AN IMPLEMENTATION OF A MODEL MANAGEMENT SYSTEM[①]

Chen Xiaohong
Department of Business Administration
Central South University of Technology, Changsha 410083
Yasuhiko Takahara
Tokyo Institute of Technology

ABSTRACT The model management system of actDSS, based on the general systems theory (GST) has been discussed. Although it employed the model integration approach, it was different from the conventional approaches. This paper presented the model management system in the following way: a concise introduction of GST and actDSS, discussions of functional model, model description language, MIE, process support, and finally demonstration of an application to a practical problem.

Key words general systems theory(GST) model management model description language (MDL) model integration environment (MIE) decision support system (DSS) menu system annual production planning

1 INTODUCTION

The authors have engaged for several years in the research and development of a DSS, called actDSS, which adopted a model integration approach (MIA) for its model management. An excellent summary of MIA in Ref. [1] showed that model representations for model integration can be classfied into three schools, namely structured modeling, logic modeling and graph grammar. This paper, however, presents another scheme of model representation called general systems theory (GST) and shows a model integration environment (MIE) implemented on it.

The mathematically general systems theory (MGST) has been developed to lay a solid foundation for systems engineering in general[2-5], and hence the MGST must be a good framework for the model integration approach where various models and various paradigms of modeling must be handled in a uniform way.

2 STRUCTURE OF actDSS

The actDSS is a DSS generator based on UNIX system. Fig. 1 shows its " hardware "

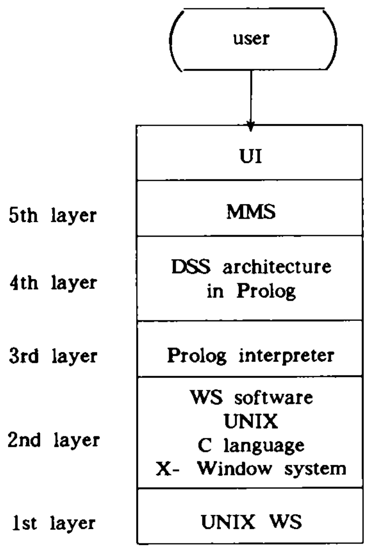

Fig. 1 Hardware structure of actDSS

① Supported by the JAPAN government research foundation; Received Jul. 18, 1995

structure. Every object is defined as a window and the access to the object is realized as an access to its window. In this sense actDSS is intrinsi cally an object oriented system.

The 3rd layer of Fig. 1 shows one of the most important features for the model management system (MMS) of actDSS. The Prolog interpreter of the 3rd layer was designed and developed specifically for actDSS. The Prolog is an extention of the conventional Prolog and most of the basic predicates necessary to describe functions of a DSS generator are written in C and Prolog can call them as its subroutines.

The 4th layer, DSS architecture, is the other important feature of actDSS. The architecture is really an operating system of the DSS, and is called actOS. All basic functions and the control mechanism of actDSS are defined here in the Prolog. Since a user can modify actOS rather easily, we claim that actDSS is a customer-made system.

The 5th layer consists of basic utilities of system including the MMS. Their control structures are also written in the Prolog.

All user models and the utilities are internally represented in Prolog forms. When they are to be executed, they are first linked to actOS dynamically and then their executions are carried out by the Prolog interpreter. The link operation is called load. Fig. 2 shows the load operation of actDSS. Therefore, a specific DSS is always a large Prolog program although its size and structure are dynamically changing . Due to this special execution structure dynamic model creation, model deletion and model modification are realized.

Fig. 3 shows the process structure of actDSS. An input of a user is accepted by the window manager (WM) of actDSS. If the input is simple, for example, a request to the spread sheet, the WM processes the input directly and outputs a response. If it is complicated, the WM sends an interrupt signal and the control of the system to actOS. The actOS then selects a suitable subsystem (or subprocessor), for instance, the MMS and requires it to process the input, and after outputing a response it returns the control to the WM, then waits for the next input from the user.

We will discuss the model management of actDSS in the following steps:

(1) Model description language (MDL) of actDSS and its internal representation

(2) Integration environment of actDSS.

3 MDL AND ITS INTERNAL REPRESENTATION

The MDL of actDSS is a functional language. As will be shown, this fact makes it possible to yield the state space representation

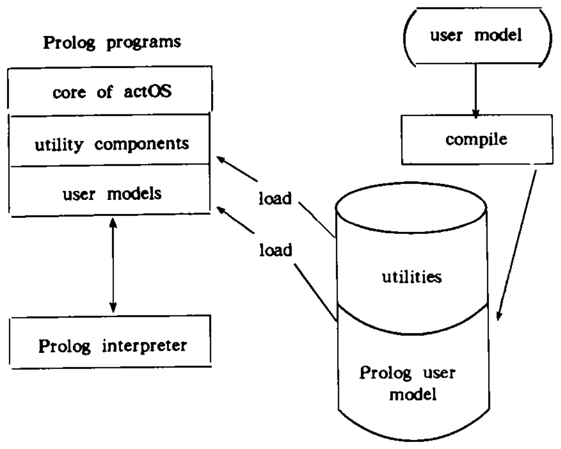

Fig. 2 Load operation

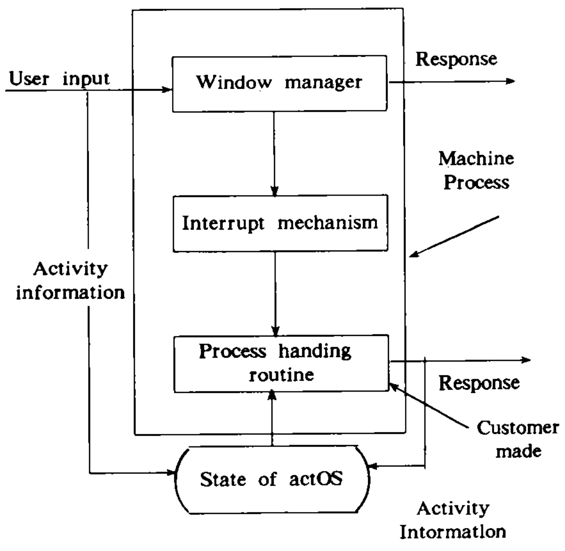

Fig. 3 Process structure of actDSS

of model easily and prepare the model integration environment of actDSS. Fig. 4 illustrates a model written in it. Actually the language itself is quite powerful in the sense that it can simulate the Turing machine[6], or it can describe any algorithm in principle.

The MDL has three important features for model management: (1) It is translated into a Prolog form internally; (2) It dose not have type declaration; (3) It can specify the message variables to a model.

Let us discuss these features in detail.

An MDL program is internally translated into a Prolog program. Suppose we have an MDL statement like

$v = f(v_1, \ldots, v_n)$

where v, $v_1, \ldots$, *and* v_n are variable names. Let us denote the above equation by S. Then, internally, it is translated as

$v(Y) : - v_1(X_1), \ldots, v_n(X_n), f^*(X_1, \ldots, X_n, X), Y \; is \; X, !.$

where $f^*(X_1, \ldots, X_n, X)$ succeeds if and only if $X = f(X_1, \ldots, X_n)$.

Let us denote the translation by $\Phi_1(S)$.

Then, the execution of the program is done in the following steps.

Step 1: The MMS of actDSS asserts initial data as facts.

Step 2: The Prolog interpreter evaluates the predicate $f(X_1, \ldots, X_n, X)$ using the asserted facts $v_1(X_1), \ldots, v_n(X_n)$ and yields the value Y $(= X)$ of the variable v to the MMS.

Step 3: The MMS asserts v (Y) as a new fact and asks the interpreter to evaluate another new variable, that is, Step 2 is executed. If, for instance, $v_1(X_1)$ is not asserted yet in Step 2, the interpreter tries to evaluate the variable v_1 as its subgoal. But if the evaluation of the variables is done in an appropriate order, such a case never happens, which will be discussed in the next section.

Since the Prolog is used for the internal representation of an MDL program, we can have at least two advantages:

(1) The MDL becomes a non-procedural language. Then order of statements is of no importance.

(2) An MDL program can include Prolog programs as subroutines in a natural way.

Let us consider the second advantage.

Suppose $f(x_1, \ldots, x_n)$ is used in an MDL program. Then, it is translated into the form

$x_1(X_1), \ldots, x_n(X_n), f^*(X_1, \ldots, X_n, Y)$

as mentioned and therefore, f can be specified by the Prolog rule of

$f^*(X_1, \ldots, X_n, Y) : -$ (body of the rule)

where $X_1, \ldots, X_n$ and Y are Prolog variables.

As Fig. 4 shows, Prolog rules are directly listed in an MDL program surrounded by the special control symbol "/%" and "%/". Internally, there is no distinction between the MDL part and the Prolog part. This feature makes the MDL powerful and flexible.

Let us consider the second feature that the MDL has no type declaration or a variable is typeless. This feature also comes from the fact that the Prolog is used for the internal representation of an MDL program.

In Fig. 4, the variables "a", "b", "c" and "_sol" of _sol=lpsolver (a,b,c) represent, in reality, a matrix, a vector, a vector and a vector, respectively although they are not declared as such. The type of a variable is determined by the situation how it is used. This is crucial for the dynamic (or on line) model integration where no information is given about the types of variables to be linked beforehand.

A very useful feature of the MDL for model integration is that it can be used to specify the message variables to and from a model explicitly. As will be discussed in section 4, every model is finally represented as an input-output relation (IORep) in actDSS and the message variables of the model are input and output variables. The variables to be displayed as inputs and outputs are those and only those which are used in MDL statements, except a hidden variable whose name has "_" as its initial letter like "_sol" and which is not displayed in the IORep, (A user or other models can not access to a hidden variable). This feature is very convenient, for instance, for solving the so called overlapping variable problem[7]. What we have to do is to introduce a function consisting of overlapping variables in-

```
    //lpsolver. m
    //this model calls the linear prog. solver written in C
    //a=matrix;b=vector;c=vector
    //problem c * x→max;a * x<=b;x>=o
    //__sol=[X,c * X] where X is the solution __sol is a
hidden variable
    __sol=lpsolver(a,b,c)

    //get X=sol from __sol
    sol=project (__sol,1)
    //get c * X=performance
    perform=if sol=[] then "infeasible" else project (__sol,
2)

    //get coeff. of sales from objpara. s
    __salesC=vec(objpara. s,[1,2,1,5])
    //get sales w. r. t the solution
    sales=if sol=[] then 0 else sum (sol * __salesC)

    //get coeff of taxes
    __taxesC=vec(objpara. s,[2,2,1,5]
    //get taxes w. r. t the solution
    taxes=if sol=[] then (   ) else sum(sol * __taxesC)

    //get coeff of quantity
    __quantityC=vec(objpara. s,[3,2,1,5])
    //get quantity w. r. t the solution
    quantity=if sol=[] then 0 else sum(sol * __quantityC)


    //start of Prolog program
    /%
    /* call LP-solver */
    /* ( A, B, C ) = LP problem; Y = ( solution,
performance) */
    lpsolver(A,B,C,Y):-!,
    procC("lpsolver",[A,B,C],[X,P]),
    Y:=[X,P];
    %/
```

Fig. 4　Lpsolver. m

to an MDL program and then the necessary variables are represented in the IORep in the typeless form. The MDL has many special functions as other MDLs do. Typical ones are "gal" and "var" functions. The "gal" function is a difference operator. For instance, the MDL statement:

sales=gal (sales)

means a difference equation

sales (n+1)=sales (n)

A dynamical system is described in actDSS in this way.

As will be discussed later, "var" function is used for data transfer, communication and control among models. The format of the function is:

var (model name, variable name).

A model can get a data from another model or assign a data to it using a "var" function.

4　MODEL INTEGRATION ENVIRONMENT OF actDSS

actDSS is based on GST to construct its model management. That is, the target of the model management is a realization of the hierarchy system of Fig. 5 and hence a model integration approach has to be adopted. Furthermore, every model is recognized as an input-output system and the interacted system. Fig. 1 is a model of a model integration, and when a model by the MDL is compiled, it is transformed into a state space representation.

Suppose an MDL model m is given by

$$m = \{v_i = f_i(v_{i1}\cdots,v_{il}\,|\,i = 1,\cdots k)\}$$

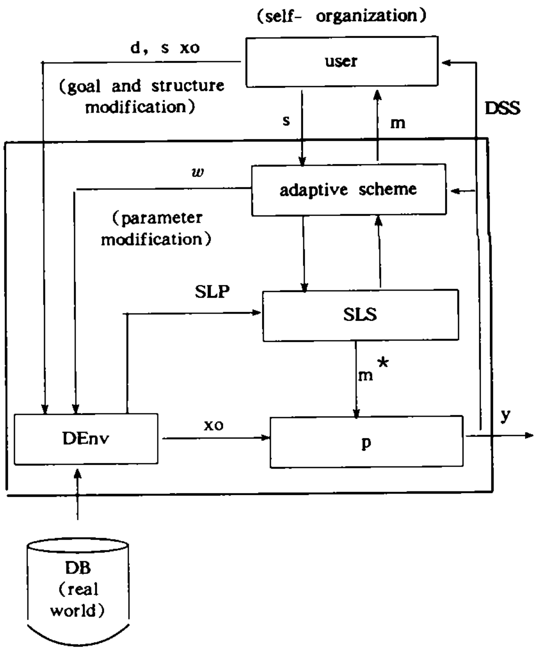

Fig. 5　Hierarchical functional model of DSS

Then, the internal representation of m, Int (m) is given by:

Int (m) = ⟨process rep, structure rep, IORep⟩.

The process rep is the Prolog translation of m which is discussed in section 3.

That is, process rep = $\{\Phi_1(f_i(v_{i1}, \cdots$

$v_{i1}))\,|\,i = 1, \cdots k\}$

The structure rep represnets the dependency relations among MDL variables of m. Each MDL statement yields a predicate "node" as follows:

$v_i = f_i(v_{i1}, \ldots, v_{i1}) \rightarrow$ node $(v_i, [v_{i1}, \ldots, v_{i1}])$.

Then

structure rep $= \{$node $(v_i, [v_{i1}, \ldots, v_{i1}])\,|\,i = 1, \ldots, k\}$.

A model is displayed on the screen of a computer as IORep where formulation is a little bit complicated. Let V_{name} be the set of variable names of m. Let V be the set of values which MDL variables take. V consists of integers, reals, vectors, matrixs, symbols and others. Let

Link$=\{\Phi$, var $(\langle$mname$\rangle, \langle$varname$\rangle)\}$

where Φ is the empty and mname and varname indicate a universal set of model names and a universal set of variable names, respectively.

In actDSS it is assumed that the dependency relation is loopfree or the structure rep can be expanded into a tree (or more generally a lattice structure). Then, basically, the leaf variables of the tree are inputs, variables specified by the difference operator "gal" are states and the remaining are outputs. In this way, V_{name} can be partitioned as:

$V_{\text{name}} = I_{\text{name}} \cup S_{\text{name}} \cup O_{\text{name}}$

where I_{name}, S_{name} *and* O_{name} correspond to input, state and output variables, respectively.

Then the IORep of m is

IORep$\subset(I_{\text{name}} \times$Link$\times V)^* \circ (S_{\text{name}} \times \{\Phi\} \times V)^* \circ (O_{\text{name}} \times \{\Phi\} \times V)^*$

where A^* is the free monoid of a set A and "o" is the concatenation operation.

The set Link represents link relations among variables. How a link ralation is set up will be discussed later. Let us consider a case where an input variable v_1 of a model m_1 is linked to a variables v_2 of a model m_2. Then, IORep of m_1 has the following component with respect to v_1:

$(v_1$, var(m_2, v_2), evaluation (var$(m_2, v_2)))$

where evaluation (var(m_2, v_2)) represents the current value of v_2 of m_2.

Since the values of state and output variables cannot be determined by other models, the set corresponding to Link is $\{\Phi\}$ for them.

The structure rep is used not only for classification of the variable types but also for determination of a proper execution sequence (Refer to section 3). By expanding the tree structure of the structure rep using the depth first strategy, we can have a linear ordering for V_{name}. It can be proved that if the evaluation of variables is done in that order, in step 1 of section 3 the Prolog interpreter never fails to get asserted facts $v_i(X_i)$[8].

It should be noticed that the concept of structure representation of models is also applicable to dependency relations among models. If input variables of a model m depend on variables of models $m_1, \cdots, m_k$, then the dependency relation is expressed by node $(m, [m_1, \cdots, m_k])$. We also assume the structure representation among models is loopfree. Then, expanding the tree structure into a linearly ordered form as mentioned above, we can get a correct execution sequence. This fact is more crucial than determination of the execution sequence of variables.

The representation of the IORep of a model is made on the spread sheet of actDSS. actDSS has one big spread sheet and a portion of it is allocated to each model for its IORep. The allocation is managed by the MMS. The important feature of the spread sheet of actDSS is that its data structure is exactly the same as that of the Prolog or "exp" and hence models can make access to and control the spread sheet directly.

Since the spread sheet is flexible, we can write almost every thing in its cell. The link mechanism and the dynamic data exchange among models are realized by this feature with the help of the direct access capability. Suppose a user wants to link a variable v_l of a model m_1 to an input variable v_2 of a model m_2. In this case, as mentioned above, the link is realized by assigning the function var(m_1, v_1) to the value part cell of v_2 of the IORep of m_2. When m_2 is executed, var(m_1, v_1) is already evaluated due to the calculation capability of the spread sheet and the evaluated value or the value of v_1 is for v_2 and hence the execution of m_2 is done keeping the link relation to m_1. The function var(m_1, v_1), in general, can be typed into the cell by the keyboard. In actDSS, however, the link is realized by the mouse operation also. If a user clicks the mouse on v_1 of m_1, an interrupt signal is sent to actOS (refer to Fig. 3). actOS interprets the signal and waits for the next mouse input. When the user clicks the mouse on v_2 of m_2, actOS writes the function var(m_1, v_1) in the cell of v_2.

In section 2 the multi-layer structure was introduced as a hierarchy concept of GST. Another hierarchy is used in the model integration. The actDSS allows a user to organize submodels in a hierarchical way. Section 5 shows that submodels, genobject. ***m***, blpdata. ***s***, genconst. ***m***, lpsolver. ***m*** and kb. ***m*** are grouped as one composite model, ***c***-model called lpca. ***c***. Then the user can treat the group of submodels as one model. Even a ***c***-model itself is recognized as an input-output system. By executing the ***c***-model, the submodels can be executed automatically in the proper sequence identified by the system by using the interaction structure as mentioned above. In this fashion submodels can be grouped and groups themselves can be grouped yielding a hierarchical structure.

5 APPLICATION

This section shows the case where actDSS is applied to a practical problem, the implementation of a specific DSS for annual production planning decision of a nonferrous company. The original system was implemented using the conventional tools (FOXBASE, Fortran and Turbo-Prolog)[9] and used in Changsha nonferrous company of China.

Due to the shortage of the space we will show a summarized form of the implementation on actDSS[10].

The decision problem is an extended product mix problem. Suppose we have five products.

These products will be indexed 1, 2, ⋯, and 5. Let $x = (x(1), \cdots, x(5))$ be a decision variable referring to the production quantity of the product 1 to the product 5. Then, the final form of the problem formulation is as follows:

Objective:

total sales income $= \Sigma_i c(1,i) \times x(i)\ i \in \{1, \cdots, 5\}$

gross profit $= \Sigma_i c(2,i) \times x(i)\ i \in \{1, \cdots, 5\}$

total production output $= \Sigma_i c(3,i) \times x(i)\ i \in \{1, \cdots, 5\}$

Constraint:

$\Sigma_i a(k, i) \times x(i) \leqslant b(k)\ k \in \{1, \cdots, 5\}\ i \in \{1, \cdots, 5\}$

The decision problem is a multi-objective. But in order to solve the problem by using an LP algorithm, the linear weighted sum method is adopted for the problem. Let the weight vector be

$\boldsymbol{w} = (w(1), w(2), w(3))$.

Then,

Integrated objective:

$\Sigma_j \Sigma_i w(j) \times c(j, i) \times x(i) \rightarrow \max\ j \in \{1, \cdots, 3\},\ i \in \{1, \cdots, 5\}$

All of the five constraints are not necessarily used for decision making. The choice depends on situations and is expressed by a vector $\mathbf{s} = (s(1), \cdots, s(5))$ where

$$s(i) = \begin{cases} 1 & \text{if the i-th constraint is used} \\ 0 & \text{otherwise} \end{cases}$$

Then, Working constraint:

$\{\Sigma_i a(k, i) \times x(i) \leqslant b(k) \mid s(k) = 1,\ k \in \{1, \cdots, 5\}$

The user is supposed to have desired values for the three objectives which is specified by a vector $\boldsymbol{d} = (d(1), d(2), d(3))$.

The three vector $\boldsymbol{w}$, $\boldsymbol{s}$ and $\boldsymbol{d}$ are controlled by the user. The matrix $A = [a(k, i)]$, the vector $\boldsymbol{B} = (b(k))$ and the matrix $C = [c(k, i)]$ are not directly given by the user. There are six tables which are called databases, aec. ***s***, arm. ***s***, apm. ***s***, amd. ***s***, apc. ***s*** and aop. ***s***. For instance, aec. $\boldsymbol{s} = (a(1,1), \cdots, a(1, k), b(1))$. The first five databases specify A and B and the last one does C. Similarly, $\boldsymbol{w}$, $\boldsymbol{s}$ and $\boldsymbol{d}$ are given by databases, ado. ***s***, acc. ***s*** and adv. ***s***, respectively. The family of the databases is the user's problem image of teh real world and he manipulates the problem through this image. The image is transformed into an LP problem by the DEnv of section 4. The DEnv consists of genobject. ***m***, blpdata. ***s***, genconst. ***m*** and the family of the databases, where the tables are implemented as *s*-models on the spreadsheet of actDSS and genobject. ***m*** and genconst. ***m*** are written by the MDL. The components of the DEnv are connected by the mechanism mentioned in section 4 and the result, an LP problem, is sent to lpsolver. ***m*** and finally kb. ***m***. The lpsolver. ***m*** is modeled by the MDL which calls an LP algorithm program written in C. The solution is sent to kb. ***m***.

The kb. ***m*** or the knowledge base subsystem is used to: (1) evaluate the obtained plans; (2) amend the parameters of the databases to realize the desired value ***d***. The knowledges are obtained from domain experts and, expressed as a set of production rules in Prolog which is called by the MDL part of kb. ***m***. If the solution is not satisfactory, the system modifies the parameter values according to the rules.

The user, who corresponds to the self-organizing system of Fig. 2, modifies ***d*** and ***s*** to get a more satisfactory solution.

The submodels genobject. ***m***, blpdata. ***s***, genconst. ***m***, lpsolver. ***m***, and kb. ***m*** are integrated into one complex model (***c***-model) whose name is lpca. ***c***. This is indicated by the rectangle enclosing the submodels in the modle space. If a user executes lpca. ***c***, the submodels are executed in the order, genobject. ***m***→ blpdata. ***s***→genconst. ***m***→lpsolver. ***m***→kb. ***m***. The order is found by the system on the link command execution as mentioned in section 4.

The whole structure of the annual production planning DSS is saved as a task "app" which is indicated by the menu statement "loadtask production planning app". The task saving is easily done by executing the command "wconfig" of the model box by the mouse.

When an end user wants to use the system, he first selects the "menu. mn" from the model box; then the first menu "menul. mn" appears on the screen. Then if he selects "menu. mn" of the first menu for the production planning DSS, the second menu "menul. mn" appears. If user selects "loadtask" from menul. mn, then the whole configration of the production planning system appears. The user choosing commands of each level of menus, the system can be guided and performed step by step.

6 CONCLUSION

Ref [1] proposed 7 aspects of the problems about model integration. Although it is interesting to examine how the problems are treated in the MMS of this paper, we cannot present discussions about it due to the shortage of the space. It is our conclusion that the problems of schema, process, models and data, and models and solvers are well or reasonably addressed to by the MMS.

REFERENCES

1 Dolk, D R. Decision Support System, 1993, (10): 1—8.

2 Mesaroric M, Macko D and Takahara Y. Theory of Hierachical, Multilevel system. Academic Press, 1970.

3 Mesaroric M and Takahara Y. General Systems Theory: Mathematical Foundation. Academic Press, 1975.

4 Wymore A W. Model Based Systems Engineering. CRC Press, 1993.

5 Bossel H. Modeling and Smulation. (Vieweg, 1994).

6 Lee Jae Kyu. Knowledge-assisted Optimization Model Formulation UNIKOPT, Forthcoming in Decision Support Systems.

7 Ramirez R, Ching C, St Louis R. Decision Support Systems, 1993, (10): 1—10.

8 Takahara Y, Shiba N. In: Proceeding of CAST Conference at Ottawa, 1994, 113—118.

9 Chen Xiaohong, Li Yizhi. J Cent-South Inst Min Metall(in Chinese) 1993, 24(5). 56—60.

10 Takahara Y and Chen X. Office Automation. 1995, 16(2): 92—95.

(Edited by He Xuefeng)