Model-Based Specification and Generation of Programs

Takumi Aida and **Shuhei Kawasaki** and **Setsuo Ohsuga**
Department of Information and Computer Science
Waseda University
3-4-1 Ohkubo, Shinjuku-ku, Tokyo, 169-8555, JAPAN

Abstract

A new modeling scheme named multi-strata modeling is discussed. A model represented in this scheme stands between persons and computer software systems, and connects them directly. It is also possible for users to represent his/her intention to the detail by means of this model. It establishes a new human-computer relation and enables users to represent complex problems and systems including human being. The concept and the basic ideas on this modeling are discussed with an application to the enterprise modeling. In particular it is used to specify programs. It is quite natural because computer program is a part of an enterprise and therefore must be included in the enterprise model.

Introduction

The objective of this paper is to discuss a new method that enables people an easy specification for programming using an advanced information technology. Specification is considered in this paper as an intermediate stage of problem solving. The term problem is used here in a wide sense to mean what a person wishes to know or wants to do. For example, a requirement for making computer program is a problem. This case is discussed in this paper.

Requirement engineering stands almost on the same viewpoint. Requirement engineers should always consider different perspectives of a diverse set of system stakeholders. This is the one of the characteristics of all problem descriptions. Multi-perspective approach is necessary, especially for collaborative works with many people whose background knowledge and representation styles are different. Viewpoint framework is one of the approaches. It advantages consistency checking [FIN94] that is commonly necessary for all problem descriptions. But viewpoint-oriented requirement engineering methods are not widely used because they may be conceptually sophisticated but too strict for all practical purpose. The method that is based on a flexible model of viewpoint and is adaptive for wide range of industrial settings was proposed as PREview [SOMM98].

Requirement engineering is specialized to problem description in the field of programming. In this paper a more general form of problem description is discussed. It leads us to requirement description for programming when it is applied to problem of programming. A new model-based method of problem solving including human being is introduced and program specification is created using this model.

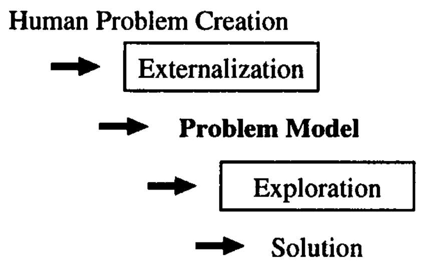
A modeling scheme plays a key role in the method. It must be comprehensive for both persons and computers. It must also be able to accept wide class of problems. Thus modeling scheme satisfying such requirements is discussed first in this paper. It is shown that with such a new method computer systems can not only be intelligent but also be able to deal with wide problems.

This scheme is applied to modeling enterprises. An enterprise is an organization including human being and activity of an enterprise is becoming more and more complex and dynamic. It is a good example for testing the usefulness of the method discussed in this paper. An enterprise modeling and program specification as a part of the modeling are discussed.

Model Building as Problem Representation

Roughly speaking problem solving is formalized as problem creation, problem model building, making a plan of problem solving, finding problem-solving method, (translating the method into a program if necessary), execution of the method (or program) in computer, displaying the result and acceptance of the solution by the person. Since problems originate from persons, there is intrinsically a problem of human-computer relation. Currently computers can commit only a small part of this process around program execution because of the difficulty of specifying and solving problem. It is necessary to establish the more intimate relation between them in order to ease programming.

Let the above process be divided into two stages; one is creation and representation of problem by persons and the other is computer's problem solving. Let the formal representation of problem by person be called the problem model. The first part and last part of problem solving are connected by the problem model and called here externalization of person's idea, or simply externalization, and exploratory problem solving, or simply exploration, respectively. Problem solution is obtained by a sequence of model transformations. Thus every problem solving style can be represented,



The location of transferring problems from person to computer depends on the problem solving style. In the conventional style, the problem model is represented in the form of program. Making program is a large burden to persons. In order to reduce it, it is necessary to change this style and to put the location of problem transfer to the uppermost position as far as possible.

Representation of problem model must be decided depending on the problem-solving capability of computers. The capability depends on the problem solving style. In this paper a knowledge-based, exploratory problem solving is used.

In many cases it is required to the exploratory problem-solving system to generate programs instead of generating solutions directly. In this case the problem model representation becomes a program specification. Thus the followings are the major issues to be discussed.

1. How to represent the problem model.
2. How to support persons to externalize his/ her idea (Externalization).
3. How to deal with the problem model to get solution and generate program (Exploration).

The requirement for externalization is easiness of describing problem by person while that of exploration is autonomy, generality and practicality of problem solving [ROS97].

Generality requires the system not only to cover different domains of problems but also to deal with different types and maturity of problems. It requires systems to provide a large knowledge base covering the different types and domains of problems.

Practicality contradicts very often to generality. If only a necessary subset of rules are provided in advance for solving the given problem instead of using large knowledge base, then this problem can be resolved considerably. Furthermore, if redundancy is dispelled completely, then it is equivalent to a procedural program. Ohsuga proposed a way of generating a problem solving system that is specific to the problem in [OHS96]. It leads us further to generation of programs because program is a special form of an automatic problem solver [OHS98].

Problem model plays many roles. First of all, it is a representation of a problem as a requirement to be satisfied. It must be comprehensive for persons. Next, it is modified in a computer system until a solution is reached and must be computer readable. Thus the model represents a state of problem solving process at every instance of time and a solution in its final state. Every decision is made on the basis of the model representation.

In general problem is created concerning with an object by person who has some interest in it. Thus it is necessary to describe the object precisely for representing the problem. It is not to represent everything of an object but of some aspects in which the person has interest. It is called an object model. Even with the same object, the object model can be different by the interest of the person to the object. Hence, the interest must also be included in the problem model as well as an object model. Thus there are two different schemes for modeling. A real model is represented by their combination. A language suited for representing these models was necessary and developed. It had to be suited for representing predicate including datastructure as argument and also for describing higher level operation such as knowledge for selecting object knowledge. KAUS (Knowledge Acquisition and Utilization System) has been developed for the purpose by our research group.

Modeling Scheme

Object model as a bottom-up model

An object model is a formal representation of an object in the world. Every object model is represented as a related collection of the conceptual constituents included in the scope of personal interest. There are two types of constituents; a structural component and functionality. The former forms a whole-part structure of the object. A structural component and a structure of the components are called here the conceptual entity or simply entity inclusively. An entity is not always a physical object but also can be a non-physical one.

Every entity has some functionality. The term functionality is used to mean inclusively attribute, property, function, behavioral characteristic of the object and of its components and relation with the other objects/components. Functionality is defined with respect to some entity and is represented by a predicate. Let such an entity in the predicate be called an argument. When two or more entities are combined to form a compound entity, a new functionality is created to the resultant compound. It is decided uniquely depending on the functionality of its components and the structure of the entities. Let a whole-part relation be represented as a hierarchical structure. Then an object model is represented as both a structure of entities and a related structure of functionality. The predicates as representations of functionality are related to each other indirectly through the structural relation of the entities that they contain as arguments.

If a structure of entities and functionality to every entity in this structure in the aspect in which a person has interest can be represented explicitly, a complete object model can be made. If some part of the model is left unspecified, it is an incomplete model. If there is some way to get the unspecified part based on the specified part, it represents a problem. An activity to fill up the unspecified part is problem solving. Different types of problems are defined depending on what part is unspecified in a model.

In addition to object knowledge, knowledge for selecting the object knowledge from a large knowledge base such that the problem solving process reach the goal as fast as possible is also provided.

Multi-strata model (MS-model) — a top-down model including human being

A modeling scheme that describes not only an object but also a subject, which deals with the object, in the context of a relation between them is necessary. Functionality of the subject defines a scope of model representation. More-than-one subjects can be included in a model. Sometimes person may have interest not in an object directly but in a problem solving by others. Then a problem solving activity by others is an object for this person. This is a problem solving too. In this case a problem solving is represented as a nested structure. Let it be called a multi-strata object. A new modeling scheme is necessary to represent such multi-strata objects as shown in Figure 1(b).

Modeling for automatic programming is a typical example. It requires three strata model. A program is what generates a solution (output) to a given problem (input) in relation with an object. It is therefore a special type of automatic problem solver. It must be the same as the work done by a person $S1$(in Figure 1(a)). A requirement such as "process transactions –" may be given to a subject. Programming is an activity of a higher level (second-stratum) subject $S2$ to create such an automatic problem solver. A requirement such as "make a program for –" may be given to this subject. An automatic programming is an activity by a still higher level (third-stratum) subject $S3$ to generate automatically an automatic problem solving system. It corresponds to a person who makes the program. A requirement such as "automate programming of the second subject" may be given to this subject. Thus a multi-strata model is also a hierarchical model but there is a substantial difference between the meanings of object hierarchy and subject hierarchy. Different from the former, functionality of the latter is decided freely without being constrained by the lower stratum functionality. Rather the higher stratum functionality (requirement) can affect the lower stratum functionality or, in some case, creates the lower functionality. Thus a multi-strata model is processed top-down.

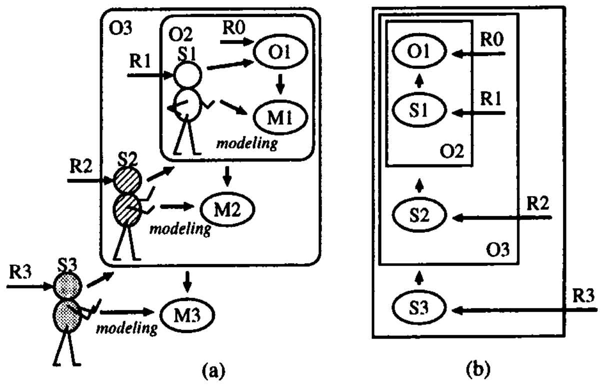


Figure 1: Multi-strata object and multi-strata modeling

Enterprise Modeling as an Example

Many organizations including persons, for example enterprises, can be represented by means of a multi-strata model. An organization can be seen in the different ways in making a model. It can be seen as an organization of persons who have their own roles. Or it can be seen more abstractly as a structure of activities. The former view is popular but the latter view is adopted here because it is more intrinsic than the former. In fact, persons can be changed without changing the activity of an enterprise but the reverse is not true in enterprises.

Every enterprise has an objective. In order to achieve the objective, various activities have been defined such as business planning, sales planning, production planning, sales management, stock order arrangement, material arrangement, and so at the upper part and production, sales, processing transactions in the field in the bottom through the long experiences. If these are not enough for representing the objective, some specific activities to the objective must be defined and knowledge for representing them must be created.

Modeling is a dynamic activity to build up an organization depending on the decision made at the upper level subjects. For example, 'management planning' is a role of the subject at the top (i.e. executive board). If the scale of the enterprise is large, it is difficult to make every decision directly here but the work is decomposed into parts. For example, it is decided to separate production planning and sales planning from the total management planning and make the separate divisions that are responsible to make these decision as well as the framework (regulation, environment, condition etc.) and budgets of the activities and the rights given to them. Some activity is remained by subtracting these parts from the original management planning. This is the actual activity of the top subject.

In this way, a multi-strata object is formed. Even though the lower stratum activities depend completely on the upper subject's decision, they can work independently after the decision is made and an organization has been settled for the remained activity. An activity is in reality composed of many small tasks. What tasks are assigned to a subject is decided referring to the past cases.It is not an easy task to do everything from scratch but it is easier for persons to modify the similar model. It is possible to refer to the similar models that have been made before and accumulated as the domain specific knowledge. The effectiveness of making an incipient model, even though it is not correct enough, is that it can be a base of thinking of persons. Knowledge is used as much as possible.

Let an activity be represented by a predicate in such a form as *activityName* (Var_1, Var_2, $\cdots$, Var_M, *condition*, *budget*), where '*activityName*' is a predicate to represent the activity, Var_1, Var_2, $\cdots$, and Var_M are the variables included therein.

If the activity is what has been established before, then its interpretation rule in the knowledge base and retrieved easily. In general a rule to represent an activity must be of such a simple form as,

$$\begin{aligned}&\mathit{activityName}(Var_1, Var_2, \cdots, Var_M) : -\\ &\quad \mathit{primitiveOp}_1(Var_{i1}, \cdots, Var_{ir})\\ &\quad \cdots\\ &\quad \mathit{primitiveOp}_n(Var_{k1}, \cdots, Var_{ks}).\end{aligned}$$

where $\mathit{primitiveOp}_n$ is a predicate to represent a primitive operation to construct the activity and $Var_{k1}, \cdots, Var_{ks}$ a subset of (Var_1, Var_2, $\cdots$, Var_M), are the variables included in this primitive operation.

Program Specification and Generation

All activities in a model are classified by the relationship between variables. Let some variables that are specified by the other activities or by the external world directly and that are referred by the other activity or the external world be called the inputs and output respectively. The remaining variables are called the auxiliary variables.

For a predicate representing an activity, if the remaining variables can be evaluated based on the input in such a way that the predicate becomes logically true, it is called computable. Otherwise the predicate is non-computable. The auxiliary variables are used for evaluating the output together with the input variables and must be obtained by the separate method from the evaluation of the predicate. For example, some additional activities are necessary to get information that is required for making decision in an enterprise. Let these be called the supporting activities. These activities are added to the main activities and classification proceeds including them. The computable predicates are also classified into two classes. The one class is of activities that are completely specified to the detail and the remaining ones. These are called programmable and non-programmable. Thus the activities including supporting activities are classified into three classes; programmable, computable but non-programmable and non-computable. The classification of activities is the role of person.

In the main activities, those that concern to the lowest strata subject are mostly in the computable class. Many activities of high-strata subjects on the other hand are of the non-computable.

The computable class is the object of programming. Some computable activities are collected to form a macro activity that is a candidate to be translated into a program. The different collections of the activities are made. If all activities in a collection are programmable, this set of activities can be translated into a program. That is, a program specification is made from this set. A new predicate is generated for the set to represent a program to be generated therefrom. It includes the variables that come into this set from outside as input and the other variables going out of this set as output. Then a rule is made of which the conclusion is the new predicate and the premises are the predicates in the set. This is a program specification.

If a (or a few) activity is computable but non-programmable in this set, then an interactive program is generated from the collection. A program specification together with the specification for the human interface is made. In these cases, if an activity refers a (part of) structure in an object model, the predicate to represent the activity includes this structure as a data structure. This becomes a data structure of the program. A computer program can be developed from the specification directly. Programming starts by issuing a question on a predicate to its specification.

The outline of the automatic program generation

If there is enough amount of knowledge to interpret every activity included in the specification, it is processed as an ordinary knowledge-based problem solving. It is considered that a succeeded path to solution is a procedure for deducing the solution. In the following, the method of program generation from problem solving process of knowledge-based system is described.

1. A sample task is made by substituting arbitrarily selected constants into variables. This sample task is processed actually.
2. A deduction tree is generated by tracing the problem solving of the sample task by a knowledge-based system. The deduction tree generated from sample task is called an instance tree.

3. An instance tree represents a program structure suited for processing the sample task. Another sample task in the specified domain may generate another deduction tree. Therefore it is necessary to generate a generalized deduction tree that covers all sample tasks in the domain. The instance tree is generalized for the purpose by restoring the constants to the variables with the originally specified domains. The deduction tree thus generated is called a general tree.

4. The basic program structures such as a straight procedure, a branch, a loop corresponds to the specific structure in the deduction tree. These program structures are identified in the tree and are marked. These are macro components. Every terminal node (leaf) represents an atomic procedure and those that have no marked node along the path from the top to the nodes are also marked. The order of these marked nodes being called decides the sequential order of the macro-components, and therefore a program structure in a program. A program results by arranging the components according to the sequential order. The same procedure is achieved within every macro-component to generate the structure in the lower level.

Application to an Actual Problem

This method of program specification and generation has been applied actually first to a real problem, a control unit programming for car electronics system, before applying directly to the more complex problems like enterprise programming. The characteristic of the car control system is that the hierarchy of activities is shallow with the programmable activities as the leaves. Some activities are collected to form a macro and accordingly a hierarchy is formed. The hierarchy is made for the purpose of making the control system comprehensive for human designer. A program specification in an ordinary style has been generated from this model. The characteristics of the control system are rather well known through the experiences so far and it is easy to select sample tasks to cover every necessary program structure. Therefore an interactive method of programming has been adopted in making a general deduction tree. Some sample trees are generated and merged to a larger tree instead of generalizing an instance tree by a rule-based operation. The latter approach is necessary for automating the programming. Except these, the modeling concept is substantially the same as the one discussed before. In the following the method of analyzing the deduction tree for making the program structure is discussed for the case of generating a branch because a branch structure is a fundamental component in order to compose other kinds of basic program component.

Generalization of an instance tree

Every node is created as the deduction of a query predicate using rules. There are two kinds of node, *AND node* and *OR node*, in a general deduction tree. An AND node is created when a query predicate is deduced by a conjunctive rule, i.e. the rule of which the body is a conjunction of predicates. These predicates are the children nodes of the AND node. If the processing fails during unfolding a child node, the other children nodes are not processed but a backtracking occurs. The AND node is erased and the other rules are explored for deducing from the original query.

On the other hand, an OR node is created either when a rule includes disjunction explicitly like,

$$\begin{aligned}&\mathit{predicate}(X\ Y\ Z)\\ &\quad :-\ \ \{\ \mathit{cond\_1}(X),\ \mathit{body\_1}(Y\ Z)\ \}\\ &\quad +\ \ \{\ \mathit{cond\_2}(X),\ \mathit{body\_2}(Y\ Z)\ \}\\ &\quad +\ \ \mathit{body\_3}(Y\ Z)\end{aligned}$$

or when there are rules with the same head but with the different bodies, like,

$$\begin{aligned}&\mathit{predicate}(X\ Y)\\ &\quad :-\ \ \mathit{p\_1}(X),\ \mathit{p\_2}(X),\ \underline{q(X\ Y)}.\end{aligned}$$

$$\begin{aligned}q(X\ Y)\ &:-\ \ \mathit{cond\_1}(X),\ r\_1(X\ Y).\\ q(X\ Y)\ &:-\ \ \mathit{cond\_2}(X),\ r\_2(X\ Y).\\ q(X\ Y)\ &:-\ \ \mathit{cond\_3}(X),\ r\_3(X\ Y).\\ q(X\ Y)\ &:-\ \ \mathit{cond\_4}(X),\ r\_4(X\ Y).\end{aligned}$$

the different instance tree are generated by selecting the different rule for the different sample task and these are merged.

An OR node corresponds either to a loop or a branch depending on the predicates in the lower nodes. If the predicate with the same predicate symbol as the original query predicate appears in a lower node, the structure starting from the query node to the lower node is translated into a recursive program or a loop program. Otherwise it corresponds to a branch. The former type OR node can appear in an instance tree but the latter type node appears by merging the different instance trees for the different sample tasks. In this case what rules are selected, and therefore what OR structure is made, depends on the domain of variable(X) in the original predicate. In the former case, on the other hand, the OR structure is decided uniquely by the rule. Therefore it is desirable that for the first type the rule is represented in a disjunctive form while the latter case is represented by a set of separate rules. KAUS as mentioned before allows a disjunctive representation of a rule and has been used for the application.

Procedures generation from general tree

A program is generated from the general tree obtained in the previous step. The process of generating a general tree and extracting program is stratified. In the general tree the node corresponding to the basic program structures like branches and loops are marked. The predicate at the leaf node that has no OR node along the path from the top to the leaf represents a primitive operation and also marked. Then the marked nodes(predicates) are chained from the left most node in the tree. A sequence including macro operations are generated. Then, for each OR node corresponding to this macro a lower structure is generated, a sequence of predicates is generated for this structure and the resultant sequence is connected to the previously generated sequence of predicates. The same procedure is repeated until every path from the top node comes to the leaf(Figure 2). Finally, the result is translated into

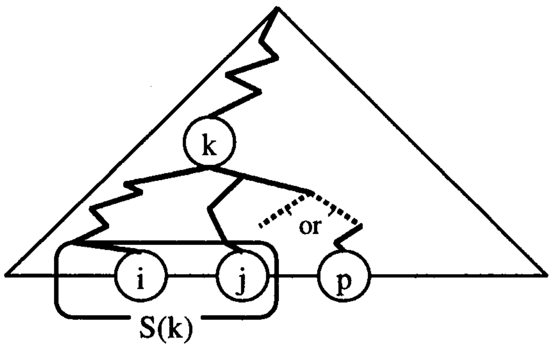


Figure 2: The set of primitive tasks which execute successively

program code. It is now in the intermediate form and a program to translate it in a C-code is being developed.

This method of program specification and automatic generation of program has been applied for generating a program of an engine control system.

First, a specification in the traditional style has been generated from this internal specification. This proved that the making an internal specification as mentioned above can be connected to the ordinary process of programming. In this process, a program has been generated at the same time. At the moment, there are still many problems. The generated program is not an optimal one and its optimization is necessary for real application. In particular there is time requirement for this kind of control programs. It must be included in the specification. Nevertheless the author believes that this is an important step toward a new era because a computer control is recent tendency for not only car electronics but also many other industrial products and necessity for the rational method of developing this type of software is rapidly increasing.

Conclusion

A new modeling scheme named multi-strata modeling was discussed. A model represented in this scheme stands between persons and computer software systems, and connects them directly. It is also possible for users to represent his/her intention to the detail by means of this model. It establishes a new human-computer relation and enables users to represent complex problems and systems including human being. The concept and the basic ideas on this modeling were discussed with an application to the enterprise modeling. A part of this idea has already been implemented and used to represent generation of a program specification for a car electronic system in cooperation with a car production company. A program has been generated based on this system. The modeling scheme can be extended further in many directions to represent and deal with complex problems.

References

[FIN94] A. Finkelstein, et. al.; Inconsistency Handling in Multi-Perspective Specifications, IEEE Transactions on Software Engineering, Vol.20, No.8, 1994

[OHS96] S. Ohsuga; Multi-Strata Modeling to Automate Problem Solving Including Human Activity, Proc.Sixth European-Japanese Seminar on Information Modelling and Knowledge Bases,1996

[OHS98] S. Ohsuga; Toward Truly Intelligent Systems - From Expert Systems to Automatic Programming, Knowledge Based Systems, Vol.10, No. 3, 1998

[ROS97] F. H. Ross; Artificial Intelligence, What Works and What Doesn't ? AI Magazine, Volume 18, No.2, 1997

[SOM98] I. Sommerville, P. Sawyer and S. Viller, Viewpoints for requirements elicitation: a practical approach, International Conference on Requirements Engineering, 1998

[SUM96] Y.Sumi, et. al.; Computer Aided Communications by Visualizing Thought Space Structure, Electronics and Communications in Japan, Part 3, Vol. 79. No.10, 11- 22, 1996