IS correspond to the implementation of decisions that are made within that IS. Identification of these decisions yields an initial decision structure of the problem that is analyzed. The level of detail of the description of the influencing and affected parts of the decisions will correspond to the level of detail of the IS-RS combination. An initial task structure can be obtained by demarcating more detailed IS-RS combinations (that is, applying the recursion principle once), and determining which of the decisions identified so far are most strongly related to which IS-RS combination. Subsequently, each of these new IS-RS combinations can be analyzed in a similar fashion.
So far, the nature of this approach has been top-down. This changes once decision structure and task structure have been determined at a satisfying level of detail (in theory, this could be any level of detail). Thorough examination of the decision structure, using the algorithmic methods mentioned earlier in this paper, may reveal the need for changes in the task structure. Such changes could be the introduction of some new coordinating mechanisms, or the reallocation of decisions among tasks to reduce the number of coordinating decisions. In either case the decision structure is changed as well. Analysis is therefore an iterative process. The final output of this process is a detailed description of a problem, the decisions that have to be made to solve this problem, the information required to properly make these decisions, and the means by which this information can be obtained.

Iteration is but one temporal aspect of problem analysis, inherent to the paradigm used to describe organizations. The coordination problem introduces a second time aspect: most coordination mechanisms will need an explicit modeling of the temporal dimension. Internal decisions concerning the scheduling of decisions or the selection of appropriate data from time series are typical examples. In order to cope with the problem of adaptiveness, a third time aspect will have to be considered: the evolution of problem descriptions. Task structures and decision structures can be stored as complex data structures, but they do change over time. Since there will be recurring problems, it is highly desirable to maintain a library of these structures or parts thereof. This obviously asks for some kind of "version management" of problem descriptions.

**Problem analysis generates knowledge**

Decision structures and task structures represent knowledge of specific problem areas. This knowledge can be applied in various ways. Its immediate use will be to help solve the problem (the analysis of which has led to this knowledge) by providing a clear understanding of the problem structure. It also specifies where information technology can offer adequate support, and as such provides guidelines for the design of a "cut to fit" information system. A third application results from the possibility to store task structures and decision structures in a library. As new
types of problems arise and are analyzed, new descriptions of tasks and decisions, but also new coordination mechanisms and supporting information technologies, are added. This will cause the library to evolve into a veritable knowledge base that will facilitate both the analysis of new problems and the design of appropriate information systems.

One step further along this line lies the idea that problem analysis will simultaneously yield an operational computer based system, where tasks are mapped onto workstations, and decisions correspond to windows or screens. Obviously, highly sophisticated hardware and software is required to realize this idea. The architecture of such a system, which we shall refer to as a problem solving support environment, is strongly object oriented and adheres to the given definitions of decision and task. Its basic elements are task objects, decision objects, data objects and support objects.

A task object corresponding to a task \( T = (D,S) \) consists of a document which verbally describes the task objective and outlines how this objective should be met, a list of decision objects corresponding to the set of internal decisions \( D \) that represent the necessary coordination mechanisms, and a list of task objects that corresponds to the set of subtasks \( S \).

Decision objects are closely related to the identification aspect of decisions. Each decision object corresponding to some decision \( d = (I,A,S) \) consists of a document containing a verbal description of the decision, a list of the data objects that model its influencing part \( I \), a list of data objects that model its affected part \( A \), and a list of support objects \( S \) that incorporate the information technology that will provide the required support in decision making.

Data objects model the components of some RS. As these components will widely differ in nature, there will be many different types of data objects, including a variety of documents, time series and matrices. This causes the need for a classification which guarantees that all data objects
will have properties that make algorithmic analysis techniques possible. An object class inheritance mechanism common to most object oriented programming languages naturally provides such classification.

Support objects constitute some form of information technology. As such, they may originate from many scientific disciplines. Typical examples are support objects that perform specific statistical analyses or optimization by means of for example linear programming, provide fast access to a database, facilitate document processing or electronic mail, or make possible spreadsheet-like manipulation or graphical representation of data objects.

Data objects and support objects will be the actual components of the "cut to fit" information system that results from a problem analysis. In the process of problem analysis, dependencies between decisions become visible as shared data objects (the influencing and affected part being modelled with such objects). The subsequent process of system design is limited to specifying which support objects are needed, and which data objects these support objects use. The resulting system is a representation of knowledge which can guide as well as support the actual performance of tasks.

Conclusion

We have outlined a methodology for the design of information systems, based on the representation of problem situations in an organization as decision structures and task structures. This methodology requires a sophisticated environment which integrates a multitude of different information technologies. To answer the question whether such an environment can be realized, a further refinement of the proposed methodology and a considerable amount of empirical research is needed.
References


