Balance between Abstract Principles and Concrete Instances in Knowledge Communication

Toshiya Akasaka and Yusaku Okada

Keio University, Faculty of Science and Engineering, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan
{to48_a,okada}@ae.keio.ac.jp

Abstract. In tasks requiring dealing with variable situations, workers are expected to do more than following prescribed instructions. In this paper, we presented our view and framework for creating instructions with a good balance between the flexibility of abstract principles and the preciseness of concrete instances, which aims at helping instruction receivers become capable of dealing with variable situations where no concrete instructions are available. Our approach represents knowledge using an abstraction hierarchy. It is situated in our grand model which deals with the whole picture of knowledge communication. A case study is also presented, which suggests that seen in our view existing manuals can be improved by providing them with principles combined with instance-dependent variables.

Keywords: Hierarchic Representation of Knowledge, Modeling Language, Knowledge Management, Knowledge Visualization.

1 Introduction

Teaching how to perform a task is always difficult, especially when the task process is complicated. Usually, instructions are needed to communicate required knowledge. Sometimes, instructions take the form of written manuals and are provided for such tasks as operation of man-made systems like plant operation, operational business activities like call center, management systems implementation like project management, and so forth. Manuals of these tasks are used not only to help new comers know how to perform the task, but also to share common knowledge among people involved in the task.

Manuals are written with the aim of giving explicit instructions free of ambiguity. For example, some manuals about plant operation describe the sequences of required operations in a well-structured form, defining contexts in a concrete manner and specifying exact buttons and levers to operate for each context as well as the order of those operations. Aiming to avoid ambiguity, such manuals sometimes end up being the compilation of case-by-case operation sequences, still falling short of covering all possibilities. In other words, manuals fail to convey principles and full range of possibilities as a result of its aim to take a rigorous, ambiguity-free form.

Receivers of instructions taking such a rigorous form often have difficulty with having well-organized understandings of the task; they simply follow prescribed instructions and do not think of any principle from which such concrete instructions

D. Harris (Ed.): Engin. Psychol. and Cog. Ergonomics, HCII 2011, LNAI 6781, pp. 285–293, 2011.

are derived. This means that they cannot take appropriate actions without instructions. Unfortunately, there are many cases for which sufficient instructions are not available, such as troubleshooting in software applications, incident handling in plant operation, demanding-customer handling in call center, and so forth. Facing novel, unfamiliar situations, they are at a loss what to do, even if the situations can be resolved by making use of the actions they are used to. They know fragmentary pieces of knowledge, but do not know what of them can be applied to the actual situation in what way.

In work settings, it is important for workers to be capable of dealing with such unfamiliar situations. If they otherwise could not deal with situations, appropriate performance would be hard to guarantee. Also, if they always consult well-experienced people every time they fail to find explicit instructions, such experienced workers may find it difficult to concentrate on their own work. This is all the more problematic if veteran workers are decreasing in number. Of course, it is almost impossible to make novices be experts simply by providing instructions, however good they may be. Still, giving them a bid of flexibility makes a big difference if readers otherwise could do no more than following explicit instructions. To succeed in this, there needs to be a good medium of knowledge, giving readers the flexibility to take appropriate actions for variable situations.

However, giving such flexible instructions is not an easy task. The challenge here is how to balance the flexibility of abstract principles and the preciseness of concrete instances. Obviously, describing all alternatives in minute detail is impossible, since such detail instructions are infinite in number as is the case with troubleshooting of software errors. On the other hand, if one tries to describe instructions in a larger granularity, the number of alternatives may become of a manageable magnitude, but there is a risk of causing instruction receivers to be at a loss what to do due to the loss of concrete instructions.

The goal of our research is to investigate a systematic way to provide instructions in the way that gives principled explanations in an abstract description while covering an appropriate variety in a concrete description. In this paper, we present our research approach toward tackling this challege as well as the current state of progress in discussion and investigation.

2 Research Approach

2.1 Abstraction Hierarchy

Our approach involves representing knowledge using an abstraction hierarchy. If some instances are derived from a certain model, rule, law, or any other form of general principle, that principle is said to be abstract as compared to those instances, while the instances are said to be concrete as compared to the principle. Likewise, if some means are to achieve a certain goal, the goal is abstract as compared to those means, and the means are concrete as compared to the goal. In short, something abstract is the reason why something concrete appears or is used. Consequently, we can find more concrete instances than abstract ones, for from one abstract principle or end come several (sometimes infinite) instances or alternative means. The abstraction

hierarchy considered here has several layers each representing a certain degree of abstraction, and elements that are considered to be abstract to the same degree are situated on the same layer. Each layer is abstract to the lower one and concrete to the upper one. Thus, we can find more instances as we go down along the hierarchy. The illustration of our abstraction figure is shown in Figure 1. Note that this is just a rough sketch of our idea, just intended to help readers better understand it. We greatly owe this abstraction hierarchy to Rasmussen [1]. Rasmussen, who had long studied human-machine interface in major physical systems such as plant operation systems, discovered that operators of such systems seemed to have several information processing modes; they first processed a set of numerical data on the information display, and the next moment they thought of the system structure including mass flow model, a totally different type of thinking from recognizing numerical data. He then went on to propose a structured representation of a physical system as it appeared to operators, with the aim of developing a framework to describe human-machine interaction. Thus, he presented an abstraction hierarchy with each layer representing the structure of a physical system from the viewpoint of a certain abstraction level. Although limited to physical systems structure, Rasmussen's idea of representing human's information processing as a travel along the abstraction hierarchy is a generally applicable framework for studying knowledge and its explanations.

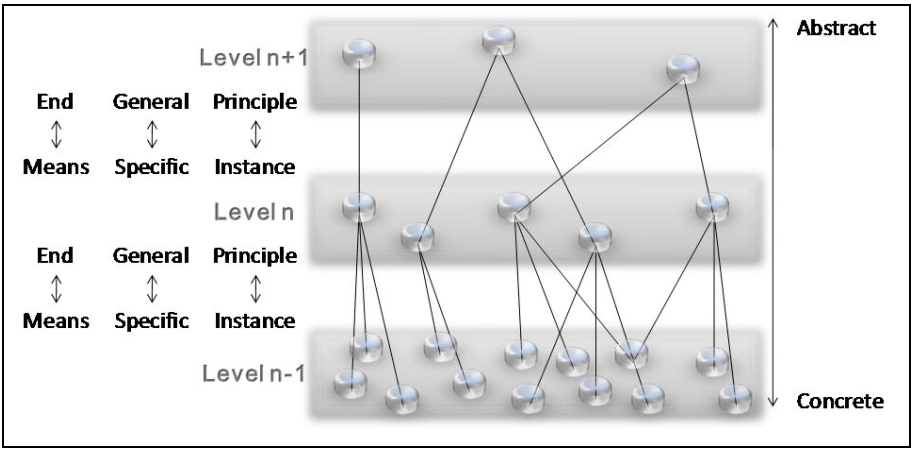

Fig. 1. Illustrative sketch of abstraction hierarchy. As going up along the hierarchy, one needs fewer descriptions to explain the same range of possibilities (illustrated by the number of circles), while description themselves become less self-explanatory, demanding supplementary explanations.

Using the abstraction hierarchy, we can represent our problem differently. First, we can say that if somehow organized, instructions are consistent in terms of abstraction level. Most instructions are given in a specific abstraction level. Take manuals of software applications for example. Some manuals give instructions by sequences of GUI (Graphical User Interface) level operations (e.g. "Select the file

item on the menu bar"), while others, perhaps those for professional software engineers, may use sequences of more granular operations like "Save the document in .txt format", "Install the driver software with the bundled CD-ROM." It is fair to say that the instructions of the second type are unlikely to coexist with those of the first type and vice versa, probably due to the difference in intended readers. The abstraction level differs across manuals, but mostly remains consistent within one.

Given that a set of instructions are provided in a consistent level of abstraction that is low enough for intended readers to comprehend the instructions (let us call this level the *ground level*), then the next step is how to help readers become capable of dealing with unexpected situations. If one tries to explain all possible solutions to every possible situation in the ground level, it may be infeasible due to an unacceptably large number of possibilities; the writer may not be able to recognize all possibilities and find sufficient pages to write them down on. A possible solution here is to raise the level of abstraction. Remember that from one abstract principle or end come several (sometimes infinite) instances or alternative means. Translating this, we can have that with a higher level of abstraction you can explain a broad range of possibilities with fewer descriptions. The problem here is how to balance between the flexibility of abstract principles and the preciseness of concrete instances. Writers are now demanded to explain a principle while showing a variety of possibilities derived from that principle, both in a comprehensible manner. On top of that, the description has to be smarter than simple enumeration of all possibilities at the ground level of abstraction; it is meaningless to make use of higher-level principles unless the resulting description becomes easier to comprehend than the original one. Thus, our core problem can be formulated as follows; to investigate how to balance between the flexibility of abstract principles and the preciseness of concrete instances in a comprehensible manner, with the aim of helping instruction receivers become more capable of handling situations with no explicit instructions prescribed.

In our view, many of existing written manuals tend to describe only one or a few typical instances at the ground level of abstraction, and that tendency is even clearer for the manuals of large systems, like plant operation systems. Academically- and industrially-proposed modeling languages have also failed to show a promise. For example, there are many modeling languages intended for software systems development, from traditional ones like flowchart and dataflow diagram to such modern techniques like UML (Unified Modeling Language). They are fairly good at describing well-defined structures, but lack the flexibility to describe the variety seen in real-world tasks, demanding the modelers to make every instruction explicit. In fact, some professional developers already gave up using a modeling method as means of eliciting requirements, and instead have tried to ensure effective communication by rapid trial-and-error; they repeat a small cycle of producing a trial product and demonstrating it to the clients. After all, the modeling languages for software development need to be compatible with computer-oriented digital processes, and it is only natural that they cannot tolerate ill-defined possibilities in the real world. On the other hand, there is a modeling language which intends to describe human-involved processes. The functional modeling language IDEF0 [2], which was based on the SA (Structured Analysis) language [3], a graphical modeling language,

has as its premise the belief that describing and managing human activities is important in order to make good use of information technology systems. Like our concept, the SA language features the hierarchic structure of a subject. Although its rigorous grammar does not immediately lead to flexible expressions, the SA language provides users with unique notations which, together with the hierarchic structure, successfully represent reasons and mechanisms. Careful study and some tailor of the SA language may bring us a promising method to express both abstract principles and concrete instances.

2.2 Grand Model

Although our current problem has been already delineated, we now briefly sketch our grand model to situate our current investigation in the whole picture. We have been discussing how an instruction should be given to its receiver. Put in a broader perspective, those who give instructions also used to be novices in the past. They learned a lot by first-hand experience and instructions from yet other people. That is why they are now capable of handling variable situations; in their mind are lots of instances and abstract principles. To give instructions is to decode such abstract principles and transform them into visible description with the help of memories regarding past instances. Then, the resulting instructions convey knowledge to instruction receivers. However, such an easy story is hardly the case. Unfortunately, the fact is that experience is the hardest thing to communicate and the gap between well- and poor-experienced people has been never seen bridged.

One possible reason is that some parts of experience are filtered out, not taken into account when abstraction occurs. Having experienced many instances and learning lots of pieces of knowledge, people for some reason try to generalize them and find some abstract principles. Although it is impossible to describe the process of abstraction, many would agree that the process is not as simple as a strict one-way sequence. Instead, it should be a kind of parallel process; from a set of instances comes an incomplete principle which has not yet taken a complete form, while the instances are sought that fit the fetus principle well. This means that there can be some selections of instances where those which do not fit the incomplete principle are neglected. Sometimes, it is useful to allow different principles to coexist so as to absorb as many instances as possible, but that is not so easy. When instances and potential unknown principles are neglected in order to allow only one principle, the variety of concrete instances are likely to be neglected when instructions are given.

Another possible reason is that the expressive power of a language prevents principles and instances from being described properly. The type of language affects people's thinking. When writing computer program code, no one would think how to write an analogy of a certain algorithm, because there is no way to describe an analogy using programming languages. On the other hand, usual people do not think of class or instance when writing a natural language. This is also because natural languages are not good at describing such concepts. In other words, devising a good language can bring us a solution to communication problems. That is why many languages have been studied and some of them have been used widely.

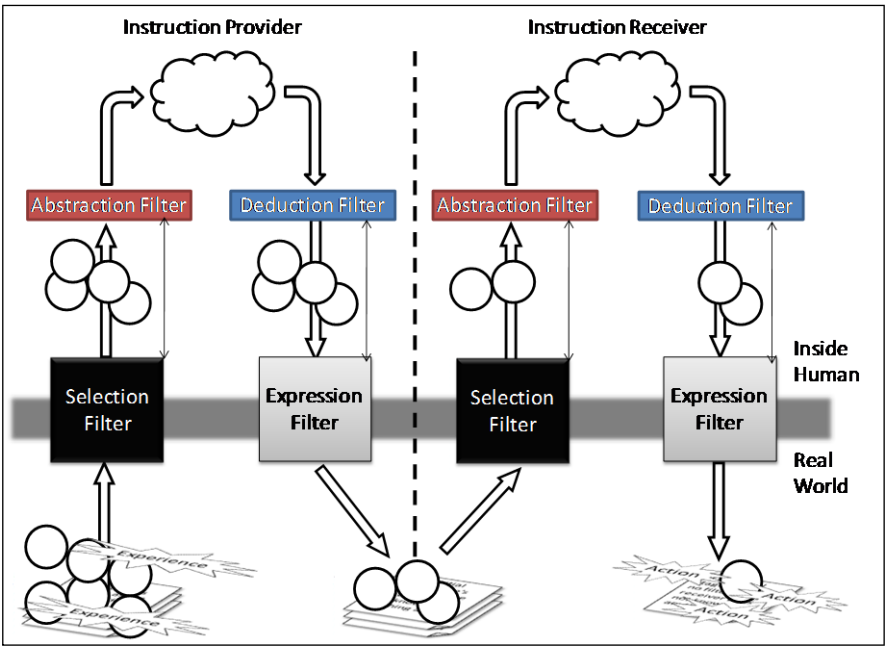

Fig. 2. Illustrative sketch of grand model

Yet another factor impeding communication is the gap of knowledge between provider and receiver of instructions. As we said earlier, decent instructions are given in a consistent level of abstraction and that level is decided so that the intended receivers of the instructions can easily comprehend what the instructions tell them. It is obvious that a problem emerges when that is not the case. Consequently, communication becomes more difficult when the gap is larger between the level that intended receivers can understand and the one that the target principle belongs to. Also, it follows that the higher level of abstraction the intended receiver can understand, the simpler instructions are.

Having discussed reasons for the difficulty in knowledge communication, we are now in the position to present the sketch of our grand model as shown in Figure 2. The instruction provider obtains abstract principles based on filtered experience and knowledge. Abstraction and selection are interrelated to each other. When trying to describe instructions, the provider has to deduce possible instances from principles and describe them along with the principles. If s/he fails to recognize important instances, the resulting instructions will have a potential failure however expressive the description language is. The receiver of instruction receives the instructions and follows the same route as the provider, although there is no guarantee that the receiver understands the instructions as the provider intends. If the instruction receiver in the figure is to give instructions to yet another person, s/he is now a provider of instructions and has to give instructions instead of taking actions, continuing information relay. This is how an organization is managed and its strategy instructed by top managers is disseminated until finally implemented. The figure shows four

types of filters, namely, selection, abstraction, deduction, and expression filters, each of which can be the cause of a communication failure. Some of the failures have been already mentioned, but the entire list is presented here;

Neglect of Instances; receiver who has an incomplete principle still under construction neglects the instances that do not fit the principle well (interrelated to *neglect of principles*).

Neglect of Principles; receiver who already had one principle, complete or not, do not create another one even though there are instances that cannot be explained fully by the existing principle (interrelated to *neglect of instances*).

Failure of Abstraction; receiver recognizes instances but cannot find any principle behind them.

Failure of Deduction; provider who has a relevant principle fails to provide some important instances derived from that principle.

Failure of Expression; provider who has a set of relevant instances in mind fails to express all of them.

3 Case Study

In this section, we investigate a practical manual and see how it can be improved according to our view. The manual used here is the one for a power plant operation. It consists of operation manuals for start-up, shutdown, and emergency situations. The emergency situations part consists of case-by-case references each indicating a sequence of operation. The start-up and shutdown situations parts are further divided into about 40 steps; still each step includes tens of events and operations. Each operation indicates the exact button to operate or measure to observe. Although each step has the label indicating what the step is like in terms of function, within each step is just a sequence of operations, describing neither principles nor alternatives. When studying in detail, however, one can find that each step consists of modular-like patterns of operations, such as pump starting-up pattern, valve closing pattern, pressure raising pattern, and so on. Sets of operations that seem to be an instance of a certain pattern often appear in several steps, but they are different in minute detail; a certain operation seen in one instance is absent from another instance; the order of operations is often slightly different from instance to instance. Of course, in the manual there is no description about to what extent these instances can be explained by a single principle. If there is a principle that can explain most of the instances, we can give principled explanations by combining the principle and instance-dependent variables. That will eventually help readers become more capable of dealing with variable situations since now they can learn the visualized principle. Thus, we performed a trial analysis focusing on the pump start-up pattern.

We investigated seven instances of the pump start-up pattern. Each instance is a sequence of operation on a different pump. The seven pumps are from the four categories which are summarized in Table 1, and the sequences of operations for all the seven pumps are shown in Table 2.

Table 1. Seven pumps categorized by the combination of pressure and type

	Type X	Type Y
High Pressure	HPX(A) HPX(B)	HPY(A) HPY(B)
Low Pressure	LPX(A)	LPY(A) LPY(B)

Table 2. Sequences of operations for seven instances. A check mark indicates that the operation is required for that instance, while a dash mark indicates that the operation is missing from that instance. The sign *confirm* means that it is required to confirm that the target state of that operation is achieved (no action is required). Finally, the numbers indicate that the operation there requires to confirm that a certain measure equals the number.

	LPX(A)	HPX(A)	HPX(B)	LPY(A)	LPY(B)	HPY(A)	HPY(B)
Close Valve A	✓	*Confirm*	✓	*confirm*	*confirm*	*confirm*	*confirm*
Open Valve B	–	✓	✓	✓	✓	✓	✓
Start-up Pump	✓	✓	✓	✓	✓	✓	✓
Pressure (kg/cm^2)	18	50	47	26	26	59	59
Flow (m^3/h)	–	–	–	85	85	250	250
Check Water Level	✓	–	–	–	–	–	–
Onsite Inspection	✓	✓	✓	✓	✓	✓	✓
Valve B→Auto	–	✓	✓	✓	✓	✓	✓
Close Valve B	–	*Confirm*	*confirm*	*confirm*	*confirm*	*confirm*	*confirm*
Check Sample Water	–	–	–	✓	✓	✓	✓
Open Valve A	✓	✓	✓	✓	✓	✓	✓
Check Tank Level	–	–	–	✓	✓	✓	✓
Stop Sub-pump	–	✓	✓	–	–	✓	✓
Next Pump→Auto	✓	✓	✓	✓		✓	
Open standby valve	✓	–	✓	✓		✓	

As shown in Table 2, some operations are seen common in all instances, showing that there are invariant structures. We can also find high-pressure-specific and type-Y-specific invariant structures (e.g., "Stop Sub-pump" and "Flow"). Actually, variables that are thought to belong to no invariant structure do exist but only to a small extent. This suggests that a principled explanation is a feasible option. Probably, a possible principle in this case takes the form of hierarchy in which there are general pump category at the top layer, the high-pressure and low-pressure pump categories at the middle layer, and the four categories of the matrix in Table 1 at the bottom layer. With this principle, we would have three types of general rules each corresponding to one of the three layers. The top layer rule is a generalized version of the middle layer rule, which in turn results from generalizing the bottom layer rule. Then, the description of pump operations would be of the form "a general rule plus instance-dependent variables."

4 Conclusion and Future Work

In this paper, we presented our view and framework for balancing between the flexibility of abstract principles and the preciseness of concrete instances, which aims at helping instruction receivers become capable of dealing with variable situations where no explicit instructions are available. Our investigation is still in the process of trial analysis where we analyze various instructions in a case-by-case manner. By conducting more case studies, we hope that our model will become well-formulated and will be capable of guiding instruction design systematically.

References

1. Rasmussen, J.: Information Processing and Human-Machine Interaction: An Approach to Cognitive Engineering. Elsevier Science Publishing Co., Inc., New York (1986)
2. Knowledge Based Systems, Inc., http://www.idef.com
3. Ross, D.T.: Structured Analysis (SA): A Language for Communicating Ideas. IEEE Software Transaction Engineering SE3((1), 16–34 (1977)