TR — 84 — 004

一個整合性辦公室自動化系統之設計與實作

計 劃 報 告

— Office Information System Design Methodology for An Interactive Office Automation System

主 持 人：郭 德 盛

共同主持人：謝 清 俊

協同研究人員：劉 寶 鈞

研 究 助 理：何 正 信

杜 文 宗

鄭 博 文

中 華 民 國 七十三 年 九 月

摘　　要

設計整合性辦公室自動化系統時，一套正確有效的設計方法，將大有助於系統的正確與效能。本報告即提出一有效的設計方法。我們發展了一個交談式辦公室設計介面（ＩODＩ）提供設計師以交談方式描述辦公室的需求，並據以產生辦公室需求分析語言（ORAL）程式。ORAL語言兼具邏輯程式語言與資料塑模特性，其邏輯推衍能力即可作爲系統執行的基礎。透過ＩODＩ介面，設計師可以分析並模擬設計的辦公室流程以確保其正確性與一致性。

ABSTRACT

An effective design methodology is proposed for the design of office information systems. An Interactive Office Design Interface (IODI) is developed, which guides a designer to specify an office requirements in an interactive way, and then generates a specification program in an Office Requirement Analysis Language (ORAL). The ORAL language has the features of logic programming languages and data modeling, and provides the system with inference capability on the execution of the specification program. And one is able to analyze and simulate the designed office procedures to ensure their correctness and consistency utilizing the IODI interface.

Contents

Chapter 1 Introduction

Regarding the evolution of the office automation, it can be divided into three growing stages [Ho82] [Baum80]. The first stage is the use of automation equipments. The second stage is procedure automation. And the third stage is management automation.

Among the three stages, the second stage is our goal to achieve and the last stage provokes our motivation for this paper. We can see that each office procedure has its goal and that to execute each office procedure is just to accomlish its goal. Using the artificial intelligence theory, we can encode these goal-oriented procedures easily for establishing an office knowledge-base.

We propose that an office automation system acquisites the required knowledge through interactive dialogue with its user. As soon as some conditions occur at executing time, the office automation system can notify the users so that they can manipulate these conditions immediately.

Several office models such as Augumented Petrl Net (APN model) [Zism77], Information Control Net (ICN model) [Elli79] [Cook80], and Form Flow Model (FFM model) [Ladd80] [Tsic82 a] [Geha82] were suggested for the analysis and design of office information systems.

The disadvantage of both the APN and the ICN models is the lack of inter-office procedure communitation mechanism. The procedures to communicate with each other are difficult.

The ICN only models the procedures which does not have predicate constraint. The FFM model, which puts an undue emphasis on the form flow, can not express an office procedure explicitly when it is related with many different forms.

Our design methodology is based on the logic deducing and data modeling which utilizes the inference capability of artificial intelligence.

Chapter 2 Interactive office Automation System

The Interactive Office Automation System (IOAS) involves two aspects: the logic programming which is suitable for system designers to describe and deduce office knowledge, and the interactive interface which is convenient for designers to update office knowledge or portray both office environment and office mission.

2.1 ORAL Language

The ORAL language, a language used for specifying the requirement of an office information system, combines the concepts of logic programming [Kowa 79] and data modeling disciplines [Tsic' 82b]. The semantic data model, together with the principles of aggregation and generalization, offers the abstract mechanism to describe the environment of an office. Also, by using the deductive capability of logic programming, the specification power of the data model is enchanecd.

The details of the ORAL language were described in [Jou 83]. In order to be self-content, some properties of the ORAL language will be given in the following, and some properties are modified and added for more usage.

2.1.1 Declarative and Procedural Interpretation

A term is either a constant, a variable, or a functional expression of the form

f(t1,..., tn);

where t1, ..., and tn are terms, and f is an n-place function name.

An atom is a predicate expression of the form

p(t1, ..., tm);

where p is an m-place predicate symbol, and t1, ..., and tm are terms.

A clause is an expression of the form

B1,..., Bm ⟵ A1, ..., An.

where B1, ..., Bm, A1, ..., An are atoms. The atoms A1, ..., and An are the conditions of the clause, and the atoms B1, ..., and Bm are alternative conclusions of the clause. If the clause contains the variable x1, ..., xk then the interpretation is stated as following:

for all x1, ..., xk

B1, B2 ... or Bm if A1, A2 ... and An.

A sentence is a set of conjunctive clauses.

For example:

Grandparent (x,y) ⟵ Parent (x,z), Parent(z,y)

Male(x), Female(x) ⟵ Parent(x,y)

Ancestor(x,y) ⟵ Parent(x,y)

Ancestor(x,y) ⟵ Ancestor(x, z), Ancestor(x,z)

Terms are abstractions of either physical objects or logic entities. Atoms are predicate expressions with logic values either true or false. An atom represents a proposition about the relationships among some terms which may be

true or false. A sentence is interpreted according to the conjunction of the clauses which the sentence contains. Declarative interpretation of a sentence can be regarded as a true-or-false assignment to every atom which can be constructed from the predicate symbols occurring in the sentence.

The refutational procedure [Nils 80] gives an operational interpretation to a logic program. Using top - down inference, a logic program allows a procedural interpretation which is comparable to the procedure call of the conventional programming languages. Yet with even more distinguished characteristics, such operational interpretation is very suitable for the specifying quality of office procedures.

Procedural interpretation of a Horn clause of the form

$$B \longleftarrow A1, \ldots, An. \quad n \geq 0$$

is interpreted as procedural declaration, where B is called a procedure head. The procedure head identifies the forms of the problems that the procedure intends to solve. (A1, ..., An) is a set of procedure calls constituting the procedure body. Here, B is also called a goal, and A1, ..., An are called subgoals.

Since our first step is to reduce individual goal to subgoal, we therefore start from the goals by repeatedly applying the top-down inferential rule to them. Following the declared procedure and matching the substitutions, the execution mechanism or the interpreter can reach to the

final solution, if there is any.

2.1.2 Type and Relation

A collection of objects with similar properties are categorized into an object type. Common office object types are: form, letter, repositories, personnel, organization units, business events, etc. The object types can be divided into two groups: one is primitive types: the other, non-primitive ones. Primitive types provide basic semantics of specific office applicable domains, including integer, character, graphic, audio, etc. Whereas non-primitive types are added by specializing and aggregating primitive types or other non-primitive types with additional semantic.

By definition, an n-nary relation R is a subset of Cartesian products over domains of n object types. In ORAL language, a relation can be expressed by specifying the name of the relation and the type of each attribute.

The membership condition of attributes which defines the permissible occurrences of tuples in a relation is called the intention of a relation, and a subset of the n-nary cross products dictated by the intention of a relation is called an extension of the relation. Each tuple in the extension denotes a specific fact relationship among objects. The extension is changeable by insertion or deletion of tuples.

Like cardinality constraints of type's extension,

cardinality constraints on the extension of a relation embody some real-world semantics and should be encoded in specification to check the completeness and consistency. We can specify the cardinality constraints in two ways:

(1) Individual cardinality. Attribute: (L..U), where the L and U are the minimum and maximum occurrences of a relation. As the example we have just mentioned, that is, obj: (1..1), it means that obj of private-obj type occurs exactly once. If the minimum or maximum occurrences is not limited, we use the symbol "-" to express L or U respectively.

(2) Mapping cardinality. Attribute 1 : Attribute 2 = (L1..U1) : (L2..U2), where (L2..U2) denotes that each instance of Attribute 1 is mapped by the relation to at least L2 and at most U2 instances of Attribute 2. and so does (L1..U1) mean.

2.1.3 Procedure

Each procedure in an office can be expressed in a process. In fact, a process is a goal statement which contains one or more subgoal statements, that is:

<office procedure> := prc <process id> <goal statement>
where <process id> is the office procedure name.

Like the form: A ⟵ B, C, D. indicates, A is the goal statement and B, C, D are the subgoals. For each subgoal we may have B ⟵ B1,..., Bq, C ⟵ C1, ...,

CK, where B1, ..., Bq is the body of subgoal B. According to this way we define each subgoal as a process, that is, prc B ←——B1, ..., Bq. Following these refined steps, the execution monitor can operate on interpreting these processes to accomplish the goal of the procedure. And each basic operation which can not take any refinement is accomplished by software tools like activity specialist.

Many office activities are either event-driven or message-driven in their nature. The event-driven activities mean the appropriate actions which will be taken only after some events have occurred; whereas the message-driven activities are invoked only when certain messages have arrived.

It is easy to depict the event-driven nature of an office application in the requirement specification as shown in the following:

trap <relation id> (para 1, ..., para n) ←——A1, ..., An. A trap procedure will monitor the designated relation and try to match with a tuple within its extension. If an action affects this relation and the side-effect results in a match, the trap procedure will be triggered. The body of a trap is the same as that of many ordinary procedures. Once raised, the body of a trap will be executed just like any other goal statements.

We specify message-driven activities by using communication predicates:

message <relation id>(para 1, ..,para n) ⟵A1,.An

with-time-bound : duration;

time-bound-proc : B1, ..., Bn

end;

When (A1, ..., An) are put into execution and yet unable to find its conjugate predicate within definite time-bound, the (B1, ..., Bn) will be taken immediately. Such actions may be to put (A1, ..., An) into execution repeatedly or to take other exceptional actions.

Conjugate predicate can be expressed as the following form:

conjugate <relation id 1>, <relation id 2>

2.2 IOAS System Architecture

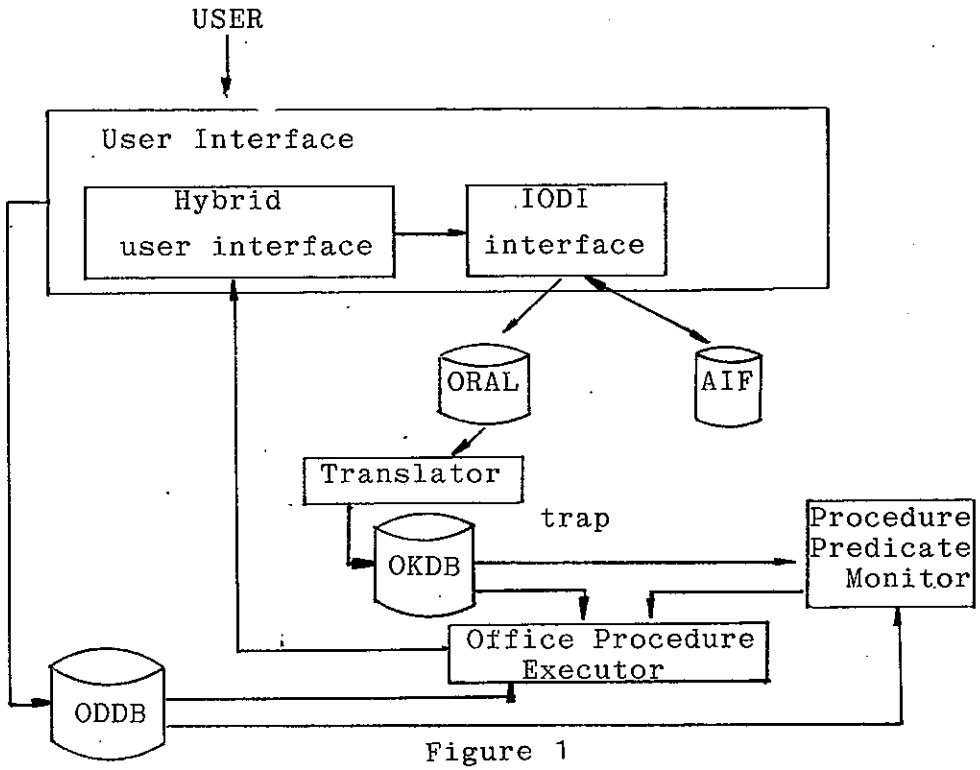


Figure 1

The system architecture of the IOAS system is illustrated in figure 1. It is composed of a User Interface, a Translator, a Procedure Predicate Monitor (PPM), an Office Procedure Executor (OPE), and a Relational Database Management System. The User Interface provides a hybrid user interface which may contain a Form Management System or something else, and an IODI interface. We will give detail discussion about IODI interface later, and the others are described thoroughly in [Jeng 84].

Here, we give an outline of each part. The IODI interface is employed by office designers to generate a specification program expressed in the ORAL language and an Analytic Internal Form (AIF) for consistency checking. The Translator translates the ORAL specification program into internal data which is stored in the relational database. The PPM moniter scans both Office Description Database (ODDB) and Office knowledge Database (OKDB). While an event is driven, the name of the invoked procedure is passed to the OPE, the OPE executor then begins to perform the invoked office procedure using the specific algorithms.

The design methodology is embedded in the IODI interface. Following the interrogating steps of the IODI interface to describe an office information system, a designer can complete his work easily.

The IODI interface is an interface between an office information system and its designers. The I/O configuration

of the IODI interface is illustrated in figure 2:

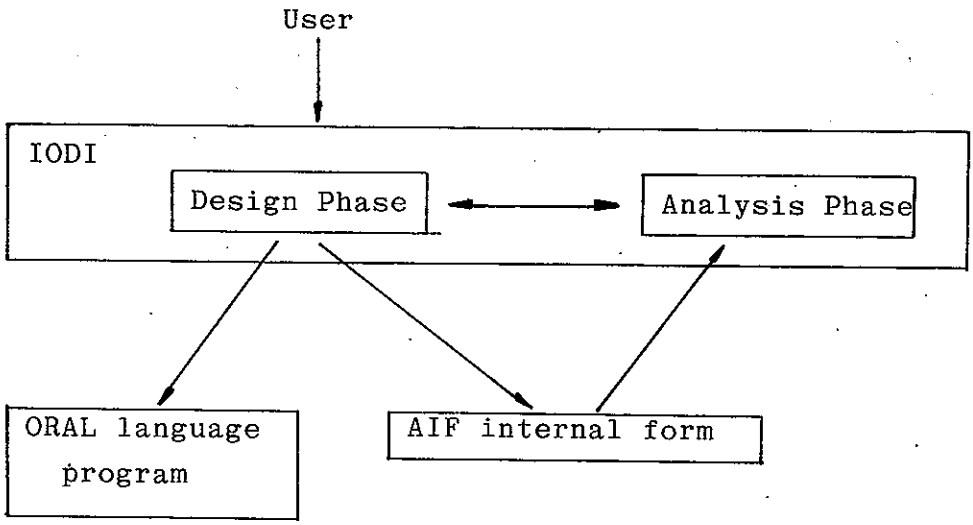


Figure 2

The IODI interface converses with the users through a series of questions, trying to get an abstract view of an office environment and its missions. The users will look at a picturesque office scene free from being bothered by the terminology that computer language uses.

While the design process is ongoing, the IODI interface generates an AIF internal form and uses it to display some illustrative diagrams for expressing the present system architecture. Thus, a user can examine how much work he has done; meanwhile, he is able to correct errors as soon as he finds any. If the users should overlook any fault, the IODI interface will pick it out which is the function of the IODI's consistency checking.

To sum up, four features guiding the design using the IODI interface are:

(1) Interactive to the users, and user-friendly interface.
(2) Real-world view. Through the device of the IODI interface, users can simulate the office activities similar to the ones in the real world.
(3) Semi-automatic office information system design. Following the interrogating steps, the IODI interface provides a semiautomatic method for designing an office information system.
(4) Self-consciousness. The IODI interface seeks to improve the correctness and completeness of an office information system by interactive extraction of the user's knowledge.

The analysis phase uses the AIF internal form as input, and finds out some faults introduced in the design phase.

For strictly typed data model of the ORAL language specification, type consistency checking in this phase is easy. Each office procedure is a process expressed by the ORAL language. By decomposing this process, we may get the goal and the subgoal statements. Accordingly, we can obtain each step for accomplishing the office procedure. It means that, we can prove whether the procedure is correct or not simply by tracing these steps backward.

Chapter 3 Design an OIS System Using the IODI Interface

We can divide office applications into static (environment) and dynamic (mission) aspects. Static aspect involves office organization structure, office-position structure, office policy, and so on. It represents a kind of declaration when a designer describes those components. Dynamic aspect, on the other hand, involves office mission or procedure which is composed of many office operations.

Subsequently, the IOKI interface intends to get an abstract view of office environment and mission through a series of questions.

3.1 Office Environment Design

3.1.1 Organization Structure

When a designer facing the office environment design, the primary consideration must be the office organization. The understanding of the office organization will surely be of great help to the designer when describing an office information system. An organization usually consists of many departments among which, the relations may be either vertically hierarchical or horizontal.

The followings are the describing steps.

(1) Define each department's attribute.

The default type "department" is provided for designer.

```
type department with
        department-code : code, key;
        name : name;
```

```
        address : address;
        tel-no : telephone-no
    end;
```

Designer can make use of this default type to specity a department's type. Besides, he may either add some attributes or define a new type to meet his requirement.

(2) Describe relations among departments.

The "subordinate-to" relation is a default relation:

```
relation subordinate-to(sub:department, spv:department)
    with
      cardinality
          sub:spv = (-..-) : (1..1);
      derivations
          subordinate-to(sub, y: department),
          subordinate-to(y,spv)
    end;
```

(3) Define the attribute's type.

Basically, the system provides some primitive type. But for requirement, the designers may want to define some other attribute types.

(4) Specify the extension of these types and relations.

The IODI interface will examine all the attribute types. If it is not a primitive type, the IODI interface will display it to the designer and then ask for a definition.

There are various ways of "question and answer", all of which conform to acceptable standards; however, some ways are awkward and others elegant. Yet we suggest one of them as an example and illustrate it in appendix A.

3.1.2 Office-Position Structure

Even though in a total automated office, human being is still indispensable because office workers serve as the agents of office procedures. Besides, they are supposed to take the responsibilities of decision making.

In an office-position structure we also have some default types and default relations. The "employee" is our default type. We regard every person in the office organization as an employee. If we examine an office organization, we find a hierarchy of management personnel controlling the office workers. The "direct-under" is our default relation. By using it, we can declare each direct-under (employee-boss) relationship. Also we consider that direct-under relationship has derivative property.

The describing steps of the office-position structure are:

(1) Using the default type "employee" to describe each office-position type of each department.

(2) Secondly, desingner comes to describe the direct-under relation among office-position types. Using the default relation, that is, "direct-under", designer may accomplish this work. In this step, designer also may define

other relations about personnel such as those of "report-to", "team-work", "group-with", and so on.

(3) When designer describes type, some new non-primitive typescan be generated. Under such circumstances, the IODI interface will store and also exhibit those types, acquiring designer to define them.

3.1.3 Office Policy and Regulation

There exist many rules to constrain the office environment configuration. They can set an arbitrary collection of office objects into an orderly way. These rules reflect either the policies of an enterprise and security regulations or restrictions imposed by the surroundings of the office.

We will try to use the logic programming to express the restrictions on office objects. But owing to the lack of a formal form so far, even a rule may have many different ways of expression.

We can express these ruless or regulations as "Promise ←——conditions" which means that if some conditions are satisfied then the promise is asserted. We find that these conditions can be divided into two parts: one involves the events related to quantity; the other involves facts.

For the quantifiable events we may provide some mathematic operators to express them, such as : .GT. (greater), .EQ. (equal), .LE. (less), .GE. (greater or equal), .SUB.

(subtract), .ADD. (add), etc.. For example, the rule "One may get promotion if he joins in the company over three years." can be expressed by

```
get-promotion(x:employee) ←——
    .GT. (.SUB. (Current-date, x.join-date),3)
```

(here, assuming the value of .SUB. (current-date, x.join-date) is represented by the unit 'year'). Following this way we can express the rules related to quantity.

As to the rules involving facts, the condition part of our form can be regarded as the other facts, since as long as these facts are satisfied then the promise may come true. For example, the rule "When the president is absent, the vice-president takes the responsibility of presidency." can be expressed by

```
responsible (x:employee, m:meet) ←——
                president (x:employee, m).
responsible (x:employee, m:meet) ←——
                president(y:employee, m),
                absent (y,m),
                vice-president(x,m).
```

Using this method, the designer can describe all the rules of policy and regulation.

In summary, there are two steps in the office regulation and policy description.

(1) Use the logic programming to represent the policy and regulation.

(2) Define the additional types which are generated at step 1.

3.1.4 Working Facilities Associated With a Position

Most office actions are centered around the manipulation of office facillities which are the entities, such as : messages, documents, letters, forms, repositories, and data files,etc.. In this section, we define the authority of each facility and assign its accesses according to each office position. Such kind of description is called facilities/position assignment.

The description of facilities/position assignment has five steps:

(1) The IODI interface has a default relation "own', so it asks designer to describe each facility type and the cardinality of this relation according to the office-position that designer has described.

(2) A designer may assign other relations among these facilities. the IODI interface can display all the office-position type and facility type to assist a designer in accomplishing his work.

(3) Using the inference capability of the logic programming, a designer may induce some other relations or properties. For example:

```
accessible(e:employee, o:object) ←——
        own(e,o);
        own(x:employee, o), authorize(x,e,o).
```

(4) Now, the IODI interface displays each facility type which the designer just described to assist the designer in defining the facility type.

(5) If there are some other non-existent facilities types, the designer may describe them at this moment.

3.1.5 Duties Associated With a Position

For each office-position; there are some duties or responsibilities which we call office-position behavior. For example : a manager should hold a meeting every weekend and his secretary has the responsibility to inform everyone who is supposed to attend the meeting.

Using the relation declaration or deductive capability of logic programming, a designer can express these behaviors easily.

According to office-position structure, the IODI interface displays each office-position one by one to assist designer in defining their behaviors. The describing steps can also be divided into three parts:

(1) Specify each duty associated with each office-position.

(2) Describe each responsibility induced from some relations or facts.

(3) In the above describing steps, some types may be generated. Accordingly, designer should define them as well.

3.1.6 Current State of Office Environment

Regarding the office processes, some special facilities

which can be manipuiated by those processes are called constants, such as: some special documents or some special meetings which will be refered to in office process. Such special facilities must be declared also.

Before the IOAS system starts working normally on the office information system it must have known all the occurrences existing in all types and relations (i.e. all the extension of type and relation). At this moment, when a designer is asked to enter these occurrences, the IODI interface will display all the attributes in the type and all the parameters in the relation one by one. With such assistance, a designer can accomplish his description efficiently.

Each attribute and each parameter has its own defined type while designer comes to describe them. In accordance with that premise, the IODI interface may take up type consistency checking when designer fills out the occurrences.

To sum up, we get three steps:

(1) Specify the special facilities in the office information system.

(2) Define the additional types which are added to the system in step 1.

(3) Describe all the occurrences of all types and all the tuples of all relation.

3.2 Office Mission Design

3.2.1 Missions

An office consists of people interacting in an environment to carry out the mission of a business, such as : meeting handling, enrollment, procurement, etc. Missions guide the designer in describing a request for the office information system.

A mission can be affected by three components; they are : (1) personnels,(2) procedures,(3) facilities. Personnels are the persons who have the responsibility and duty to trigger and finish this mission or who are involved in this mission. Procedures are the steps to accomplish the mission, and the facilities are those involved in the execution of the procedures.

The IODI interface takes several steps for asking a designer to describe a mission:

(1) If this mission is an event-driven process, the designer describes the trigger condition and entities which will be invoived. And the IODI interface generates a "trap" head to represent it. It the mission is a message-driven process, the designer, on the other hand, must describe time-bound, time-bound-proc and its conjugate process additionally.

(2) When the mission is under execution, it will involve many personnels. So the designer has to describe them at this time.

(3) Furthermore, this mission will also invoive many facili-

ties, so the designer should pick them out for the IODI interface.

3.2.2 Procedures

Since each mission can consist of many procedures (in other words, process), a designer has to describe them in detail. As we have mentioned in section 3.1.3, each procedure, being like a goal, can be reduced into subgoals until it becomes a basic operation. Therefore, we use the form. Goal(...) ⟵ subgoal (...) .,,,, subgoal(...). The IODI interface displays "Goal(...) ⟵" which is the mission name at first time and asks a designer to describe "subgoal (...), ..., subgoal(...)". Then the IODI interface checks each subgoal. If it is not an existent relation, a basic operation, or a procedure head of other procedure, the IODI interface displays "Subgoal(...) ⟵" to request a refining goal for this subgoal.

In this strategy there are two steps:

(1) Employ the inference capability of logic programming to describe the procedures which constitute the mission.

(2) Define the additional facility types or relations that are not described in office environment design.

Chapter 4 Analysis of Office Procedure

According to the AIF internal form, the IODI interface will take up the type checking if some types or relations are not existent, and simulate the procedure execution to find out some contradictions or incorrectness. And the analysis phase will notify the design phase to inform the designer. The analysis phase is simultaneously as active as the design phase. The design phase will ask informations from the designer and put them into the analysis phase which still take up checking. If there is any inconsistency occurring, it will notify the designer through design phase.

4.1 The output of the IODI interface

The IODI interface has two outputs: an ORAL specification program and an AIF internal form. Now, we will explain how to establish them.

In the type declaration, the head "type" can be generated by the IODI interface, and the attributes within the type are filled by the designer. The IODI interface inquires the designer for describing the attributes of this type, and then displays the attributes described for assistance.

The way to declare a relation is the same as the one to declare a type. The IODI interface generate the "relation" head, and then the designer describes each parameter name and corresponding type. After the designer has finished the parameter description, the IODI interface will ask whether cardinality and de-

rivation are necessary or not. The designer has to specify them if necessary.

Because the patterns of the ORAL language are strict, the IODI interface only gets the information from the designer and fills out the patterns. Thus, the ORAL specification program can be accomplished without any trouble.

The And-Or tree diagram is shown in figure 3.

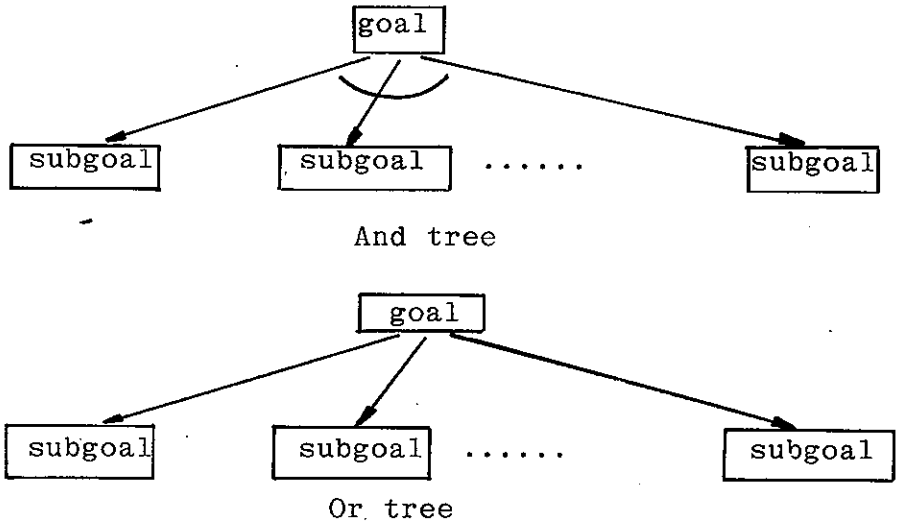


figure 3

The goal presented in the And-tree means that it will be accomplished only when all subgoals are successfully obtained. On the other hand, in the case of Or-tree, the goal will be accomplished as long as any of the subgoals is done.

Using the And-Or tree structure, we may express the deduction rule easily. When a designer inserts or deletes a rule from the office information system, all the IOAS system has to do is add or delete a branch of the And-Or trees.

According to these And-Or trees, we can simulate the procedure execution only by tracing them. Then we may find out some contradictions or incorrectness.

4.2 Analysis Functions

First is the type consistency. When designer describes the office information system, the IODI interface can store all the types that he has described. Therefore, if some types have not been defined exactly, the designer will be informed of describing them.

As to the relation consistency, similarly, if any type of paramenter checked by the IODI interface checks is not declared, the designer will also be informed of describing it. Later, when designer fills in the extension of these relation, the IODI interface may check each type of parameter for consistency.

A goal consists of many subgoals. If the relation among these subgoals is And relation, it represents that all subgoals must be satisfied at the same time. But sometimes they can not be satisfied simultaneously. Uning the inferential rule, we may find out this inconsistency.

We can simulate all the office procedures represented with And-Or tree in the AIF internal form. A result may be found and the IODI interface displays it to the designer to check whether the result corresponds to his requlrement. At each step of this analyzing process, the IODI interface

shows the intermediated result and the deduced rule which it takes to the designer, and asks weather the result is correct or not. The designer may forward the process if the result is correct; otherwise, he ought to terminate it. When he terminates the process, the IODI will require the knowledge for deducing correcting procedure. At this point, the designer may enter the new knowledge or update the old knowledge given by mistake.

Chapter 5 An Example : A Meeting Scheduling

Based on the description of section 4, here we illustrate a meeting scheduling procedure [Byrd82, Jou83] as an example. It may not cover all the features of our interrogating steps in the IODI interface, but it can present an overall view of how the IODI interface works.

A quarterly reporting conference will be called by the director of the office. The meeting scheduling mission can be constituted of five procedures. They are: (1) Notify the secretary of preparing for the meeting when the next meeting-time is upcoming. (2) Find out a non-conflicting time interval from the participants' calendars. In the meantime, a conference room with enough capacity should be reserved for the scheduled period. (3) Draw up a quarterly-meeting-notice. (4) Record this meeting-notice in a log file. (5) Inform all the attendants.

5.1 OFFICE ENVIRONMENT OFESIGN.

(1) Organization structure.

step 1:

```
type department with
        department-code : code, key;
        name : name;
        address : address;
        tel-no : telephone-no
end;
```

```
        type division ISA department;
        type section ISA department;
        type class ISA department;
    step 2:
        relation subordinate-to(s:section, d:division)
        with cardinality
        s:d = (-,.-): 1..1)
        end;
        relation subordinate-to (c:class, s:section) with
        cardinality
        c:s = (-..-): (1..1)
        end;
        relation subordinate-to(sub:department, spv:depart
        ment) with
        derivations
            subordinate-to(sub, x:department),
            subordinate-to(x, spv)
        end;
```

Here the type "department" and the relation "subordinate-to" are our default type and default relation respectively.

(2) Office-position structure.

```
    step 1:
        type employee with
        id : (name, residential), key;
        name : name;
```

```
        residential : address;
        join-date ; date;
        sex : sex;
        marital-status: mar-status;
        title-held : title;
        weight : weight;
        clearance : security
    end;
    type manager ISA employee with
        title-held : (Manager);
        weight : (Important, Very-important );
        clearance : (Confidential, Top-secret)
    end;
    type secretary ISA employee with
        title-held : (Secretary);
        weight : (Low );
        clearance : (Ordinary );
        performance : performance
    end;
    type section-chief ISA employee with
        title-held : (Section-lead);
        weight : (Low, Middle, Important );
        clearance : (Classified, Confidential)
    end;
    type class-lead ISA employee with
```

```
        title-held : (Class-lead );
        weight : (Low, Middle );
        clearance : (Ordinary, Classified)
    end;
    type staff ISA employee with
        title-held : (Employee);
        weight : (Low );
        clearance : (Ordinary );
        performance : performance
    end;
step 2:
    relation direct-under (s:secretary, M:manager) with
       cardinality
        s:m = (1..1) : (1..1)
    end;
    relation direct-under(s:section-chief, m:manager)with
        cardinality
        s:m = (1..-):(1..1)
    end;
    relation direct-under (c:class-lead, s: section-
    chief) with
        cardinality
        c:s = (1..-):(1..1)
    end;
    relation direct-under (s:staff, c:class-lead) with
        cardinality
        s:c = (1..-):(1..1)
```

```
        end;
        relation direct-under (e:employee, b:employee) with
            derivations
                direct-under(e, x:employee),
                direct-under(x,b)
        end;
    step 3:
        type sex = {Male, Female} ;
        type mar-status = { Single, Married};
        type title-held = { Manager, Section-lead, Class-
                            lead, Secretary, Employee};
        type security = { Ordinary, Classified, Confidential,
                          Top-secret};
        type weight = { Low, Middle, Important, Very-import-
                        ant};
        type performance = { Perfect, Good, Fair, Poor, Bad}
```

(3) Office policy and regulation.

Suppose there is only one policy, such as the months when the quarterly meeting is held.

```
    step 1:
        prc quarter-meeting-month(m:month) ←——
                .BELONG-TO.(m, { Jan, Apr, Jul, Oct}).
    step 2:
        type month = { Jan, Feb, Mar, Apr, May, Jun, Jul,
                       Aug, Sep, Oct, Nov, Dec};
```

(4) working facilities associated with a position.

Here assuming that document, repository, private-calendar are the working facilities associated with manager, secretary, employee respectively.

```
step 1:
    relation own (m:manager, d:document) with
        cardinality
            d: (1..1)
    end;
    relation own(s:secretary, r:repository)with
        cardinality
            r:(1..1)
    end;
    relation own(s:secretary, l:location-calendar) with
        cardinality
            l:(1..-)
    end;
    redation own(e:employee, p:private-calendar) with
        cardinality
            p:(1..1)
    end;
step 2:
    relation authorize(sown:secretary, sacp:secretary,
        r:repository)
        cardinality
            sown:sacp = (1..1):(- .-)
    end;
```

```
        relation authorize(sown:secretary, sacp:secretary,
             l:location-calendar)
             cardinality
                  sown:sacp:l=(1..1):(-..-):(1..-)
        end;
        relation authorize(e:employee, s:secretary,
             p:private-calendar)
             cardinality
                  e:s =(1..-):(1..1)
        end;
    step 3:
        prc accessible(s:secretary, r:repository) ←
            own(s,r).
        prc accessible(s:secretary, r:repository) ←
            own(x:secretary,r), authorize(x,s,r).
        prc accessible(s:secretary, l:location-calendar)
             ← own(s,l).
        prc accessible(s:secretary, l:location-calendar) ←
            — own(x:secretary, l), authorize(x,s,l ).
        prc accessible(e:employee, p:private-calendar) ←
            - own(e,p).
        prc accessible(e:employee, p:private-calendar) ←
            - own(x:employee,p), authorize(x,e,p).
    step 4:
        type document ISA text;
        type repository ISA log-file;
```

```
type private-calendar ISA calendar;
type text with
     content:content
end;
type log-file with
     content: content, cardinality (1..-)
end;
type calendar with
     date: date, cardinality (1..-);
     busy-status: busy-status, cardinality
                  date: busy-status =(1..1):(1..1);
     interval : time-interval, cardinality
                  date: interval =(1..1):(1..1)
end;
type busy-status =(Busy, Leisure);
```

step 5:

```
type meeting-notice ISA document with
     name : name;
     date : date;
     loc  : place
end;
```

(5) Duties associated with a position.

In this example, if a manager is an executor, he has to hold the quarterly meeting; also, a secretary under the direction of the executor has the responsibility to schedule this meeting. And those employees whose weight in official

duties is up to certain degree should attend the meeting.

step 1:

relation hold(e:executor, m:meeting) with

cardinality

e:m = (1..1):(1..1)

end;

step 2:

prc attend (e:employee, 'Quarterly-meeting') ←——

.GE. (e.weight, 'Middle').

prc attend(e:employee, 'Annual-meeting') (- - -

.GE.(e.weight, 'Important').

prc responsible-for-schedule(s:secretary, m:meeting

←—— direct-under(s, e:executor), hold(e,m).

step 3:

type executor ISA manager with

weight : (Very-important)

end;

type meeting with

name : name, key;

end:

(6) knowledge for current office environment.

step 1:

const Quarterly-meeting : meeting;

const Quarterly-meeting-notice : name;

const Quarterly-meeting-file : repository;

step 2:

Because there is no additional type generated, IODI may skip this step.

step 3:

division:

department-code	name	address	tel-no
101	PLANS DIVISION	.....	123-1234
102	PERSONNEL DIVISION	.....	123-1235
	⋮		

section :

department-code	name	address	tel-no
201	R and D SECTION	.....	456-1234
202	TECHNIQUE SECTION	.....	456-1235
	⋮		

subordinate-to(s:section, d:division):

section	division
201	101
202	101
	⋮

5.2 OFFICE MISSION DESIGN.

(1) Missions.

The quarterly meeting is an event-driven mission in which the personnels and facilities involved may be described as following:

step 1:

trap At-time(y:year, quarterly-meeting-month(m:
month))

step 2:

personnels : secretary, employee, executor.

step 3:

facilities: private-calendar,
location-calendar,
meeting-notice,
Quarterly-meeting,
Quarterly-meeting-notice,
Quarterly-meeting-file.

(2) Procedures.

step 1:

trap At-time(y:year, quarterly-meeting-month(m:
month)) ⟵ responsibe-for-schedule
(s:secretary, 'Quarterly-meeting'),
schedule(s, 'Quarterly-meeting',
t:time-interval, l: location-calendar,
mt:meeting-date),
create-meeting-notice
('Quarterly-meeting-notice', mt, l,
n:meeting-notice),
deposit(s, n, 'Quarterly-meeting-file'),
attend(e:employee, 'Quarterly-meeting'),
send-to(s,n,e).

```
prc schedule (s:secretary, m:meeting, t:time-inter-
              val, l: location-calendar, mt:meeting-
              date) ←——
     attend(e:employee, m),
     free-time(e,t,mt),
     find-place(mt,t,l),
     .GE.(l.capacity, .SUM. (attend(e, m))),
     reserve-person-time(s,e,mt),
     reserve-loc-time(s,l, mt).
prc free-time(e:employee, t:time-interval,
              mt:meeting-date) ←——
     own(e, p:private-calendar),
     .GE.(p.interval,t),
     .EQ.(p.busy-status, 'Leisure'),
     .SET.(p.date, mt).
prc find-place(mt:meeting-date, t:time-interval,
              l:location-calendar) ←——
     .EQ. (mt, l.date),
     .GE.(l.interval,t),
     .EQ.(l.busy-status, 'Leisure').
prc reserve-person-time(s:secretary, e:employee,
              d:date) ←——
     own(e, p:private-calendar),
     .EQ.(p.date, d),
     accessible(s,p),
```

```
        .SET.('Busy', p.busy-status).
    prc reserve-loc-time(s,secretary, l:location-calen-
                dar, d:date)  <----
        .EQ. (l.date, d),
        accessible(s,l),
        .SET.('Busy', l.busy-status).
    prc create-meeting-notice(m:name, d:date,
                l:location-calendar, n:meeting-notice)
                  <----
          .GENERATE.(n),
          .SET.(m, n.name),
          .SET.(d, n.date),
          .SET.(l,n.loc).
     pro deposit(s:secretary, n:meeting-notice,
                r:repository)  <----
          accessible(s,r),
          .INSERT.(n,r).
     message send-to(send-emp:employee, d:document,
                receive-emp:employee)  <----
               .ROUTE_TO.(d, receive-emp);
          with-time-bound-days:
               2;
          time-bound-proc:
               .ROUTE_TO,(d, receive-emp)
    end.
```

```
    message return-from(send-emp:employee, d:document,
                receive-emp:employee) ←
            .ROUTE_TO.(d, send-emp)
    end.
    conjugate send-to, return-from.
step 2:
    type location-calendar ISA calendar with
        loc-no : code,
        capacity : capacity
    end;
    type meeting-date ISA date;
```

Chaptor 6 Conclusions

An office information system design methodoiogy is developed. Following the steps which are depicted in the methodology an office designer can design his office information system easily and neatly.

The design methodology is based on the semantic oriented data model and the logic programming. The semantic oriented date model is applied to describe office environment in terms of office objects and relationships between these entities; while the logic programming is to formalize office mission in the requirement specification.

The design methodololgy of this office information system obeys the following principles:

(1) The design methodology adopts a real-world view of office systems so that office designers may be willing to accept it and use it with comfort.

(2) The system developed offeres a designer to handle his work through an interactive and intergrated interface.

(3) The system has the features of active and automatic guidance, and the designers need not memorize the designing steps.

Besides, our design methodology also divides office applications into static and dynamic aspects. The designer firstly describes the static components such as: office organization structure, office-position structure, office

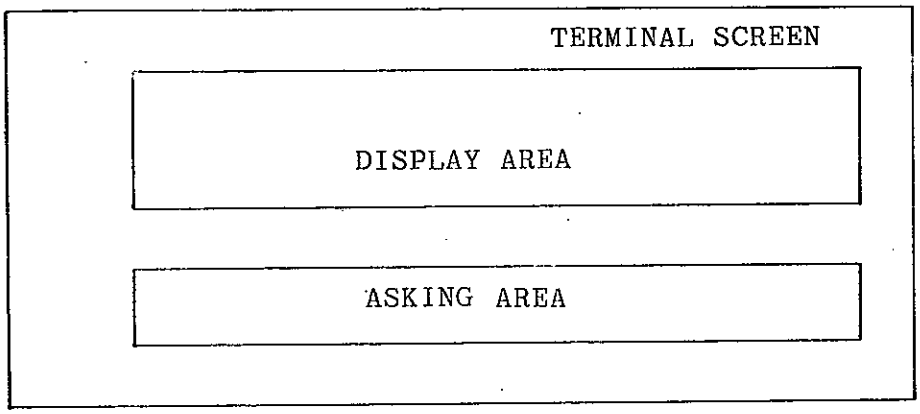
policy, working facilities, and so on. Secondly, he describes the dynamic components such as : office mission and procedures. Owing to our dual partition of office applications, office designer may completely describe any type of information system in detail.

Appendix A

An example describing an Office Organization Structure is illustrated in the following.

By means of "question-and-answer", the IODI interface displays a question and the designer answers it. The asking sequence changes with the designer's answers. In our example the asking sequence is depicted by branches. The conditional branches are indicated by —(x)→ , where (x) is the condition for the branch. The unconditional branches are indicated by ——————→ . If no branch is taken, the item on the next line will be next.

The IODI interface may divide the computer terminal screen into two parts : display area and asking area, as following:



In the display area, the IODI interface displays some informations to assist the designer; while the asking area, the IODI interface displays questions, asking designer to enter his answers.

1.

DISPLAY default type ——— "department" and other existent types. When designer is entering a new type IODI updates this screen dynamically.

2. ENTER THE OEPARTMENT TYPE'S NAME : _____ —(END)→ 7

3. USE THE DEFAULT TYPE'S ATTRIBUTES (Y/N)? _____—(N)⟶ 5

4. ADDITIONAL ATTRIBUTES (Y/N)? _____ ⟵ (N)→ 2

   4.1 attribute's name : _____—(END)→ 2

   4.2 type's name : _____ ⟶4.1

5. USE THE OTHER TYPE (Y/N)? _____—(N)→ 6

   5.1 type name : _____ ⟶ 4

6. DEFINE ATTRIBUTES

   6.1 attribute's name : _____ —(END)→ 2

   6.2 type's name: _____ ⟶ 6.1

7.

DISPLAY all existent types' names.

8. ENTER SUBORDINATE_TO RELATION

   8.1 super-department type : _____ —(END)→ 9

   8.2 subordinate-department type: _____ ⟶ 8.1

9. DEFINE THE OTHER RELATION

   9.1 relational name : _____ —(END)→ 10

   9.2 parameter name : _____ —(END)→ 9.4

   9.3 parameter type : _____ ⟶ 9.2

   9.4 cardinality : _____

   9.5 derivations : _____ ⟶ 9.1

10.

DISPLAY all existent types' names, attributes and relations' names, parameters, cardinalities, derivations.

11. ENTER THE OCCURRENCES OF EACH TYPE (Y/N)? _____—(N)→ 12

    11.1 (IODI displays an existent type name, and then its attribute names.)

    11.2 (According to the attributes of this type, desinger enters the occurrences.)

12. ENTER THE OCCURRENCES OF EACH RELATION (Y/N)?_____

    —(N)→ 13

    12.1 (IODI displays an existent relation name, and then its parameter names.)

    12.2 (According to the parameters of this relation, designer enters the occurrences.)

13.

DISPLAY all the non-primitive data types.

14. (Designer defines each non-primitive data type.)

15. END

REFERENCES

[Baum80] Baumann, L. S. and Coop, R.D.
"Automated workflow control : a key to office productivity"
NCC 1980,pp,549-554

[Byrd82] Byrd, R. J., Smith, S. E. and Peter de Jong, S.
"An actor-based programming system"
ACM 1982,pp.67-78

[Chan82] Chang, J.M. and Chang, S.K.
"Database alerting technique for office activities management"
IEEE Trans. on Comm. Vol.COM-30, No.1, Jan. 1982, pp.74-81

[Cook80] Cook, C. L.
"Streamlining office procedures—an analysis using the information control net model"
NCC 1980,pp.555-565

[Ellis79] Ellis, C.A., Gibbons, R. and Morris, R.
"Office streamlining" in
Integrated Office System-Burotics, IFIP 1980, pp. 111-123

[Ellis80] Ellis, C.A., and Nutt, G.J.
"Office information system and computer science"
Computing Surveys Vol. 12, No. 1, Mar.1980,pp.3-36

[Geha82] Gehani, N.H.
"The potential of forms in office automation"
IEEE Trans. on Comm. Vol.COM-30, No. 1, Jan. 1982, pp. 120-125

[Hamm80] Hammer, M. and Kuin, J.S.

"Design principle of an office specification language"
NCC 1980,pp.541-547

[Ho82] Ho, C. S.
"Office systems modeling, analysis, and design"
Master thesis, Graduate institute of EE, NTU. 1982.

[Jou83] Jou, M. S.
"Office information system requirement specification and analysis"
Master thesis, Graduate institute of EE, NTU.,1983.

[Jeng84] Jeng, P. W.
"Design of interactive office automation system"
Master thesis, Graduate institute of EE, NTU.,1984.

[Kowa79a] Kowalski, R.A.
"Algorithm= logic + control"
CACM Vol.22, NO.7, Jul. 1979,pp.424-436

[Kowa79b] Kowalski, R.A.
"Logic for problem solving"
North Holland 1979.

[Ladd80] Ladd, I. and Tsichritzis, D.C.
"An office form flow model"
NCC 1980,pp.533-539

[Maso83] Mason, R.E.A. and Carey, T.T.
"Prototyping interactive information systems"
CACM Vol. 26, No. 5, May. 1983.

[Mont81] Montgomery, C.A. and Ruspini, E. H.
"The active information system: a data-driven system for the analysis of imprecise data"
IEEE 1981.

[Nau83] Nau, D. S.
"Expert computer systems"
IEEE Computer, Feb. 1983,pp.63-85

[Newm79] Newman, W.
"Office models and office systems design"
Integrated Office System-Burotics, IFIP 1980,
pp.3-10

[Nils80] Nilsson, N. J.
"Principle of artificial intelligence"
Tioga publishning Co.,1980.

[Nutt81] Nutt, G.J. and Ricci, P.A.
"Quinault : an office modeling system"
IEEE Computer, May. 1981,pp.41-57

[Tsic80] Tsichritzis, D.
"OFS: An integrated form management system"
IEEE 1980,pp.161-166

[Tsic82a] Tsichritzis, D.
"Form management"
CACM Vol.25, No.7, July. 1982,pp.453-478

[Tsic82b] Tsichritzis, D.C. and Lochovsky, H. L.
"Data models"
Prentice-Hall, Inc.,1982.

[Zism77] Zisman, M.D.
"Representation, specification and automation of
office procedure"
Ph.D. dissertation, University Pennslyvania 1977.